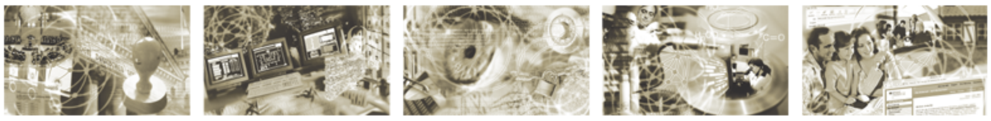
Bundesamt
für Sicherheit in der
Informationstechnik



Group Signatures: Authentication with Privacy

Authors

Prof. Dr. Mark Manulis,
Nils Fleischhacker,
Felix Günther,
Franziskus Kiefer,
Bertram Poettering

Cryptographic Protocols Group
Department of Computer Science
Technische Universität Darmstadt
Mornewegstrasse 30
64293 Darmstadt
GERMANY

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53113 Bonn
Tel.: +49 22899 9582-0
E-Mail: bsi-publikationen@bsi.bund.de
Internet: https://www.bsi.bund.de

Contents

List of Figures

List of Tables

List of Main Symbols and Notations

$x \leftarrow f(y)$	x is the output of $f()$ on input y
$\in$, $\in_R$, $\leftarrow_R$	choice, random choice, assignment of a randomized output
$\lvert x \rvert$	length (in bits) or absolute value of x (context-dependent)
$\mathbb{N}$	set of natural numbers (≥ 0)
$[1, n]$	integers in the interval between 1 and n
$\bmod$	modulo operator
$\mid$	divisor of, e.g. $q \mid (p-1)$ for q divides $p-1$
κ, ℓ, λ	security parameters or lengths
Λ, Γ	integral ranges
p', q', p, q	primes numbers (RSA setting)
N	(safe) RSA modulus
$\mathbb{Z}_N^*$	multiplicative group of $\mathbb{Z}_N$, of $\mathbb{Z}_Q$
$QR(N)$	group of quadratic residues modulo N
P, Q	prime numbers (DL setting and bilinear maps)
$\mathbb{Z}_P$, $\mathbb{Z}_Q$	set of integers modulo P, modulo Q

$\mathbb{Z}_P^*$, $\mathbb{Z}_Q^*$	multiplicative group of $\mathbb{Z}_P$, of $\mathbb{Z}_Q$
$\mathbb{G} = \langle g \rangle$, $1_{\mathbb{G}}$	cyclic group $\mathbb{G}$ with generator g, identity element of $\mathbb{G}$
$(\mathbb{G}_1, \mathbb{G}_2)$, $\mathbb{G}_T$	bilinear (input) groups, target group
$\psi : \mathbb{G}_2 \mapsto \mathbb{G}_1$	homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ (in bilinear groups)
$\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$	a (two-stage) adversary algorithm
st	state information passed between the stages $\mathcal{A}_1$ and $\mathcal{A}_2$
$\mathsf{Expt}^{\mathsf{X}}_{\Pi,\mathcal{A}}$	probabilistic experiment used to define security property X of a cryptographic scheme Π against an adversary $\mathcal{A}$
$\Pr[\cdot]$	probability function
$\mathsf{Adv}^{\mathsf{X}}_{\Pi,\mathcal{A}}$	advantage function used to define success probability of an adversary $\mathcal{A}$ against the security property X of a cryptographic scheme Π
(sk, pk)	private/public key pair (in encryption, signature schemes)
$gmsk$, ik, ok	group manager's secret key, issuing key, opening key
$\boldsymbol{reg}[i]$	registration entry for member i
$\boldsymbol{grt}[i]$	revocation token for member i
gpk	group public key
$\boldsymbol{upd}$	(public) update information used to manage revocation
RL	revocation list
$(\boldsymbol{usk}[i], \boldsymbol{upk}[i])$	private/public key pair of i within a user PKI
$\boldsymbol{gsk}[i]$	secret signing key of member i

Part I.

Group Signatures: Authentication with Privacy

1. Introduction and Background

In the age of digital interaction where synchronous and asynchronous communication of users and exchange of data is increasingly carried out over unprotected public networks, including the Internet, diverse wireless and mobile networks, the ability of an user to prove own identity to the distant communication partner or to claim being the origin of transmitted data is one of the central security goals subsumed under the notion of *authentication.* Cryptography offers manifold techniques for achieving this property in different contexts and with different techniques.

1.1. Authentication with Digital Signatures

One of the most well-known and widely used cryptographic authentication mechanisms is a *digital signature.* It is often viewed as a cryptographic analog of a handwritten signature and belongs to mechanisms that use techniques of *asymmetric* or *public-key cryptography.* We will give a short introduction to digital signatures, highlight their use in public key infrastructures, and discuss some privacy limitations.

1.1.1. Digital Signatures

The basic concept of digital signatures involves a signer and potentially many verifiers. The signer is given the ability to sign arbitrary messages or documents. For this purpose the signer is in possession of some secret information, his *private key* sk, which is usually bound through some strong mathematical relationship to the signer's *public key* pk. This public key uniquely identifies the signer amongst other parties, i.e. pk serves as a *cryptographic identity* of the signer and is used by verifiers to check the validity of digital signatures that were generated by the signer. A digital signature scheme is represented by three main algorithms:

Key generation. The key generation algorithm Kg is executed by the user initially to setup own private/public key pair (sk, pk). The private key sk is kept secret by the user, whereas the corresponding public key pk is made public, for example distributed to all potential verifiers.

Signature generation. A user in possession of a key pair (sk, pk) can apply the signature generation algorithm Sign to produce a digital signature σ on some message m.

Signature verification. Through the verification algorithm Vrfy any verifier can check the validity of a signature σ on the given message m using the public key pk of the purported signer. This algorithm decides whether σ is valid or not.

As in case of handwritten signatures, digital signatures should protect signers from impersonation attacks. The traditional *unforgeability* requirement for digital signatures prevents any other party from generating valid signatures on behalf of some signer. No other party that knows the signer's public key pk and can possibly lure the signer into signing arbitrary messages m, the so-called *chosen message attacks*, may produce a valid signature σ^* on behalf of that signer on some new message m^*.

Digital signatures are already widely used today to ensure authenticity of the signer and his documents, for example to establish authenticated channel to some web server over a public network, or as a tool for enforcing access control to a distant service, network, or any other resource, or as an authorization mechanism for digital transactions, e.g. in online banking, or simply for ensuring the integrity and proving the source of origin for digital documents created and distributed by the signer.

1.1.2. Public Key Infrastructures

The most prominent application domain of digital signatures is in the construction of modern *public key infrastructures (PKI)*, where the primary goal of digital signatures is to establish an authentic link between the *cryptographic identity* (i.e. public key) of some entity, possibly of a human user or some digital device, and other (non-cryptographic) identities (or attributes) of this entity, such as the real name, email address, role within some organization, domain name, etc. This link can be established by the means of certification that reflects the existing trust relationship between the issuer of the digital certificate and the certified entity.

Digital Certificates

In its very basic form a *digital certificate* $\texttt{cert}_{A\to B}$ is a signature generated by A in possession of the key pair (sk_A, pk_A) on a message containing (B, pk_B), i.e. the non-cryptographic identity B and the public key of B. Whenever B presents $(B, pk_B, \texttt{cert}_{A\to B})$ to some third party (verifier) V, the latter can verify, whether $\texttt{cert}_{A\to B}$ is a valid signature of A on the message (B, pk_B) using the public key of A. The underlying idea is that A is trusted by V not to certify invalid links between non-cryptographic identities and public keys. In this case valid certificate $\texttt{cert}_{A\to B}$ would convince V that B is the owner of pk_B, or, in other words, that pk_B is the cryptographic identity of B. This trust put into the digital certificate of B can further be limited in time, for example if A includes some expiration time t and thus signs the message (B, pk_B, t). A may also exclude the non-cryptographic identity of B from the signed message, in which case the issued certificate $\texttt{cert}_{A\to B}$ becomes *anonymous*.

Certification Authorities

In traditional public key infrastructures, the role of certificate issuers is exhibited by trusted *certification authorities* that are often organized into a certification hierarchy, with some *root authority* located at its highest level. Every certification authority $CA^{(i)}$, located at a lower level i, has a public certificate $\texttt{cert}_{CA^{(i-1)}\to CA^{(i)}}$ issued by the certification authority $CA^{(i-1)}$ from a higher level $i-1$. The root authority $CA^{(0)}$ holds *self-certified* certificate $\texttt{cert}_{CA^{(0)}\to CA^{(0)}}$, i.e. it uses own private key $sk_{CA^{(0)}}$ to sign its own identity $CA^{(0)}$ and the public key $pk_{CA^{(0)}}$. The root CA is trusted by all intermediate CAs and by all users of the PKI. Certification authorities located at the lowest level of the hierarchy are typically responsible for issuing the corresponding certificates to the actual PKI users. We observe that certification authorities need not to be organized into a hierarchy, i.e. it is sufficient to have a single CA for setting up the PKI. This CA would act at the same time as root CA and would also issue PKI certificates to the PKI users. Irrespective of how many CAs are involved into a PKI it is ensured that any PKI user A is in possession of the PKI certificate $\texttt{cert}_{CA^{(i)}\to A}$ issued by some certification authority $CA^{(i)}$ of that PKI.

Revocation of Certificates

One of central properties of public key infrastructures is the ability of certification authorities to *revoke* PKI certificates that were issued in the past. There are many reasons, why PKI certificates may need to be revoked, even before the possibly indicated expiration time t. This may happen, for example, once the certified party A can no longer use its secret key sk_A associated to the PKI-certified public key pk_A, because that key was lost, accidentally erased, stolen, or fallen to a cryptanalysis. Also PKI certificate of some certification authority $CA^{(i)}$ may need to be revoked, either for the same reasons as PKI certificates of the users, or because $CA^{(i)}$ can no longer be trusted to perform certification in a correct way.

In general, each certification authority is responsible for the revocation of certificates that were issued by this authority. For this purpose, each $CA^{(i)}$ maintains a corresponding *certificate revocation list* $\texttt{crl}^{(i)}$ that it authenticates to prevent manipulations. This list is updated with unique identifiers (serial numbers) of certificates that were issued earlier and have to be revoked. That is, each certificate issued by $CA^{(i)}$ usually contains some unique identifier and the link to a location, from which $\texttt{crl}^{(i)}$ signed by $CA^{(i)}$ can be obtained. Using the latest version of $\texttt{crl}^{(i)}$ any third party can check, whether some certificate issued by $CA^{(i)}$ has already been revoked, in which case this certificate will be treated as invalid.

Validation of Certificates

Any PKI user A, in possession of a key pair (sk_A, pk_A) and certificate $(A, pk_A, \texttt{cert}_{CA^{(i)}\to A})$ can use sk_A to produce a signature σ on some message m, and send $(m, \sigma, (A, pk_A, \texttt{cert}_{CA^{(i)}\to A}))$ to the potential verifier V. In order to check whether A is the signer of m verifiers will perform several verification checks:

- check validity of σ using the public key pk_A,
- check validity of the certificate $\texttt{cert}_{CA^{(i)}\to A}$ using the public key of the trusted $CA^{(i)}$, and
- check validity of certificates $\texttt{cert}_{CA^{(i-1)}\to CA^{(i)}}$ for all intermediate authorities $CA^{(j)}$ with $j = i-1, \ldots, 0$ using their respective public keys $pk_{CA^{(j)}}$.

1.1.3. Privacy Limitations

Aiming at authenticity of signers and on the establishment of a unique binding between the signer and the authenticated documents, digital signatures, or more generally, PKI-based authentication threatens user privacy. First and foremost, a digital signature σ issued by some user A in possession of a PKI certificate $(A, pk_A, \mathtt{cert}_{CA\to A})$ on some document m reveals the identity A of the signer to all potential verifiers. Furthermore, multiple digital signatures $\sigma_1, \ldots, \sigma_n$ on respective messages $m_1, \ldots, m_n$ produced by the same signer A in different contexts can be linked to that signer and thus reveal more information about A. For example, digital signatures used to authenticate different digital transactions of some user can be misused for profiling purposes.

One may think that the problem of anonymity in PKI-based authentication can be solved with *anonymous certificates*, i.e. where during the certification process some CA verifies the identity A but issues a signature $\mathtt{cert}_{CA\to A}$ on the public key pk_A (rather than on (A, pk_A)). This is, however, not enough since such anonymous certificate would still contain the cryptographic identity of A, namely the public key pk_A. Hence, digital signatures issued by A in possession of an anonymous certificate $(pk_A, \mathtt{cert}_{CA\to A})$ would still remain linkable and traceable to the cryptographic identity of A.

1.2. Group Signatures: Authentication with Privacy

Digital signatures stand in conflict with privacy, in particular with regard to anonymity of signers and unlinkability of issued signatures. On the other hand, their unforgeability authenticates the signer as the origin of the signed document. In order to achieve both authenticity and privacy it appears necessary to decouple public verification procedure from the information that would uniquely identify the signer. This can be done, for example, by assuming a group of potential signers and requiring that verification is performed with respect to the whole group.

1.2.1. Group-based Authentication

In the *group-based authentication* approach users can authenticate themselves on behalf of some group, rather than on the individual basis. That is, the authentication process does not disclose any information that could be used to identify some particular user. Since all disclosed information can only be linked to some group of users, group-based authentication is a suitable approach for achieving user privacy. With this approach users are considered as being authenticated if they can provide a proof of the group membership. Note that group-based authentication is often used for the purpose of access control, where individuals are often assigned to groups and permissions to access and operate on certain resources is granted based on these assignments. In our context we are interested however in group-based authentication techniques applied to digital signatures.

1.2.2. Concept of Group Signatures

The concept of *group signatures*, introduced by Chaum and van Heyst [71], adopts group-based authentication to achieve privacy of signers against potential verifiers. At a high level, group signatures implement the following idea: All potential signers are considered as members of some group. Each signer can issue a signature on behalf of the whole group. Such group signature is publicly verifiable using the public key of the entire group, which provides anonymity of the actual signer. However, there exists a dedicated, possibly trusted party, which can link the group signature to the identity of the signer. We now provide a more detailed view on the architecture behind group signatures and give a brief comparison to the classical PKI authentication.

Group Manager and Group Members

The architecture of a group signature scheme consists of the *group manager* and multiple *group members*. The group manager, which can either be a single authority or a coalition of several entities, is responsible for the initialization of the group, for the admission and in some schemes also for the revocation of group members. During the initialization process the group manager chooses own secret key and defines public group parameters containing the *group public key*. Once the group parameters are established the group manager can use own secret key to issue membership certificates to the prospective group members. In some schemes the group manager can further use own secret key to *revoke* existing group members from the group.

Each group member is in possession of a membership certificate issued by the group manager. This certificate represents the *secret signing key* of the respective group member. That is, each group member can use it to produce group signatures on arbitrary messages. Any verifier can publicly check the validity of some issued group signature using the group public key. The group signature thus proves that the signer belongs to the group.

The distinguished property of group signatures is that the group manager can *open* group signatures and identify their signers using the information collected during the admission process.

In comparison to ordinary digital signatures, group signatures have extended security goals. In particular, the unforgeability requirement ensures that only group members are able to issue valid group signatures. In addition, group signatures provide privacy by requiring that no other party, except for the manager of the group, should be able to identify the actual signer. Furthermore, group signatures should remain unlinkable, meaning that no party, except for the group manager, can link two or more signatures produced by the same signer. Also the opening procedure performed by the group manager implies security requirements of its own to protect a group member from malicious accusations of having produced some group signature if this was not the case.

Differences to Digital Signatures and PKI-based Authentication

Authentication with group signatures reminds of PKI-based authentication, yet with some significant differences. One can see the role of the group manager as being related to that of a PKI certification authority. Indeed, the group manager issues membership certificates to new users and in some schemes also revokes their membership. The main difference to PKI certification is that membership certificates issued by the group manager are confidential and should never be disclosed by the corresponding group member.

Another difference to traditional digital signatures is that group members do not need to possess any public keys since verification of group signatures is performed with respect to the public key of the entire group, defined by the group manager during the initialization procedure. Observe that group signatures are nonetheless publicly verifiable.

The most important difference between ordinary signatures and group signatures are the additional privacy guarantees offered by the group signature schemes: namely, their (public) verification procedure does not leak any information about the actual signer. The verifier is only convinced that the signer is a valid member of the group. This can be seen as a relaxation of the authentication goals offered by traditional signatures, which allow verifiers to uniquely identify the signer of a given signature. In group signatures this property is replaced with the ability of the group manager to open group signatures. That is, although publicly verifiable group signatures do not disclose their signers, any verifier is assured that signers can be identified by the group manager. Note that by requiring the (trusted) group manager to open all group signatures we would immediately obtain the functionality of traditional signature schemes. This shows intuitively that group signature scheme have richer functionality and more versatile applicability than traditional signature schemes.

1.2.3. Applications of Group Signatures

Group signatures find applications in scenarios where verifiers shouldn't learn the actual identity of the signer and are willing to accept signatures that can be verifiably attributed to some member of the group, knowing that the signer can be identified by the group manager, if this becomes necessary.

The most popular application of group signatures is to *conceal organizational structures*: For example, employees of a company may be trusted to sign contracts, issue orders or press releases, participate in public tenders, and authorize financial transactions on behalf of their company. If group signatures are deployed for this purpose then the actual identity of the employee will not be disclosed. However, if some employee abuses this trust, for example by authorizing a risky transaction or by signing documents that would damage the reputation of the company, then this employee can be identified using the opening mechanism of the group signature scheme. That is, the opening procedure offers reactive form of protection for companies and organizations in case their members abuse the initial trust, granted through allocation of signing rights. The opening procedure could then trigger punishment actions against such members, typically leading to revocation of those signing rights.

Further applications of group signatures can be found in anonymous online communications. For example, group signature schemes can be used in *identity escrow schemes* [124] (a.k.a. as group identification [126]), which allows anonymous communication between two parties, unless one party misbehaves, in which case its anonymity can be revoked by some trusted authority. In fact, group identification schemes can be seen as an interactive counter-part of group signature scheme, in which the signature generation process is turned into the identification process that protects the anonymity of users, who identify themselves to some third parties. Lee, Smart, and Warinschi [126] showed that group signatures can be constructed from group identification schemes using general transformation techniques that were previously applied, e.g. in [89, 1], for the conversion of (non-private) identification schemes into traditional digital signatures.

Group signatures find further application in various e-commerce scenarios. For example, group signatures and underlying techniques have been used in e-cash schemes [174, 136, 63] aiming to protect privacy of users that perform transactions with electronic money, e.g. to prevent profiling of those users. Other e-commerce applications of group signatures include digital auctions [150] for protecting privacy of the bidders and digital voting schemes [62] where voters should be able to cast votes anonymously. Group signing techniques are of prime importance for the design of anonymous credential systems [50] used to support privacy in identity management, and they have further been used in privacy-preserving remote attestation protocols for computing platforms [44].

Furthermore, group signatures have applications in the domain of digital rights management, where they can be used for anonymous fingerprinting [46] to protect privacy of buyers of digital goods, yet allowing identification of distributors of illegal copies. Group signing techniques can also be found in traitor tracing schemes [118] that help to identify pirates in broadcast-based content distribution systems.

1.3. Classification of Group Signature Schemes

Group signature schemes can be classified based on their functionality. As previously mentioned, common to all such schemes is the ability of the signer, while being a member of the group, to generate group signatures that can be publicly verified using the group public key and that do not leak any information about the signer's identity. The only party that can revoke signer's anonymity is the group manager. This basic concept gives rise to different flavors of group signature schemes, depending on the optional support for the following set of actions:

- the ability of the group manager to dynamically admit new group members and/or revoke previously granted membership,
- the ability of the group manager to provide publicly verifiable proofs that some group signature opens to a concrete signer, and
- support for the distribution of the group manager's duties amongst several entities: (i) an *issuer* being responsible for the sole management of the group membership, and (ii) an *opener* being equipped with sole rights to open signatures and identify the signer.

Our classification of group signature schemes in the following Sections 1.3.1 to 1.3.4 aims to address support for the above listed functionality properties and give a high-level overview of the corresponding algorithms and their roles. This classification will serve as a basis for our more detailed description of existing group signature schemes and their security properties. We observe that our classification captures the majority of existing group signature schemes.

In Section 1.3.5 we will also mention several further types of group signature schemes, whose properties are too specific to be addressed in our classification. We will limit the description of those schemes to the high-level overview of their core properties, without detailing their constructions in the remaining part of this work.

1.3.1. Static Group Signatures

We start with *static* group signature schemes, where the number of group members is assumed to be fixed during the initialization stage. This stage includes the computation of secret signing keys for each member by the group manager. At a high level static schemes contain algorithms for key generation, signing and verification, and the opening procedure that identifies the signer. They involve only one group manager, which takes care of computing the secret signing keys of prospective group members and of opening their group signatures. Static schemes have the following four main algorithms as also illustrated in Figure 1.1:

Key generation. The key generation algorithm executed by the group manager will be denoted by GKg. In static schemes this algorithm generates public key of the group, private key of the group manager allowing the latter to open group signatures, and a (personal) secret signing key for each member of the group.

Signature generation. Each group member, in possession of her (personal) secret signing key can issue group signatures using the group signing algorithm, which we denote GSign.

Signature verification. The validity of an issued group signature on some message can be checked using the verification algorithm GVrfy. This algorithm is public in that it can be executed by any party using the public group key generated by the manager.

Opening procedure. In case of dispute the group manager can identify the signer of some (valid) group signature using the opening algorithm, which we denote by Open. This algorithm can only be executed by the group manager using the secret key of the latter.

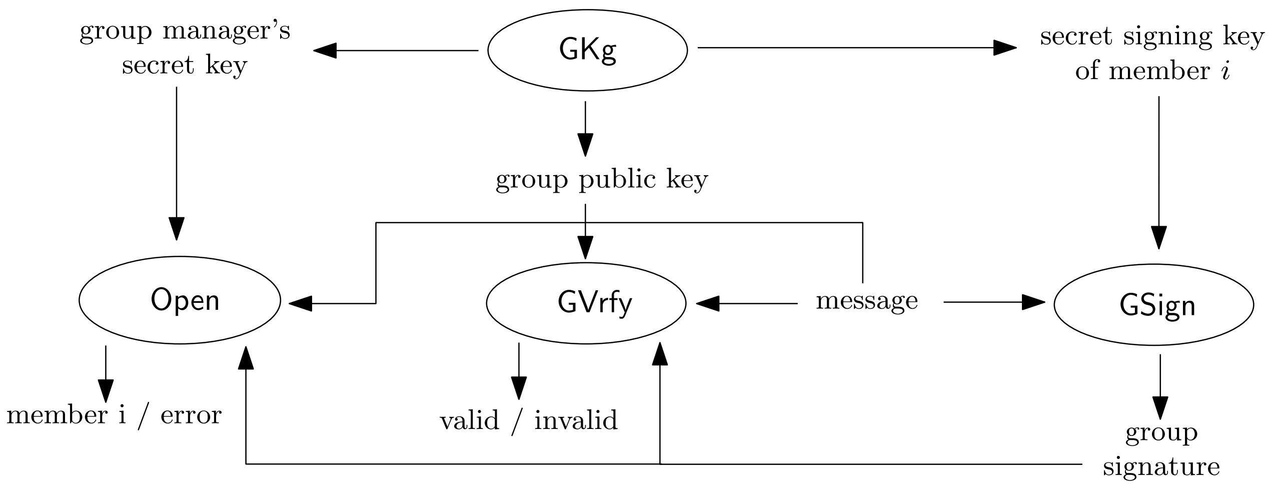


Figure 1.1.: Static Group Signatures

Observe that this functionality, described by the algorithms GKg, GSign, GVrfy, and Open, is in fact basic for all group signature schemes. Other flavors use slight modifications and extensions of these algorithms as well as new algorithms, which then reflect the additional functionality offered by those schemes and may have further impact on their security properties.

1.3.2. Dynamic Group Signatures

The requirement to fix the number of group members in advance and generate their signing secret keys during the initial key generation procedure may not be appropriate for applications where no prior knowledge about prospective group members exists. In this case it is desirable to provide support for the on-demand admission of new members by the group manager. This property is satisfied by *dynamic* group signature schemes through the additional protocol executed between the group manager and the joining group member. In addition to this protocol the key generation procedure has to be modified such that it does not output any secret signing keys. Other procedures such as generation of group signatures, their public verification, and opening by the group manager are performed in the same way as in static schemes.

Key generation. In dynamic schemes the key generation algorithm GKg executed by the group manager generates only the public key of the group and the private key of the group manager.

Join protocol. Every dynamic group signature scheme offers a protocol Join executed between the group manager and the prospective group member. At the end of this protocol the admitted group member receives its secret signing key whereas the group manager obtains some (secret) information that will be used later to open group signatures produced by the new member.

This functionality described by five algorithms/protocols GKg, Join, GSign, GVrfy, and Open is typical for dynamic schemes and is illustrated in Figure 1.2.

Dynamic group signature schemes may further provide support for *membership revocation* aiming to prevent former group members from issuing valid group signatures. This is typically handled through the following additional algorithms, where the group manager, which is responsible for revocation, publishes some update information that is then used by either unrevoked members to update their secret signing keys and/or by verifiers as an additional input to the verification procedure.

Revocation procedure. The group manager can revoke members using the revocation algorithm denoted Revoke. This algorithms typically updates some public group information to indicate that some member was revoked.

Update procedure. Each of the remaining (unrevoked) group members can update own secret signing key using a special algorithm, which we call UpdM.

In general, requiring unrevoked members to update their secret signing keys after each revocation step is not the best solution for practice. Therefore, some group signature schemes handle

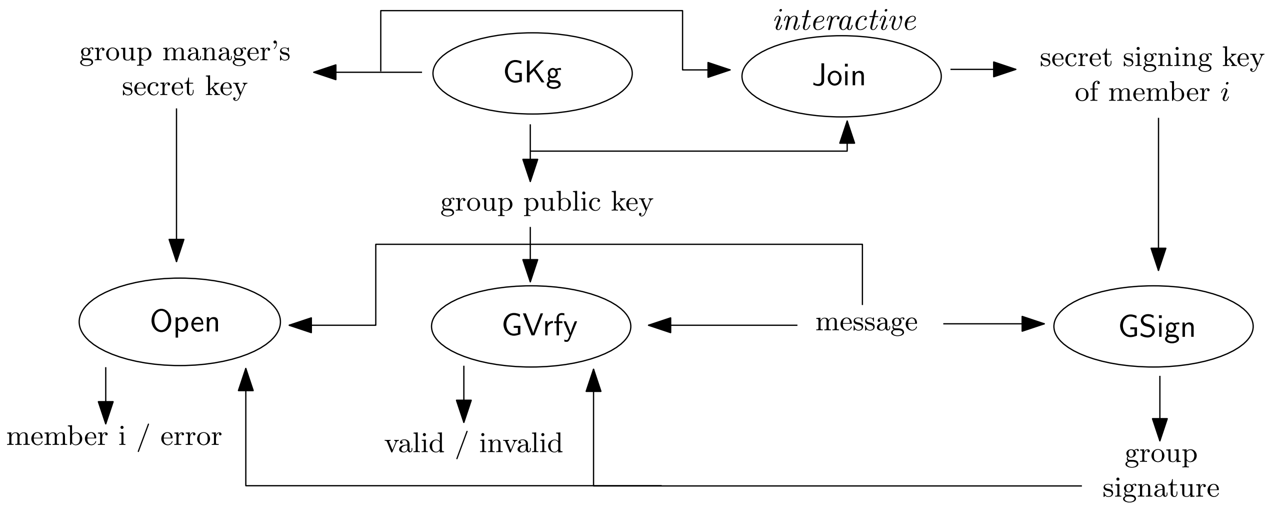


Figure 1.2.: Dynamic Group Signatures

membership revocation in a slightly different way: in particular, group signature schemes with the property of *verifier-local revocation*, require the group manager to update only public group information, which in turn is used by verifiers to check locally whether a given signature was issued by some revoked signer. This property is typically realized using revocation lists, akin to CRLs used in public key infrastructures.

We also note that membership revocation can be a useful property in static schemes, where the entire group of members is determined during the key generation procedure. In such schemes the group manager could still decide to revoke group members at a later stage.

1.3.3. Group Signatures with Verifiable Opening

The distinguished property of group signatures is to preserve anonymity of signers, yet allow their identification by the group manager through the corresponding opening procedure Open. This procedure is usually seen as a fall-back mechanism, which should be invoked in (rare) cases that require identification and possibly subsequent punishment of the signer. The basic functionality of group signature schemes does not prevent the group manager from falsely accusing some particular signer of having produced the signature. In particular, the group manager may not always be trusted to perform the opening procedure in a correct way. In order to account for these risks many group signature schemes provide additional support for the *public verifiability* of the opening procedure. That is the group manager is required not only to output the identity of the signer but also to provide some publicly verifiable proof that the output identity indeed belongs to the signer who produced the signature. This functionality is typically obtained by modifying the opening procedure to output the appropriate proof and by introducing another verification procedure that can be used to judge whether the identified signer and provided proof are valid.

Opening procedure. In group signature schemes with verifiable opening the group manager upon execution of the opening procedure Open on some given group signature outputs not only the claimed identity of the signer but also a proof to support the claim.

Judgement procedure. Verifiable opening implies existence of a public judgement procedure, denoted Judge, which using public group parameters checks whether the group manager's proof output in the opening procedure with respect to the claimed signer is valid.

Considering the above modifications group signature schemes with verifiable opening contain at least the algorithms GKg, GSign, GVrfy, Open, and Judge. The main differences with regard to the opening and judgement procedures are illustrated in Figure 1.3.

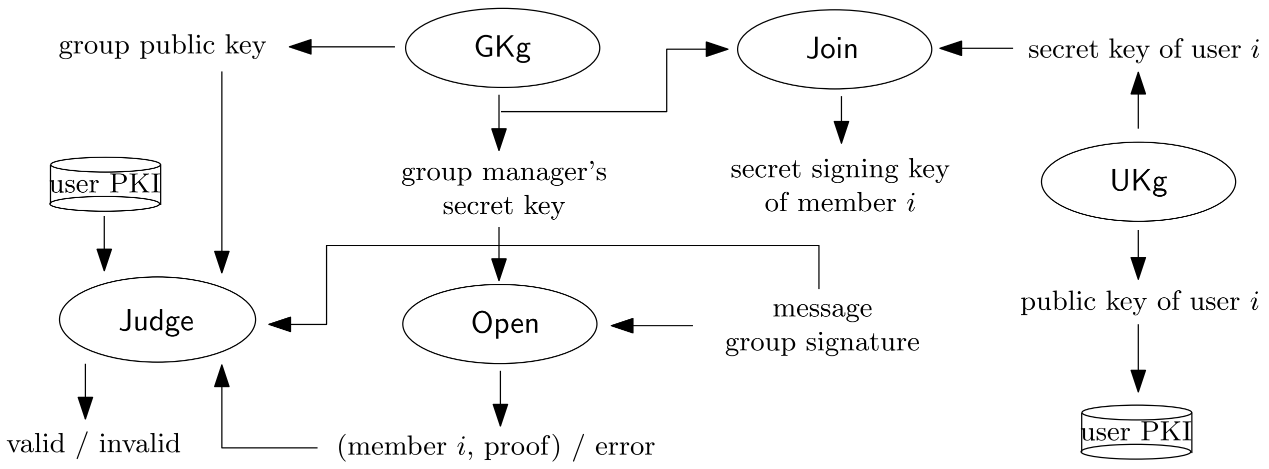


Figure 1.3.: Verifiable Opening and User PKI

One of important elements in the construction of group signatures with verifiable opening is the use of public key infrastructures (PKIs) to allow for unique identification of signers. Many schemes with verifiable opening implicitly assume that certified public keys of users are linked to their membership credentials and that in dynamic schemes users use their PKI-certified key pair during the execution of the joining protocol with the group manager. Many modern group signature schemes, however, explicitly define *user PKI* as part of the group signature setting and consider its impact on the security of the scheme. In this case a group signature scheme with verifiable opening may contain the following additional procedure for the generation of PKI-certified user keys.

User key generation. A group signature scheme with verifiable opening that explicitly uses a PKI for its users allows each user to generate own private/public key pair using the appropriate key generation procedure denoted UKg.

This additional UKg procedure can be seen as a simplified way to adopt a user PKI into the setting of group signatures. In practice, UKg will likely be realized using the common registration procedure of an user inside some PKI (cf. Section 1.1.2). The management of the user PKI would then be performed independently from the management of the group. In particular, the group manager role in the group signature scheme and the certification authority role in the PKI setting will typically be executed by different parties, independently of each other.

1.3.4. Group Signatures with Distributed Authorities

Another flavor of group signatures comes from the separation of group manager's duties amongst different authorities. This helps to mitigate the amount of trust placed into the group manager and suits applications where such trust is either not available or where a separation of duties is necessary from the architecture point of view. There exist two main tasks that a single group manager is supposed to take care of. The first task concerns management of the group membership. In static schemes this includes generation of secret signing keys, whereas in dynamic schemes the group manager admits and possibly revokes group members. The second task concerns the ability of the group manager to identify signers by opening their signatures.

Intuitively, the two tasks can be separated amongst two authorities. This separation is foundational for group signature schemes with *distributed authorities*. The first authority, which is responsible for the management of the group membership is typically called an *issuer*, whereas the second authority in possession of opening rights is called an *opener*. From the functionality point of view, schemes with distributed authorities assume that the issuer is involved into the generation of secret signing keys, either during the key generation algorithm GKg (in static schemes) or during the admission protocol Join (in dynamic schemes), while the opener is the only party being able to execute the opening procedure Open, as illustrated on the example of dynamic schemes in Figure 1.4. Such separation of duties is also reflected in the security properties of these schemes. Namely, the issuer should not be able to identify signers, whereas the opener should not be able to grant or revoke membership credentials.

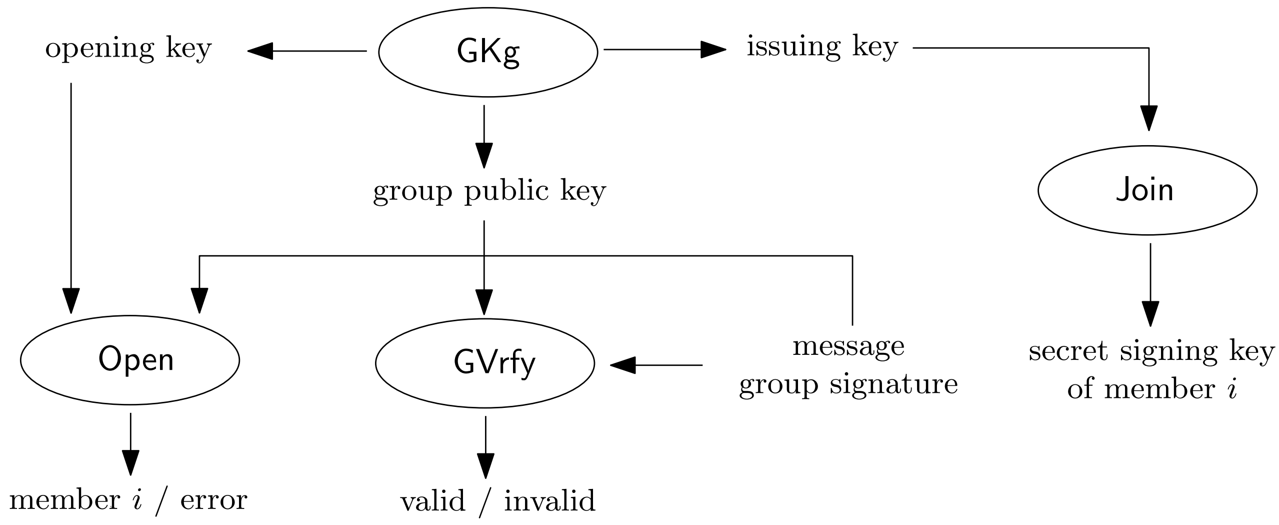


Figure 1.4.: Group Signatures with Distributed Authorities

In general, the functionality of distributed authorities can be initialized using the slightly modified key generation procedure:

- **Distributed key generation.** The distributed key generation algorithm GKg generates the public key of the group, the private key of the issuer allowing the latter to admit and possibly revoke group members, and the private key of the opener, allowing the latter to open group signatures and identify the signer. In static schemes such GKg would further generate individual secret signing keys for all members of the group.

In practice, distributed key generation algorithm may be split in two algorithms, one for the issuer and another one for the opener, allowing both parties to generate their private keys locally. An alternative approach would be to require some third trusted party to generate both types of private keys in advance and then hand the keys to the issuer resp. the opener using secure channels.

1.3.5. Group Signatures with Special Properties

Here, for completeness, we will mention several types of group signature schemes that do not fall into our general classification above. These schemes are specialized constructions, whose functionality and security properties are not directly comparable with the majority of existing group signature schemes.

Group Blind Signatures

A *group blind signature scheme* [134] corresponds to a group signature scheme where the signing process becomes interactive in the following sense: the message m is chosen by a third party and the generation of the group signature is performed such that the signer does not see the message in clear, while the third party does not know the identity of the signer. At the end of the interaction the third party obtains a valid group signature on m whereas the signer does not receive any information about m nor about the resulting value of the group signature.

Group blind signatures have been proposed for applications where the group signature generation procedure should remain unlinkable to the message and the issued group signature. Imagine for example that the role of the signer is performed by some bank and that signed message corresponds to an electronic coin. The bank is part of a larger financial group. The generation process of the blind group signature would correspond in this case to the withdrawal of the electronic cash by some bank's customer. This customer can use the withdrawn e-cash in later digital transactions, without revealing information about the bank that issued the coin and without allowing the issuing bank to keep track on how the customer used the e-cash.

Democratic Group Signatures

In a *democratic group signature scheme* [137, 143] the dedicated role of the group manager is eliminated and corresponding tasks are carried out by the members of the group in the following sense: the admission and revocation of group members is decided and executed jointly by all (current) members of the group, whereas the opening procedure can be performed on an individual basis by each group member alone. In this way, democratic group signatures offer anonymity against non-members only, while allowing each current member to individually keep track on the other group members' actions (i.e. issued signatures). Some democratic group signature schemes can further restrict the execution of the opening procedure to a certain fraction of group members [185] or allow the signer to specify a subset of group members who can open the signature in a collective manner [130].

An example application of democratic group signatures is a joint venture company, formed by several, possibly anonymous, parent companies. Each parent company may act in the name of the joint venture company and can be held accountable for its actions by other parent companies without relying on some dedicated party.

Mediated Group Signatures

In a *mediated group signature scheme* [83, 84] the traditional setting of group signatures is extended with an additional party, called a mediation server. This server assists group members in the generation process of group signatures and performs revocation of group members on group manager's request. In order to issue a group signature the signer identifies himself to the mediation server and submits a “partial” signature which is then transformed by the mediation server into a full group signature under the condition that the signer is not revoked. This approach allows for *immediate revocation* of group signatures, i.e. the mediation server prevents generation of group signatures on behalf of revoked members.

Mediated group signatures were introduced in the context of another security property, called *leak-freedom*, which is supposed to prevent the signer from convincing third parties of having generated some given group signature in the past. As an example application, consider an employee, who is allowed to sign purchase orders on behalf of the company. A misbehaving employee may attempt to convince some supplier company of having signed a particular purchase order in order to gain personal benefits in return. The leak-freedom property would prevent employees from abusing their signing rights.

1.4. Intuitive Security Requirements for Group Signatures

Group signatures extend the basic concept of digital signatures used primarily to achieve authenticity towards anonymity of signers and the ability of the group manager to identify the signer. Therefore, group signatures not only inherit classical security goals of ordinary digital signatures, such as unforgeability, but also introduce new requirements. Additional complexity in handling the security requirements for group signatures stems from the involvement of different parties with their own security goals. For example, group members acting as signers wish to preserve their anonymity against verifiers that in turn must be ensured that valid signatures were produced by existing group members who can be identified by the group manager. In the following we will informally discuss various security requirements stated in the early literature on group signatures, at time when no formal security models for these schemes were available.

1.4.1. Unforgeability of Signatures

The unforgeability property of group signatures is widely similar to ordinary digital signatures and was already mentioned by Chaum and van Heyst [71] in their seminal paper on group signatures. This property prevents generation of a valid message-signature pair without knowledge of corresponding private keys. In the context of group signatures this property would assume that secret signing keys of group members (and of group managers) remain unknown to the potential forger. This requirement typically considers *chosen-message attacks*, where the forger may learn valid group signatures for any messages of its choice, produced by any valid group member of its choice.

1.4.2. Exculpability

The requirement of exculpability was first introduced by Ateniese and Tsudik [17] as another flavor of unforgeability. It requires that no member of the group and not even the group manager can produce some valid message-signature pair on behalf of another group member. Since valid group signatures can be computed by any member of the group, exculpability offers security with respect to the opening procedure — it ensures that no group member can be accused of having generated some (valid) group signature if this was not the case. This requirement is quite strong since it also provides security against malicious group managers. In schemes with verifiable opening where the group manager outputs an additional proof that a group signature was produced by the claimed signer, exculpability would prevent the group manager from cheating in the generation of this proof and thus, offer additional security for the judgement procedure.

1.4.3. Traceability of Signers

Chaum and van Heyst [71] considered traceability as a property of group signatures allowing the group manager to identify the signer of a given signature. This property can also be seen as a security requirement to ensure that any valid group signature must be openable by the group manager. Hence, traceability prevents attacks where some output group signature passes the verification procedure, yet its opening by the group manager fails. In this respect the term “failure” may have two meanings — either the opening procedure does not output any identity at all, or it outputs some identity, which is, however, not the identity of the actual signer. Traceability thus protects interests of the group manager and potential verifiers of group signatures.

1.4.4. Coalition Resistance

Ateniese et al. [11] extended the requirement of traceability towards a stronger attacker model, where they considered coalitions of malicious group members combining their secret signing keys to produce an untraceable group signature. Coalition resistance guarantees that if some group signature passes the verification procedure then it is always openable to *at least* one member of the coalition. This requirement thus protects other (honest) group members from false accusations and further provides similar guarantees to the group manager and verifiers as the previous traceability requirement.

1.4.5. Protection against Framing Attacks

A slightly different formulation of coalition resistance is protection against framing attacks, first considered by Chen and Pedersen [73]. In a framing attack a coalition of malicious group members works together with the malicious group manager to generate a group signature, which passes the verification procedure, yet opens to another member of the group. These attacks are typically relevant for schemes with verifiable opening where the judgement procedure could accept some falsely accused group member as a signer. Hence, protection against framing attacks can be seen as an individual security goal of any (honest) group member.

1.4.6. Anonymity of Signers

Chaum and van Heyst [71] considered anonymity of signers as a core privacy property of group signatures. This requirement implies that no party except for the group manager is able to identify the signer of some given group signature. Adversaries against anonymity could be not only potential verifiers but also other members of the group. The latter makes sense if the group includes at least three members so that any member who did not sign some particular message would still have to decide, which of the at least two remaining group members produced the signature.

1.4.7. Unlinkability of Signatures

The requirement of unlinkability, addressed by Bellare, Micciancio, and Warinschi [22], is related to anonymity in that it prevents correlation amongst signatures produced by the same signer. An adversary against unlinkability should not be able to decide whether two or more group signatures were signed by the same member of the group. It is also possible to consider other group members as potential adversaries against the unlinkability with similar restrictions on the minimal number of group members as in case of anonymity. Clearly if the adversary can break anonymity and identify the signer then this adversary could also link group signatures produced by that signer.

1.5. Group Signatures and Provable Security

In modern cryptography it is important to argue about security of cryptographic schemes by giving convincing *security proofs*. In addition to the efficiency considerations and practicality of the scheme, existence of a valid security proof belongs to the main factors influencing the deployment of the scheme in practice. Security proofs require some suitable formalism that is able to capture the desired functionality of the scheme and define its intuitive security goals. An abstract description of the scheme's functionality, including its algorithms, their inputs and outputs, is usually referred to as the *syntax* of the scheme. By combining this syntax with formal specification of security goals one obtains the so-called *security model*. That is security proofs are always given with respect to a specific security model, meaning that changes in the security model may invalidate the proofs. Security models can change for many reasons. For example, extension of the functionality of the scheme, such as extension of static group signature schemes with dynamic behavior, will ultimately result in the update of the security model to cope with the new properties and security threats that these properties introduce. In fact, our classification of group signature schemes according to different functionalities and flavors will contain respective security models and definitions.

1.5.1. Computational Security and Adversarial Experiments

In the following we will provide a high-level intuition on how security in modern cryptographic schemes is modeled. Security definitions typically assume an *adversary* $\mathcal{A}$ seeking to break some security property X of a cryptographic scheme Π. In practice, $\mathcal{A}$ will likely be required to compute a solution for some — presumably hard — problem and therefore require computing power to perform its attack. The notion of *computational security* assumes that this power is not infinite, yet also not constant, and may increase in time along with the progress in the area of computing technology. This stands in contrast to the notion of *information-theoretic security*, which considers unbounded adversaries with infinite computing resources. Although information-theoretic security is the strongest form of security a scheme can achieve, it is, usually, not practical. In the setting of computational security adversaries $\mathcal{A}$ are modeled as *probabilistic, polynomial-time (PPT)* algorithms. The running time of $\mathcal{A}$ is polynomially bounded in the (public) *security parameter* 1^κ, $\kappa \in \mathbb{N}$, meaning that $\mathcal{A}$ accomplishes its attack within time $a \cdot \kappa^c$, for some suitable constants $a, c > 0$. In addition, all algorithms of the scheme Π are assumed to have polynomial running time in the same parameter 1^κ, so that security properties of Π should hold in the presence of adversaries that have at least the same computational power as the computing devices that will execute the algorithms of Π. By carefully increasing the security parameter 1^κ it is possible to keep pace with the technology progress and protect security of the scheme as time passes by.

The actual security property X of the cryptographic scheme Π is usually defined through some *probabilistic experiment* $\mathsf{Expt}^{\mathrm{X}}_{\Pi,\mathcal{A}}(1^\kappa)$ which interacts with the adversary $\mathcal{A}$ and outputs 1 if the attack of $\mathcal{A}$ was successful and 0 otherwise. Depending on the property X the experiment $\mathsf{Expt}^{\mathrm{X}}_{\Pi,\mathcal{A}}(1^\kappa)$ can provide $\mathcal{A}$ with some further information. This information can be given to $\mathcal{A}$ either directly as input or it can be obtained by $\mathcal{A}$ from the experiment by interaction. The latter means that $\mathcal{A}$ can submit queries to the so-called *oracles* $O(\cdot)$ the behavior of which is fully defined by the experiment, e.g. upon submitting a query q of particular form the experiment may provide $\mathcal{A}$ with the output of $O(q)$. For the definition of security it is important to consider probability with which the experiment $\mathsf{Expt}^{\mathrm{X}}_{\Pi,\mathcal{A}}(1^\kappa)$ outputs 1, i.e. the attack of $\mathcal{A}$ was successful. A cryptographic scheme Π is usually said to satisfy some property X if $\Pr[\mathsf{Expt}^{\mathrm{X}}_{\Pi,\mathcal{A}}(1^\kappa) = 1]$ is "almost identical" to some *target probability* p. The term "almost identical" used here depends on the security parameter 1^κ and is typically related to the notion of a negligible function.

Definition 1.1 (Negligible Function) A function $\epsilon : \mathbb{N} \mapsto \mathbb{R}$ is said to be *negligible* if for every positive polynomial $poly(\cdot)$ there exists an integer $N \in \mathbb{N}$ such that for all integers $n > N$: $\epsilon(n) < \frac{1}{poly(n)}$. $\Diamond$

To summarize, in the computational security setting scheme Π is said to satisfy some property X if the probability of a successful attack over all PPT adversaries $\mathcal{A}$ given by $|\Pr[\mathsf{Expt}^{\mathrm{X}}_{\Pi,\mathcal{A}}(1^\kappa) = 1] - p|$ is negligible (in κ). The target probability p is property-specific but usually takes values in 0 or $\frac{1}{2}$. For example, if for a successful attack $\mathcal{A}$ should solve some computational problem then p is typically set to 0, e.g. if $\mathcal{A}$ is required to forge a group signature. On the other hand, if $\mathcal{A}$ should solve some decision problem then p is typically set to $\frac{1}{2}$, in order to account for the chance of $\mathcal{A}$ to guess the result, e.g. if $\mathcal{A}$ is required to decide whether a given group signature was produced by signer i_0 or signer i_1 while the actual signer i_b was picked at random (via bit $b \in_R \{0, 1\}$).

1.5.2. Formal Security Requirements — Anonymity, Traceability, Non-Frameability

The informal security requirements for group signatures, discussed in Section 1.4, can be formalized using adversarial experiments within the setting of computational security. However, since the list of intuitive security notions is rather large, defining a separate experiment for each such notion and later proving that some group signature scheme provides all security notions is not very handy. For this reason, it is advisable to keep the security model simpler and look for relationships amongst the informal security notions allowing for formal definitions that would define several such notions using one adversarial experiment. Furthermore, formalizing the same security goal for different types of schemes could be tricky since each type has its own functionality that must be considered in the adversarial experiments. There are no guidelines on how informal security requirements should be mapped to formal definitions. The integration of several informal notions into one formal security definition is thus an intuitive process that could lead to different security models for the same type of schemes.

The first formal security model for group signatures was introduced by Bellare, Micciancio, and Warinschi [22], who considered static schemes. This model evolved over the years to capture other types of group signature schemes and/or to address new properties, both from the functionality and security point of view. Today, there exist several formal models for different types of group signature schemes, aiming at related notions of security. One of the goals of our work is to provide a cleaner view on these definitions, identify their relationships and differences, and present them in a way that would allow for a comparative security analysis amongst different (types of) group signature schemes. In the following we describe at a high level three security requirements — **anonymity**, **traceability**, and **non-frameability** — that we will later formally define for each group signature type. Table 1.1 shows how these requirements capture the intuitive security goals of unforgeability of signatures, exculpability, traceability of signers, coalition resistance, protection against framing attacks, anonymity of signers, and unlinkability of signatures from Section 1.4.

Anonymity

The formal security goal of *anonymity* is supposed to address the two informal privacy notions of anonymity and unlinkability. The adversarial experiment will require from the adversary to decide whether some group signature was produced by signer i_0 or signer i_1. We will provide

Formal Security Requirements	Intuitive Security Requirements
Anonymity	Anonymity of Signers, Unlinkability of Signatures
Traceability	Traceability of Signers, Coalition Resistance
Non-Frameability	Unforgeability of Signatures, Exculpability, Coalition Resistance, Protection against Framing Attacks

Table 1.1.: Relationship between Formal and Intuitive Security Requirements

corresponding definitions of anonymity for all types of group signatures, that is for static schemes, for dynamic schemes, possibly with (verifier-local) revocation, for schemes supporting verifiable opening procedure, and for schemes with distributed authorities. In general, we will consider two flavors of anonymity differing in the amount of information provided to the adversary in the course of its attack. Our first flavor, called *insider anonymity* will allow adversarial control over all members of the group by disclosing their secret signing keys, except for the signers i_0 and i_1. In schemes where authorities are distributed into the issuer and the opener we will also provide the adversary with the secret key of the issuer, who according to the separation of duties principle should not be able to identify signers. Our second flavor, denoted *full anonymity* will provide the adversary with secret signing keys of all group members, including i_0 and i_1. As we will see many existing group signature schemes satisfy this stronger definition of anonymity.

Traceability

The formal security goal of *traceability* will address the intuitive requirement on the traceability of signers, possibly in the presence of coalitions of malicious group members. Also this goal will be defined for all types of group signatures. The adversarial experiment modeling traceability will require from the adversary to output a group signature that passes the verification procedure but for which the opening procedure results in a failure. In schemes with verifiable opening the experiment will be extended to address security of the judgement procedure so to make sure that if the opening procedure does not fail then the output proof regarding the identity of the claimed signer will always be accepted. In general, we will consider two flavors of traceability. In the first flavor, called *insider traceability*, the adversary will be given secret signing keys of all group members (modeling their potential coalitions). This flavor guarantees traceability of signers against honest group managers. In group signature schemes with distributed authorities the adversary will furthermore be given the secret key of the opener. In this way the experiment will ensures protection of the issuer — only the issuer should decide whom to admit to the group and the secret signing keys issued in the admission process may not be misused for the generation of signatures that cannot be opened. Our second flavor, denoted *full traceability* will additionally provide the adversary with the entire secret key of the group manager. As we will see only static group signature schemes where potential group members are determined in advance and receive their secret signing keys during the key generation procedure can satisfy this stronger requirement.

Non-Frameability

The formal security goal of *non-frameability* will address intuitive requirements of unforgeability and exculpability, and further protect signers against framing attacks, possibly in the presence of coalitions of malicious group members and the manager of the group. In the corresponding adversarial experiment the adversary must come up with a group signature that passes the verification procedure but for which the opening procedure outputs an identity i^* of some group member who was not involved in the generation of that signature. Also for this notion we will consider two flavors, which will be defined for all types of group signatures. The first flavor, called *insider non-frameability*, will allow the adversary to obtain secret signing keys of all group members, except for the member with identity i^*. The adversary can still obtain valid signatures produced by i^* to model chosen message attacks, that are typical in the security definitions of digital signatures. In schemes with verifiable opening the experiment will be extended to ensure protection of the judgement procedure, i.e. to eliminate false accusations against i^* through the potential falsification of an opening proof. The second flavor, denoted *full non-frameability*, will furthermore provide the adversary with the secret key of the group manager. In particular, in schemes with distributed authorities the adversary will be given secret keys of the issuer and the opener. This models potential coalitions of malicious group members with the group manager and considers protection against framing attacks performed by the latter. In contrast to full traceability, which can only be satisfied by static schemes, the notion of full non-frameability can be fulfilled by dynamic group signature schemes as well.

1.5.3. Group Signatures and Quantum Computers

In this work we focus on group signature schemes that can be securely and practically deployed in modern digital systems. The assumed adversarial setting considers polynomial-time adversaries in the standard model of computation based on universal (polynomial-time) Turing machines. This model excludes computations on *quantum computers*, for which a more general computation model has been defined, e.g. [82, 19]. Therefore, schemes described in this work, in particular those that admit efficient realizations in practice, are based on specific hard problems from number theory such as integer factorization problem or discrete logarithm problem. These problems, however, can be efficiently solved on quantum computers, for example using the well-known algorithm of Shor [169].

A recent line of research in cryptography — *Post-Quantum Cryptography* (see e.g. [29]) — is dedicated to the construction of cryptographic schemes that would remain secure in the future, once quantum attacks become feasible in practice. Security of such schemes requires other hard problems than those that can efficiently be solved by the existing algorithms from [169]. Current techniques for building post-quantum secure schemes typically rely on either coding theory, multivariate cryptography, hash-based approaches, or lattices. Especially, the area of lattice-based cryptography, originated in the work of Ajtai [8], seems to be most promising due to the rich algebraic structure of lattices.

The so-far only group signature scheme, designed to resist quantum attacks, was proposed by Gordon, Katz, and Vaikuntanathan [103] based on lattices. This scheme, however, is more of theoretical than practical interest; in particular, it cannot compete with existing (non quantum-resistant) group signature schemes in terms of efficiency and has somewhat limited functionality (e.g. does not offer dynamic support). Nonetheless, post-quantum group signatures represent an important area of research that is likely to become of high practical relevance in a long term.

1.6. Related Approaches for Authentication with Privacy

In this section, we describe security protocols and notions that, although designed for a different purpose, share some properties with group signature schemes. Not surprisingly, they build on similar cryptographic primitives and follow resembling design strategies.

1.6.1. Anonymous Signatures

The notion of *anonymous signatures*, introduced by Yang et al. [186], aims to achieve anonymity in the traditional setting of digital signatures, where the private key of a signer is used to produce signatures that are then verified using the signer's public key. At first sight, achieving anonymity in this setting may sound contradictory to the anticipated authentication goals of signature schemes. The crucial observation here is that the verification procedure of a signature scheme requires as input the public key of the signer and the corresponding message. In the presence of system-wide known public keys there is hope to keep the signer anonymous as long as messages are not publicly disclosed. These ideas were formulated in [186], assuming that signed messages have sufficiently high entropy to prevent otherwise unavoidable attacks, by which the anonymity adversary would guess the message and try out different public keys until the right public key allowing successful verification of the given signature is found.

The original definitions of anonymity for digital signatures from [186] were simplified by Fischlin [91] and adopted to address the potential exposure of secret signing keys. He also described a general transformation that adds the anonymity property to any unforgeable digital signature scheme, while the original work in [186] showed more concrete constructions of anonymous signatures using number-theoretic constructions based on integer factorization and discrete logarithms. A slightly different concept for anonymous signatures was introduced independently in [21, 166, 189], where the assumption on high entropy of messages was traded against partial disclosure of signatures, i.e., by splitting the signature in two or more distinct components of which at least one is withheld. For these revised definitions of anonymity [21, 166, 189] provided several general transformations, achieving anonymity for arbitrary signature schemes, and [166] showed in addition a more concrete construction in the setting of bilinear maps.

The anonymity property of anonymous signatures differs, however, from the anonymity provided by group signatures in many ways. While group signatures aim to protect anonymity of the signer against verifiers, yet allowing the latter to perform the verification procedure, anonymous signatures lose their anonymity property as soon as the entire message-signature pair is revealed. This information, however, is required to perform the verification procedure. Interestingly, techniques underlying anonymous signature schemes for high entropy messages have recently been used in the constructions of group signatures, e.g. [31], to achieve a somewhat better efficiency in comparison to many earlier schemes.

1.6.2. Anonymous Credentials

An *anonymous credential system (ACS)*, originated in the work of Chaum [69], is a cryptographic scheme that provides strong authentication of users to verifiers, while protecting privacy of the former. ACS users receive credentials from organizations and treat them as their own secret. ACS credentials are typically given through certificates that organizations issue on a unique identity of the user, i.e., the identity under which this user is known to that particular organization. They resemble, in some sense, paper-based credentials like passports, driver's licenses, cinema tickets, or proofs of qualification, and suit as replacement of those credential types in the virtual world, e.g., on the Internet.

The core functionality of an ACS system is to allow users to prove possession of valid credentials without revealing their (certified) identities. Early ACS schemes, e.g. [69, 79, 135], were not very efficient or required additional (trusted) parties to assist the execution of ACS protocols [70]. Modern ACS schemes, designed with both security and efficiency in mind, were proposed only in the last decade [54, 56], improved and extended in [57, 48, 51, 52]. Some ACS offer efficient support for the revocation of issued credentials, e.g. [51, 52]. Currently, ACS are finding their ways into practice [47]: for example, Bichsel et al. [30] implemented the scheme from [54] on a smart card and a more general implementation with direct integration into an access control system is currently being developed as part of the *idemix* credential system [50] (see also http://idemix.wordpress.com/).

ACS schemes have several important security properties: for example, credentials should be *unforgeable* to prevent users from claiming possession of credentials that were never issued; ACS protocols for proving possession of valid credentials should be further *unlinkable* to guarantee user's privacy against third-party verifiers and even colluding organizations. As ACS are multi-user systems, it should not be possible for malicious users to combine their ACS credentials to create a new one. These requirements are satisfied by all modern ACS systems. The collusion-resistance property is especially important for ACS schemes that support certification of *attributes* in addition to unique user's identity, e.g. [48]. Attributes, such as age or address information, allow for a finer form of access control. As it is not possible to prevent malicious users from transferring their private credentials to other users, some ACS systems discourage this behavior using the concept of "all-or-nothing sharing", where malicious users can transfer their proving rights only by disclosing some secret information about themselves. For example, the concept of "one-show credentials" in [54] provides user privacy for at most one use of the ACS credential. ACS systems supporting revocation [51, 52] aim further to prevent users from proving possession of credentials if these were revoked by respective organizations. The challenge in this case is to allow revocation without compromising user privacy.

ACS schemes can be used for user authentication and access control, thereby preserving privacy of user credentials. An anonymous credential can thus be used to prove membership of a user in some organization (group), which parallels the goals of group signature schemes, i.e., organizations issuing ACS credentials to the users could play the role of group management authorities. The main difference between ACS schemes and group signatures is the missing ability of ACS organizations to identify users based on ACS protocol executions. In group signatures such identification is realized using the opening procedure. Thus, ACS schemes offer stronger privacy guarantees in this respect. Nonetheless, constructions of ACS schemes often share similar techniques that are used in constructions of group signatures and revocation mechanisms developed for ACS schemes can often be adopted to realize dynamic group signatures with revocation support and vice versa.

1.6.3. Affiliation-Hiding Authentication

Affiliation-hiding (AH) is a property of privacy-preserving authentication protocols such as secret handshakes [18, 184, 64, 180, 179, 111, 14, 113, 142] and, more generally, affiliation-hiding key establishment protocols [109, 110, 141, 140, 139]. These schemes hide user identities by means of group authentication, but in contrast to group signatures they also hide the actual groups (affiliations) of participants: whenever the affiliations of both protocol participants match, the authentication is successful, whereas, in all other cases, the protocol terminates without leaking any information about the groups, except for the fact that they do not match. In many AH protocols, a successful authentication session results in a shared secret key that participants can use to protect their subsequent communication. In this light, AH protocols are interactive and bear high potential for protecting privacy in online communications. AH protocols further support efficient revocation of group membership, which is usually performed by respective group authorities, who manage their own groups independently of each other.

AH protocols can be of two flavors: linkable or unlinkable. In linkable protocols, such as [18, 64, 180, 110, 141, 140, 139], users communicate using pseudonyms for which they hold corresponding group membership credentials. These pseudonyms can be revoked by the group authorities, i.e., put into revocation lists which are distributed in an authenticated manner. Linkable protocols are usually more efficient than unlinkable protocols. In unlinkable protocols [14, 111, 179, 113], in order to prevent any correlation amongst sessions of the same user, complex group management techniques are deployed or some security compromise is taken into account: For example, [14] does not support revocation, [111] needs synchronization of revocation epochs amongst members, [179] combines group signatures with broadcast encryption techniques, while [113] combines group signatures with verifier-local revocation and private conditional oblivious transfer schemes.

Several AH protocols, e.g. [141, 142], aim to protect privacy of group members against respective group authorities, in particular to ensure that group members remain anonymous during the protocol execution. Some recent protocols, e.g. [41, 112, 138, 140, 139], offer efficient solutions for handling multiple group membership credentials in a single protocol session, which considerably improves the practicality of AH schemes.

Although techniques from group signatures have also been used in the construction of AH protocols, the two concepts have one significant difference: in group signatures the verification process, i.e., the check whether the signer belongs to a claimed group or not, is a public operation that can be performed by any verifier in possession of the group public key on his own, whereas in AH protocols only group members are entitled to verify each other's group membership via interaction.

1.6.4. Blind Signatures

With *blind signatures*, introduced by Chaum [68], signatures can be obtained on messages that remain undisclosed to the signer. The resulting signature is publicly verifiable using the signer's public key and the message. That is, blind signature can be seen as an analogue to a closed carbon copy envelope that contains some messages, which is signed without opening the envelope. The signature generation process requires interaction between the signer and the user, who is in possession of the message that should be signed.

The typical properties of blind signatures are unforgeability and blindness (sometimes called unlinkability). In particular, the latter is a privacy property by which signers should not be able to link any message-signature pair to the interaction in which that signature was issued. This property also means that signers do not obtain any information about messages during the signature generation procedure. Applications of blind signatures include e-cash and e-voting systems, and they have also been used in the construction of anonymous credential systems and some affiliation-hiding authentication protocols, where blind signatures are mostly used for the generation of credentials that users obtain from respective management authorities.

Since the seminal works of Chaum [68], efficient constructions were proposed in different cryptographic settings, e.g. using discrete logarithms [161, 3], based on integer factorization [23, 53, 122, 123], in the setting of bilinear maps [33, 155, 4], using lattices [164], and from general hardness assumptions [114, 90, 107, 5, 98]. These schemes have varying efficiency, for example schemes in [68, 33, 90, 122, 4, 98] achieve optimal communication complexity of two rounds.

Some ideas underlying blind signatures have also been extended to group signatures. In particular, Lysyanskaya and Ramzan [134] introduced the notion of *group blind signatures* aiming to adopt the blindness property to group signature schemes. As an application they proposed an e-cash system where multiple banks can distribute anonymous and untraceable digital coins.

1.6.5. Direct Anonymous Attestation

Direct anonymous attestation (DAA) [175] is a cryptographic protocol designed primarily to enhance user privacy within the remote attestation process of computing platforms, which has been adopted by the Trusted Computing Group (TCG, http://www.trustedcomputinggroup.org/) in its latest specification. The remote attestation process follows roughly the following concept: a tamper-resistant hardware chip, often referred to as the *Trusted Platform Module (TPM)*, is embedded into the computing platform (e.g. a laptop or smartphone) during the manufacturing process and is initialized by an issuing authority, which installs its private key into all issued TPMs. TPMs can certify local configurations of computing platforms and are important for the remote attestation process, in which some verifier wishes to remotely check whether the computing platform is equipped with a TPM and has been configured to match some pre-defined policy. This process is realized through a challenge-response protocol, where the verifier's challenge is authenticated by the TPM together with the local configuration of the platform. The tamper-resistance of the TPM protects against impersonation attacks. In addition, the process requires protection of privacy, in particular different attestation sessions should remain *unlinkable*.

The DAA solution, originated by Brickell, Camenisch and Chen [44] specifies how issuers, hosts, TPMs, and verifiers interact to cope with the given problem in a privacy-preserving way, i.e., verifiers cannot recognize TPMs across different attestation sessions. Moreover, TPMs can be efficiently revoked by the issuing authorities. The DAA approach doesn't require any trusted third parties (unlike, for example, the Privacy CA approach used in earlier TCG specifications). The original work on DAA has further been extended, e.g. in [173, 129, 43, 72], however TCG specification of DAA is widely based on the initial version from [44].

DAA uses techniques that are similar to those utilized in group signatures and other privacy-preserving authentication schemes such as anonymous credential systems. In general, the TPM is in possession of a private credential, which it receives from the issuer, and uses this credential in the remote attestation process without leaking information that would allow the verifier to link two distinct sessions to the same TPM. On the other hand, the user of the computing platform has freedom to decide whether unlinkability of its attestation sessions should be dropped.

In fact, attestation process in DAA has similar goals to group signatures but one important difference: in DAA unlinkability of attestation sessions also guarantees anonymity of TPMs and none of the parties in the system is able to identify which TPM participated in the procedure. That is, the opening procedure, which is inherent to group signatures, is not available in the DAA setting.

1.6.6. Ring Signatures

With *ring signatures*, introduced by Rivest, Shamir, and Tauman [162], users in possession of public keys can create lists containing public keys of other users and produce signatures that would not leak any information about the original signer, except that the signer's public key is part of the composed list. A ring signature scheme should be *unforgeable*, meaning that generation of ring signatures without knowledge of at least one secret key corresponding to one of the public keys in the list must remain infeasible. Ring signatures offer privacy since the actual signer can remain *anonymous* and even *unlinkable*, that is no verifier should be able to determine which public key in the list belongs to the signer of the message or whether two ring signatures were produced by the same signer. For example, ring signatures can be used to leak insider information to the press in a manner that authenticates the source as a knowledgeable insider but protects its identity.

The first ring signature scheme in [162] was constructed using special techniques that employed trapdoor permutations. Several ring signature schemes were constructed later, e.g. [6, 108, 37, 97, 183, 85, 76, 66], based on different number-theoretic assumptions. The original security definitions from [162] were strengthened later by Bender, Katz, and Morselli [27] to take possible leakage of private keys and the possibility of insider attacks into account, that are sometimes considered in group signature schemes.

Some ring signature schemes offer additional properties such as linkability [132, 177, 176] and verifiability [125, 188]: in a linkable scheme several signatures from the same signer can be linked; in case of verifiability the signer himself – and nobody else – can issue a proof of having produced the signature. Additionally, several schemes, e.g. [42, 183, 78], distributed the signing abilities amongst multiple signers by requiring their collaboration during the signature generation process.

The inherent limitation of ring signatures is that the length of the signature is not constant, which weakens their applicability in practice. The most efficient scheme in that respect was proposed by Chandran, Groth, and Sahai [66]. In this scheme, the length of ring signatures is sub-linear in the number of public keys in the list, whereas all previous schemes experienced linear increase of the signature length.

The properties of ring signatures are very similar to group signatures: both schemes rely on group-based authentication, i.e., by viewing the list of public keys in ring signatures as some sort of a group of potential signers. However, in ring signatures signers can decide freely which public keys to include into their list. Moreover, the distinguishing property of ring signatures is that the anonymity is guaranteed unconditionally, i.e., no party can identify the signer. This differs from group signatures whose signers can be identified through the corresponding opening procedure.

1.6.7. Traceable Signatures

The notion of *traceable signatures*, introduced by Kiayias, Tsiounis, and Yung [117], can be seen as a special type of group signatures with a slightly different traceability concept: the group manager is in possession of a special *tracing trapdoor* for each member of the group. By revealing the trapdoor for some user i, all signatures issued by i in the past can be identified independently by other parties, called tracing agents. For this purpose, tracing agents link all signatures of i using the given trapdoor. On the other hand, tracing agents do not learn the actual identity of i. That is, traceable signatures offer anonymity (through the group-based authentication approach) and unlinkability of signatures, while the latter can be revoked through the publication of the appropriate trapdoor. The group manager is the only authority which can identify the actual signer i through the corresponding opening procedure, as in group signatures. Furthermore, a signer of some traceable signature can prove that he indeed was the signer. Such *signature claiming* procedure is inherent to traceable signatures and it allows other parties to check whether a signature stems from a certain user without asking the group manager to execute the opening procedure for the given signature. For example, the scheme from [117] allows signers to provably claim own signatures without keeping the state for each generated signature and without involvement of the group manager.

Traceable signatures can be used to implement an anonymous auction protocol with open bids. That is, bidders submit signed bids and the signature of the highest bid is opened to determine the winner of the auction. Bidders can use the claiming functionality to prove the ownership of submitted bids and the tracing property can help to identify bids submitted by misbehaving bidders.

The first traceable signature scheme was proposed in [117] and improved later in [28]. Several more efficient schemes [100, 74] have been proposed thereafter. In particular the scheme from [75] improved the efficiency of the tracing process by tracing agents. Variations of traceable signatures also include schemes, e.g. [182, 181], where the tracing process cannot be performed by a single tracing agent but requires collaboration of several agents and by this decreases the risk of an abuse of the tracing procedure.

A traceable signature scheme can be seen as a group signature scheme with the additional property that signatures of one user can be identified without the need of opening all signatures in the system. Indeed, as mentioned in [117], the group manager in a group signature scheme can open all signatures and by this identify all signatures belonging to the same signer. This, however, would breach the privacy of all other group members and is, therefore, undesirable. Nonetheless, the technical realization of traceable signatures uses techniques that are also used in constructing group signatures. In particular, the tracing process has some similarities to the verifier-local revocation approach offered by some group signature schemes. The signature claiming procedure is, however, unique to traceable signatures. In fact, some variations of group signature schemes explicitly aim to prevent the ability of the signer to claim own signatures, for example through the additional security requirement called “leak-freedom” in the scheme by Ding, Tsudik, and Xu [83, 84].

2. Group Signatures: Definitions and Security Models

The classification of group signature schemes assumes four main types: static schemes, dynamic schemes (possibly with revocation), schemes with verifiable opening procedure (possibly with explicit consideration of user PKI), and schemes where the role of the group manager is distributed between the issuer and the opener. For each type we will formally define the syntax of corresponding algorithms and their correctness requirements together with the three formal security notions of anonymity, traceability, and non-frameability.

2.1. Static Group Signature Schemes

In this section we introduce formal definitions for static group signature schemes, where the number of group members $n \in \mathbb{N}$ is fixed in advance. Our definitions model the *basic* functionality of such schemes, namely: generation of secret and public keys, generation of group signatures by members of the group, public verification of group signatures, and their opening through the corresponding manager of the group. Our description of the algorithms follows the model for static group signatures proposed by Bellare, Micciancio, and Warinschi [22] with some slight modifications.

2.1.1. Algorithms of Static Schemes and Their Correctness Property

Definition 2.1 (Static Group Signature Scheme) A *static* group signature scheme $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ consists of four polynomial-time algorithms:

Key generation. The randomized *group key generation* algorithm GKg takes as input the security parameter 1^κ, $\kappa \in \mathbb{N}$, and the total number of group members $n \in \mathbb{N}$, and returns a tuple $(gpk, gmsk, \boldsymbol{gsk})$, where gpk is the *group public key*, $gmsk$ is the *group manager's secret key*, and $\boldsymbol{gsk}$ is an n-element vector of keys with $\boldsymbol{gsk}[i]$ being the *secret signing key of member* i, $1 \leq i \leq n$.

Signature generation. The randomized *group signing* algorithm GSign takes as input a secret signing key $\boldsymbol{gsk}[i]$ and a message m, and returns a *group signature* σ.

Signature verification. The deterministic *group signature verification* algorithm GVrfy takes as input the group public key gpk, a message m, and a candidate group signature σ for m, and returns either 1 (to indicate that the signature is valid) or 0 (to indicate a failure).

Opening procedure. The deterministic *opening* algorithm Open takes as input the group manager's secret key $gmsk$, a message m, and a signature σ, and returns either a signer's identity $i \in [1, n]$ or 0 (to indicate a failure). ◊

Note that the opening procedure is defined independently of the verification procedure. This means that $\mathsf{Open}(gmsk, m, \sigma)$ can be potentially executed on a message-signature pair (m, σ) for which the verification procedure would fail. In practice, however, it is likely that the Open algorithm will check whether $\mathsf{GVrfy}(gpk, m, \sigma) = 1$ before attempting to identify the signer i. Nevertheless, it is left up to the scheme Γ to decide whether the opening procedure should perform this check.

A static group signature scheme Γ should satisfy the following correctness property. Its first condition guarantees that group signatures produced by group members can be verified successfully, while its second condition ensures correct signer identification by the manager of the group.

Definition 2.2 (Correctness : Static) A group signature scheme $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ is *correct* if for all $\kappa, n \in \mathbb{N}$, all keys $(gpk, gmsk, \boldsymbol{gsk}) \leftarrow \mathsf{GKg}(1^\kappa, n)$, all identities $i \in [1, n]$, and all messages $m \in \{0, 1\}^*$:

$$\mathsf{GVrfy}(gpk, m, \mathsf{GSign}(\boldsymbol{gsk}[i], m)) = 1 \quad \text{and} \quad \mathsf{Open}(gmsk, m, \mathsf{GSign}(\boldsymbol{gsk}[i], m)) = i.$$

◊

Remark 2.1.1 In general, static group signature schemes fix the entire number of group members and generate their secret signing keys in advance during the key generation procedure. Nevertheless, static schemes may also offer revocation support, allowing the group manager to only remove group members (but not to add new members). In this case algorithms of static schemes can be updated and extended with additional algorithms for handling the revocation procedure. We do not describe these optional algorithms at this point. Instead, we refer the reader to Section 2.2.2, where we provide a more detailed discussion on revocation handling and corresponding algorithms in the context of dynamic group signature schemes. Nevertheless, we notice that those algorithms and their parameters can also be applied to static schemes.

2.1.2. Adversary Model and Oracles for Static Schemes

Our formal definitions of security for static group signature schemes $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ will be provided through probabilistic experiments $\mathsf{Expt}_{\Gamma,\mathcal{A}}(1^\kappa)$ as mentioned in Section 1.5. In general, $\mathcal{A}$ will be given access to (a subset of) the following three oracles:

Corruption oracle. The *corruption oracle* $\mathsf{Corrupt}(\cdot)$ takes as input an identity $i \in [1, n]$ and returns the secret signing key $\boldsymbol{gsk}[i]$.

The corruption oracle $\mathsf{Corrupt}(\cdot)$ models attacks in which the adversary $\mathcal{A}$ can form coalitions of group members aiming to break some security property of the scheme.

Signing oracle. The *signing oracle* $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$ takes as input an identity-message pair (i, m) with $i \in [1, n]$, and returns the output of the group signing algorithm $\mathsf{GSign}(\boldsymbol{gsk}[i], m)$.

The signing oracle $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$ models *chosen-message attacks* by which $\mathcal{A}$ may lure some group member into signing a message, hoping that this signature will reveal information that is useful for the attack.

Opening oracle. The *opening oracle* $\mathsf{Open}(gmsk, \cdot, \cdot)$ takes as input a message-signature pair (m, σ), and returns the output of the opening algorithm $\mathsf{Open}(gmsk, m, \sigma)$.

The opening oracle $\mathsf{Open}(gmsk, \cdot, \cdot)$ models *chosen-signature attacks* by providing $\mathcal{A}$ access to the opening procedure of the scheme for arbitrary message-signature pairs.

2.1.3. Anonymity Definitions for Static Schemes

The anonymity property of group signature schemes aims at protection of the signers' identities. This requirement can be formalized through an anonymity experiment, in which a PPT adversary algorithm $\mathcal{A}$ on input some information about the deployed group signature scheme Γ is supposed to decide, whether a "challenge" group signature σ^* on a message m^* has been produced by signer i_0 or by signer i_1. Any successful attempt of $\mathcal{A}$ to attribute σ^* to its signer i_b, $b \in \{0, 1\}$ with a probability being non-negligibly greater than $\frac{1}{2}$ (for a simple guess of bit b) would immediately break the anonymity property of the scheme.

The strength of some particular formal definition of anonymity can be influenced by defining the amount of information given to the adversary $\mathcal{A}$. In this section we define two anonymity requirements that differ in their strengths. We start with insider anonymity, where $\mathcal{A}$ operates on public information that can be obtained from the environment in which the scheme is deployed, and, additionally, may learn secret signing keys of some group members, except for the keys owned by the signers i_0 and i_1 from the anonymity experiment. This requirement preserves anonymity of group members with respect to other members of the group. Our second definition considers full anonymity, where the adversary $\mathcal{A}$ learns secret signing keys of all group members. This is the strongest notion of anonymity for static group signature schemes as defined by Bellare, Micciancio, and Warinschi [22].

Insider Anonymity

The notion of insider anonymity considers adversaries being in possession of secret signing keys of other members of the group. The corresponding anonymity experiment provides the adversary $\mathcal{A}$ with all public information about the deployed group signature scheme Γ and the three oracles $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$, and $\mathsf{Open}(gmsk, \cdot, \cdot)$ whose behavior has been specified in Section 2.1.2. Using the opening oracle $\mathcal{A}$ could trivially attribute the challenge group signature σ^* on a message m^* to its signer i_b. For this reason the experiment prohibits $\mathcal{A}$ from opening (m^*, σ^*).

Definition 2.3 (Insider Anonymity : Static) A group signature scheme $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ provides *insider anonymity* if, for all probabilistic, polynomial-time adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{I-AN}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = \left| \Pr\left[\mathsf{Expt}^{\text{I-AN}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = 1\right] - \frac{1}{2} \right| .$$

The associated I-AN-*experiment* $\mathsf{Expt}^{\text{I-AN}}_{\Gamma,\mathcal{A}}(1^\kappa, n)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa, n)$ is executed to produce $(gpk, gmsk, \boldsymbol{gsk})$.

Attack Stage I. Adversary $\mathcal{A}_1$ receives gpk.

1. $\mathcal{A}_1$ can submit queries to the oracles $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$, and $\mathsf{Open}(gmsk, \cdot, \cdot)$.
2. $\mathcal{A}_1$ stops and eventually outputs a tuple (st, i_0, i_1, m^*) containing some state information st, two *uncorrupted* identities $i_0, i_1 \in [1, n]$, and a challenge message m^*.

Challenge Stage. A bit $b \in \{0, 1\}$ is chosen at random and the signature generation algorithm $\mathsf{GSign}(\boldsymbol{gsk}[i_b], m^*)$ is executed to produce the challenge group signature σ^*.

Attack Stage II. Adversary $\mathcal{A}_2$ receives (st, σ^*).

1. $\mathcal{A}_2$ can submit queries to the oracles $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$, and $\mathsf{Open}(gmsk, \cdot, \cdot)$ as before, subject to the following restrictions:
   a) The corruption oracle $\mathsf{Corrupt}(\cdot)$ ignores queries of the form i_0 and i_1.
   b) The opening oracle $\mathsf{Open}(gmsk, \cdot, \cdot)$ ignores queries of the form (m^*, σ^*).
2. $\mathcal{A}_2$ stops and eventually outputs a bit b^*.

Output: If $b^* = b$ then the experiment outputs 1, otherwise it outputs 0. $\diamondsuit$

Remark 2.1.2 Our definition of insider anonymity resembles the notion of **selfless anonymity** introduced by Boneh and Shacham [38] in the context of group signatures with the verifier-local-revocation (VLR) property. The term "selfless" in their notion refers to the restriction that neither identity i_0 nor i_1 can be submitted to the corruption oracle, which is also true for our definition. Our Definition 2.3 thus adopts their ideas to the case of static group signature schemes that do not have the VLR property.

Full Anonymity

The notion of full anonymity is the strongest anonymity notion for static group signatures. The anonymity adversary $\mathcal{A}$ is provided with secret signing keys of *all* group members (including i_0 and i_1 from the challenge stage of the anonymity experiment). The corresponding anonymity experiment can thus be simplified in comparison to the experiment for insider anonymity: Since $\mathcal{A}$ learns all signing keys in $\boldsymbol{gsk}$ both the corruption oracle $\mathsf{Corrupt}(\cdot)$ and the signing oracle $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$ become obsolete and can be safely omitted. We proceed with the formalization of full anonymity in Definition 2.4, which is essentially the one proposed by Bellare, Micciancio, and Warinschi [22].

Definition 2.4 (Full Anonymity : Static) A group signature scheme $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ provides *full anonymity* if, for all probabilistic, polynomial-time adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{F-AN}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = \Pr\left[\mathsf{Expt}^{\text{F-AN}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = 1\right] - \frac{1}{2}\ .$$

The associated F-AN-*experiment* $\mathsf{Expt}^{\text{F-AN}}_{\Gamma,\mathcal{A}}(1^\kappa, n)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa, n)$ is executed to produce $(gpk, gmsk, \boldsymbol{gsk})$.

Attack Stage I. Adversary $\mathcal{A}_1$ receives $(gpk, \boldsymbol{gsk})$.

1. $\mathcal{A}_1$ can submit queries to the oracle $\mathsf{Open}(gmsk, \cdot, \cdot)$.
2. $\mathcal{A}_1$ stops and eventually outputs a tuple (st, i_0, i_1, m^*) containing some state information st, two identities $i_0, i_1 \in [1, n]$, and a challenge message m^*.

Challenge Stage. A bit $b \in \{0, 1\}$ is chosen at random and the signature generation algorithm $\mathsf{GSign}(\boldsymbol{gsk}[i_b], m^*)$ is executed to produce the challenge group signature σ^*.

Attack Stage II. Adversary $\mathcal{A}_2$ receives (st, σ^*).

1. $\mathcal{A}_2$ can submit queries to the oracle $\mathsf{Open}(gmsk, \cdot, \cdot)$, except that queries of the form (m^*, σ^*) are ignored.
2. $\mathcal{A}_2$ stops and eventually outputs a bit b^*.

Output: If $b^* = b$ then the experiment outputs 1, otherwise it outputs 0. ◊

Remark 2.1.3 Our definition of full anonymity provides the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ not only with secret signing keys $\boldsymbol{gsk}$ of all group members but also with an unlimited access to the oracle $\mathsf{Open}(gmsk, \cdot, \cdot)$ with the only exception that $\mathcal{A}_2$ is not allowed to open the challenge message-signature pair (m^*, σ^*). Although several group signature schemes can satisfy this strong notion, there exist, as we shall see, several schemes that can only handle a somewhat weaker case, where adversary access to the oracle $\mathsf{Open}(gmsk, \cdot, \cdot)$ is blocked completely. This weaker flavor of full anonymity was introduced by Boneh, Boyen, and Shacham [36], and is called **CPA-full anonymity**, where the term CPA refers to chosen plaintext attacks in analogy to the IND-CPA security of encryption schemes, which prohibits adversary access to the decryption oracle. In this terminology our Definition 2.4 can also be called **CCA-full anonymity** in analogy to the chosen ciphertext attacks used to define IND-CCA security of encryption schemes that allows adversary access to the decryption oracle with some restrictions. (At this point we refer the reader to Section 3.5 for definitions of IND-CPA and IND-CCA security in case of public key encryption schemes.)

2.1.4. Traceability Definitions for Static Schemes

An inherent property of group signatures is the ability of the group manager to open the signature and identify its signer. The notion of traceability considers attacks in which a group signature is generated for which the opening procedure fails. This requirements can be modeled through a traceability experiment, in which a PPT adversary $\mathcal{A}$ succeeds if it can output a message-signature pair (m^*, σ^*) for which the verification procedure results in $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$ but the opening procedure results in $\mathsf{Open}(gmsk, m^*, \sigma^*) = 0$.

Again, the strength of some particular formal definition of traceability can be influenced by defining the amount of information given to the adversary $\mathcal{A}$. Our definitions comprise two traceability notions: First, we define insider traceability by considering malicious group members who form coalitions aiming to generate untraceable group signatures. Second, we strengthen this notion towards full traceability by considering coalitions that include the manager of the group.

Insider Traceability

The notion of insider traceability in static group signature schemes considers traceability attacks by members of the group, while assuming honest group managers. This requirements is modeled through a traceability experiment, in which $\mathcal{A}$ is given all secret signing keys in $\boldsymbol{gsk}$ and can thus generate group signatures on behalf of any group member. In this experiment all group members collude and work against the group manager, which remains honest in that its secret key $gmsk$ remains unknown to the adversary. Insider traceability allows $\mathcal{A}$ to submit message-signature pairs of its choice that will then be opened by the group manager.

Definition 2.5 (Insider Traceability : Static) A group signature scheme $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ provides *insider traceability* if for all probabilistic, polynomial-time adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{I-TR}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = \Pr\left[\mathsf{Expt}^{\text{I-TR}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = 1\right].$$

The associated I-TR-*experiment* $\mathsf{Expt}^{\text{I-TR}}_{\Gamma,\mathcal{A}}(1^\kappa, n)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa, n)$ is executed to produce $(gpk, gmsk, \boldsymbol{gsk})$.

Attack Stage. Adversary $\mathcal{A}$ receives $(gpk, \boldsymbol{gsk})$.

1. $\mathcal{A}$ can submit queries to the oracle $\mathsf{Open}(gmsk, \cdot, \cdot)$.
2. $\mathcal{A}$ stops and eventually outputs a message-signature pair (m^*, σ^*).

Output. If $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$ and $\mathsf{Open}(gmsk, m^*, \sigma^*) = 0$ then the experiment outputs 1, otherwise it outputs 0. $\Diamond$

We note that Definition 2.5 is sufficient for most applications of static group signatures since the group manager's secret key $gmsk$ is likely to be well protected from potential leakage. Nevertheless, we show in the following, how to strengthen this traceability notion against such attacks.

Full Traceability

The notion of full traceability extends the notion of insider traceability by allowing the adversary $\mathcal{A}$ to additionally learn the secret key of the group manager *gmsk*. This introduces additional protection for the traceability of group signatures in case where the group manager's secret key is leaked. It is important to mention that giving *gmsk* to $\mathcal{A}$ does not mean that the group manager is considered to be dishonest. Indeed any group manager can misbehave during the execution of the opening procedure. For example, the group manager can refuse to open some signature or simply output 0 ignoring the actual signature that it is supposed to open. For these reasons, our definition of full traceability models honest execution of the opening procedure with respect to the adversary generated group signatures, even if this adversary knows all secret information used by the scheme.

Definition 2.6 (Full Traceability : Static) A group signature scheme $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ provides *full traceability* if for all probabilistic, polynomial-time adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{F-TR}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = \Pr\left[\mathsf{Expt}^{\text{F-TR}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = 1\right].$$

The associated F-TR-*experiment* $\mathsf{Expt}^{\text{F-TR}}_{\Gamma,\mathcal{A}}(1^\kappa, n)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa, n)$ is executed to produce $(gpk, gmsk, \boldsymbol{gsk})$.

Attack Stage. Adversary $\mathcal{A}$ receives $(gpk, \boldsymbol{gsk}, gmsk)$. $\mathcal{A}$ stops and eventually outputs a message-signature pair (m^*, σ^*).

Output. If $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$ and $\mathsf{Open}(gmsk, m^*, \sigma^*) = 0$ then the experiment outputs 1, otherwise it outputs 0. $\diamondsuit$

Remark 2.1.4 Our definition of full traceability differs from the one proposed by Bellare, Micciancio, and Warinschi [22] in that it does not consider attacks by which the opening algorithm on input the adversary generated message-signature pair (m^*, σ^*) outputs some identity $i^* \in [1, n]$ of a member who did not sign m^*. We model this requirement explicitly in our definitions of non-frameability in Section 2.1.5. This separation allows us to maintain consistency when considering the case of dynamic group signatures, where such separation becomes essential.

2.1.5. Non-Frameability Definitions for Static Schemes

The notion of non-frameability focuses on attacks in which a coalition of group members aims to generate a group signature which then opens to some other member of the group. These attacks can be mounted to accuse some particular group member of having signed some message, which this member never signed. This requirement can be modeled through a non-frameability experiment, in which a PPT adversary$\mathcal{A}$ succeeds if it can output a message-signature pair (m^*, σ^*), for which the opening algorithm $\mathsf{Open}(gmsk, m^*, \sigma^*)$ returns identity $i^* \in [1, n]$ that belongs to some signer who never signed m^*.

This basic experiment can be strengthened by providing the adversary $\mathcal{A}$ with more information and capabilities. We will discuss two definitions of non-frameability for static group signature schemes: Our first definition of insider non-frameability targets at coalitions of malicious group members aiming at the generation of group signatures that will open to group members that are not part of that coalition. Our second definition of full non-frameability considers stronger coalitions, where malicious group members cooperate with the manager of the group.

Insider Non-Frameability

The notion of insider non-frameability in static group signature schemes considers framing attacks by a coalition of group members, while assuming that the group manager remains honest. The ability of the adversary to build coalitions of malicious group members is modeled through the corruption oracle $\mathsf{Corrupt}(\cdot)$. Furthermore, $\mathcal{A}$ is given access to the signing oracle $\mathsf{GSign}(\boldsymbol{gsk}[\cdot],\cdot)$ and to the opening oracle $\mathsf{Open}(gmsk,\cdot,\cdot)$. We refer to Section 2.1.2 for the specification of these oracles. The experiment captures meaningful framing attacks as it prohibits $\mathcal{A}$ from using the secret signing key $\boldsymbol{gsk}[i^*]$ directly to compute the message-signature pair (m^*,σ^*).

Definition 2.7 (Insider Non-Frameability : Static) A group signature scheme $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ provides *insider non-frameability* if for all probabilistic, polynomial-time adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{I-NF}}_{\Gamma,\mathcal{A}}(1^\kappa,n) = \Pr\left[\mathsf{Expt}^{\text{I-NF}}_{\Gamma,\mathcal{A}}(1^\kappa,n) = 1\right].$$

The associated I-NF-*experiment* $\mathsf{Expt}^{\text{I-NF}}_{\Gamma,\mathcal{A}}(1^\kappa,n)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa,n)$ is executed to produce $(gpk, gmsk, \boldsymbol{gsk})$.

Attack Stage. Adversary $\mathcal{A}$ receives gpk.

1. $\mathcal{A}$ can submit queries to the oracles $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot],\cdot)$, and $\mathsf{Open}(gmsk,\cdot,\cdot)$.
2. $\mathcal{A}$ stops and eventually outputs a message-signature pair (m^*,σ^*).

Output. If all of the following holds then the experiment outputs 1:

1. $\mathsf{GVrfy}(gpk,m^*,\sigma^*) = 1$ and $\mathsf{Open}(gmsk,m^*,\sigma^*) = i^*$ with $i^* \in [1,n]$
2. $\mathcal{A}$ did not submit i^* to $\mathsf{Corrupt}(\cdot)$
3. $\mathcal{A}$ did not submit (i^*,m^*) to $\mathsf{GSign}(\boldsymbol{gsk}[\cdot],\cdot)$.

Otherwise it outputs 0. ◊

Full Non-Frameability

The notion of full non-frameability in static group signature schemes considers framing attacks mounted by a coalition of group members with the group manager. The corresponding non-frameability experiment thus provides $\mathcal{A}$ with the secret key of the group manager $gmsk$.

Since $\mathcal{A}$ can now execute the opening procedure on its own, the corresponding opening oracle $\mathsf{Open}(gmsk, \cdot, \cdot)$ becomes obsolete and can be omitted from the experiment.

Definition 2.8 (Full Non-Frameability : Static) A group signature scheme $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ provides *full non-frameability* if for all probabilistic, polynomial-time adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = \Pr\left[\mathsf{Expt}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = 1\right].$$

The associated F-NF-*experiment* $\mathsf{Expt}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa, n)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa, n)$ is executed to produce $(gpk, gmsk, \boldsymbol{gsk})$.

Attack Stage. Adversary $\mathcal{A}$ receives $(gpk, gmsk)$.

1. $\mathcal{A}$ can submit queries to the oracles $\mathsf{Corrupt}(\cdot)$ and $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$.
2. $\mathcal{A}$ stops and eventually outputs a message-signature pair (m^*, σ^*).

Output. If all of the following holds then the experiment outputs 1:

1. $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$ and $\mathsf{Open}(gmsk, m^*, \sigma^*) = i^*$ with $i^* \in [1, n]$
2. $\mathcal{A}$ did not submit i^* to $\mathsf{Corrupt}(\cdot)$
3. $\mathcal{A}$ did not submit (i^*, m^*) to $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$.

Otherwise it outputs 0. $\Diamond$

2.2. Dynamic Group Signature Schemes

In this section we extend our formal definitions towards dynamic group signature schemes where new group members can be dynamically admitted to the group. We again focus on the basic functionality of such schemes, namely: generation of secret and public keys, admission of new members to the group, generation of group signatures by members of the group, public verification of group signatures, and their opening through the corresponding manager of the group. In addition to defining the algorithms and their correctness property we revisit our security definitions for different flavors of anonymity, traceability, and non-frameability in the context of dynamic schemes.

2.2.1. Algorithms of Dynamic Schemes and Their Correctness Property

The main difference in the definition of dynamic group signature schemes is the additional protocol Join, which is executed between the group manager and some user wishing to become a group member i. Our definition also uses integer $n \in \mathbb{N}$ to indicate the upper-bound on the total number of users that can be admitted to the group. This does not mean that the scheme is static since n can be thought of being sufficiently large and seen as a place-holder for identities of future group members. In particular, we will use n to check whether some identity i is a valid identity by verifying that $i \in [1, n]$. We define the protocol Join as a pair of interactive algorithms $(\mathsf{JoinM}, \mathsf{JoinU})$ assuming that the group manager executes its part denoted JoinM and the prospective group member its part denoted JoinU. In order to emphasize that the group manager's secret key $gmsk$ does not evolve over the time and at the same time allow for the dynamic update of information that can be used by the group manager for the purpose of opening group signatures, we introduce a *registration list* $\boldsymbol{reg}$ which is supposed to be kept secret by the group manager. This list is initially empty and is updated on every successful execution of the JoinM part of the protocol.

Definition 2.9 (Dynamic Group Signature Scheme) A *dynamic* group signature scheme $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ consists of five polynomial-time algorithms/protocols:

Key generation. The randomized *group key generation* algorithm GKg takes as input the security parameter 1^κ, $\kappa \in \mathbb{N}$ and returns a tuple $(gpk, gmsk, \boldsymbol{reg})$, where gpk is the *group public key*, $gmsk$ is the *group manager's secret key*, and $\boldsymbol{reg}$ is the *registration list*, which is initially empty.

Join protocol. The randomized Join protocol is a two-party protocol composed of two interactive algorithms $(\mathsf{JoinM}, \mathsf{JoinU})$. JoinM takes as input the group manager's secret key $gmsk$ and identity $i \in [1, n]$, and at the end of the interaction adds a *registration entry* $\boldsymbol{reg}[i]$ to the list $\boldsymbol{reg}$. JoinU takes as input the group public key gpk and identity $i \in [1, n]$, and at the end of the interaction outputs the *secret signing key* $\boldsymbol{gsk}[i]$. If either JoinM or JoinU fails then the respective output is set to $\perp$.

Signature generation. The randomized *group signing* algorithm GSign takes as input a secret signing key $\boldsymbol{gsk}[i]$ and a message m, and returns a *group signature* σ.

Signature verification. The deterministic *group signature verification* algorithm GVrfy takes as input the group public key gpk, a message m, and a candidate signature σ for m, and returns either 1 (to indicate that the signature is valid) or 0 (to indicate a failure).

Opening procedure. The deterministic *opening* algorithm Open takes as input the group manager's secret key $gmsk$, a message m, a signature σ, and the registration list $\boldsymbol{reg}$, and returns either a signer's identity $i \in [1, n]$ or 0 (to indicate a failure). ◊

We again define the opening procedure independently of the verification procedure, thus leaving it up to the dynamic scheme Γ to decide, whether $\mathsf{GVrfy}(gpk, m, \sigma) = 1$ should be checked prior to opening (m, σ).

A dynamic group signature scheme Γ should satisfy the following correctness property. Its first condition guarantees that group signatures produced by group members can be verified successfully, while its second condition ensures correct signer identification by the manager of the group. Note that since Γ is dynamic the second condition should hold only for members i that have been admitted to the group through the execution of the Join protocol. This is modeled by requiring that correctness conditions hold for all outputs $(\boldsymbol{reg}[i], \boldsymbol{gsk}[i]) \leftarrow$

$(\mathsf{JoinM}(gmsk, i), \mathsf{JoinU}(gpk, i))$ for any $i \in [1, n]$. In particular, if some identity $i \in [1, n]$ has not been admitted to the group then the opening algorithm is supposed to return 0.

Definition 2.10 (Correctness : Dynamic) A group signature scheme $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ is *correct* if for all $\kappa, n \in \mathbb{N}$, all outputs $(gpk, gmsk, \boldsymbol{reg}) \leftarrow \mathsf{GKg}(1^\kappa)$, all outputs $(\boldsymbol{reg}[i], \boldsymbol{gsk}[i]) \leftarrow (\mathsf{JoinM}(gmsk, i), \mathsf{JoinU}(gpk, i))$ for any $i \in [1, n]$, and all messages $m \in \{0, 1\}^*$:

$$\mathsf{GVrfy}(gpk, m, \mathsf{GSign}(\boldsymbol{gsk}[i], m)) = 1 \quad \text{and} \quad \mathsf{Open}(gmsk, m, \mathsf{GSign}(\boldsymbol{gsk}[i], m), \boldsymbol{reg}) = i.$$

$\Diamond$

2.2.2. Optional Algorithms for Membership Revocation

The algorithms of a dynamic group signature scheme introduced in Definition 2.9 consider groups where new members can be added to the group. In fully dynamic schemes, members can also leave the group or be excluded from the group by the group manager, depending on the application. Obviously, the group signature scheme has to be updated with the revocation procedure. Additionally, one should ask the following question: When should a group signature σ produced by some user i^*, $i^* \in [1, n]$ be considered as valid? In fact, there are two alternative approaches: (1) σ is valid if i^* belonged to the group at the moment it invoked GSign to generate σ, or (2) σ is valid if i^* belongs to the group at the moment GVrfy is invoked to check its validity. In both cases the group signature scheme should be able to treat different revocation epochs t to account for different executions of the revocation procedures. That is, if some member i^* is revoked, the group manager should be able to perform a revocation operation resulting in the update of the remaining members' keys $\boldsymbol{gsk}[i]$ or of the group public key gpk. In case (1) the group public key gpk may stay the same across multiple revocation epochs but remaining group members i should be able to execute some update procedure UpdM to compute $(\boldsymbol{gsk}[i], t+1)$, i.e. a valid signing key for epoch $t+1$, out of $(\boldsymbol{gsk}[i], t)$. This procedure should be inaccessible to the revoked member i^* being in possession of $(\boldsymbol{gsk}[i^*], t)$. This implies some sort of interaction between remaining members and the group manager. In case (2) the signing keys $\boldsymbol{gsk}[i]$ may stay the same while the group manager should be able to compute $(gpk, t + 1)$, i.e. the group public key for epoch $t + 1$, from (gpk, t). In the following we extend and update the syntax of dynamic group signature schemes from Definition 2.9 to address the revocation procedure. The new *update information* $\boldsymbol{upd}$, which is public and modifiable only by the group manager, without necessarily being part of the group public key, is used to keep track on membership changes. The group manager can modify $\boldsymbol{upd}$ upon the admission and the revocation of group members. For the latter task a new algorithm Revoke is introduced. Additionally, group members can update their secret signing keys using the new algorithm UpdM.

Key generation. The algorithm GKg is modified to additionally output public *update information* $\boldsymbol{upd}$, which is initially empty.

Join protocol. The protocol Join allows the group manager to modify gpk and $\boldsymbol{upd}$ at the end of its JoinM part of the protocol.

Revocation procedure. The randomized *revocation algorithm* Revoke takes as input the group manager's secret key $gmsk$, the identity $i \in [1, n]$ of a member to be revoked, the registration entry $\boldsymbol{reg}[i]$, and the current update information $\boldsymbol{upd}$, and results in a possible update of gpk and $\boldsymbol{upd}$.

Update procedure. The randomized *update algorithm* UpdM takes as input the current secret signing key $\boldsymbol{gsk}[i]$ and at least the update information $\boldsymbol{upd}$, and results in the modification of $\boldsymbol{gsk}[i]$.

The algorithms for signature generation/verification and the opening procedure remain the same. It is implicitly assumed that algorithms GVrfy and Open use the most recent version of the group public key gpk, whereas algorithm GSign uses the up-to-date secret signing keys $\boldsymbol{gsk}[i]$.

2.2.3. Adversary Model and Oracles for Dynamic Schemes

Our formal definitions of security for dynamic group signature schemes $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ will be provided through probabilistic experiments $\mathsf{Expt}_{\Gamma,\mathcal{A}}(1^\kappa)$ as mentioned in Section 1.5. In general, $\mathcal{A}$ will be given access to (a subset of) the following oracles:

Add user oracle. The *add user oracle* $\mathsf{AddU}(\cdot)$ takes as input an identity $i \in [1, n]$, which does not belong to any member of the group and executes $(\mathsf{JoinM}(gmsk, i), \mathsf{JoinU}(gpk, i))$ to compute $(\boldsymbol{reg}[i], \boldsymbol{gsk}[i])$.

Join oracles. The *join oracle* $\mathsf{JoinM}(gmsk, \cdot)$ takes as input an identity $i \in [1, n]$ and executes part $\mathsf{JoinM}(gmsk, i)$ of the Join protocol, which eventually leads to the computation of $\boldsymbol{reg}[i]$.
The *join oracle* $\mathsf{JoinU}(gpk, \cdot)$ takes as input an identity $i \in [1, n]$ and executes part $\mathsf{JoinU}(gpk, i)$ of the Join protocol, which eventually leads to the computation of $\boldsymbol{gsk}[i]$.
These join oracles are defined only if i does *not* already belong to some group member, previously admitted via the AddU, JoinM, or JoinU oracle.

Corruption oracle. The *corruption oracle* $\mathsf{Corrupt}(\cdot)$ takes as input an identity $i \in [1, n]$ and returns the secret signing key $\boldsymbol{gsk}[i]$. This oracle is defined only if i belongs to some group member, previously admitted via the AddU or JoinU oracle.

Signing oracle. The *signing oracle* $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$ takes as input an identity-message pair (i, m) with $i \in [1, n]$, and returns the output of the group signing algorithm $\mathsf{GSign}(\boldsymbol{gsk}[i], m)$. This oracle is defined only if i belongs to some group member, previously admitted via the AddU or JoinU oracle.

Opening oracle. The *opening oracle* $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$ takes as input a message-signature pair (m, σ), and returns the output of the opening algorithm $\mathsf{Open}(gmsk, m, \sigma, \boldsymbol{reg})$.

Since the group is initially empty $\mathcal{A}$ should be given the possibility to add new members to the group. The add user oracle $\mathsf{AddU}(\cdot)$ models the honest admission process by which a new member i receives its secret signing key $\boldsymbol{gsk}[i]$ and the group manager the corresponding registration information $\boldsymbol{reg}[i]$. Note that $\mathcal{A}$ does not learn these secrets, i.e. $\mathcal{A}$ only knows that member i has been added to the group. The $\mathsf{JoinU}(gpk, \cdot)$ oracle also allows for admission of an honest member i to the group, except that now $\mathcal{A}$ can possibly execute the group manager's part of the protocol and thus compute the corresponding registration entry $\boldsymbol{reg}[i]$ but not the secret signing key of the admitted member.

In contrast, $\mathsf{JoinM}(gmsk, \cdot)$ oracle allows $\mathcal{A}$ to participate in the join protocol on behalf of the prospective group member i and obtain $\boldsymbol{gsk}[i]$.

In practice, the Join protocol may have several rounds of interaction. Our oracles $\mathsf{JoinM}(gmsk, \cdot)$ and $\mathsf{JoinU}(gpk, \cdot)$ are assumed to execute these rounds sequentially, i.e. no new invocation call to the oracle is processed until the oracle completes its interaction from the previous call. It is implicitly assumed that in a multi-round Join protocol each of these oracles passes its running state between the rounds and answers incoming protocol messages until the interaction completes, in which case the running state is re-set.

The oracles $\mathsf{Corrupt}(\cdot)$ and $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$ are identical to those defined for static group signatures, except that they check the existence of the particular member i for which the queries are asked. The oracle $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$ differs in the additional parameter $\boldsymbol{reg}$ but otherwise proceeds identically to the static case.

To keep the exposition simple we do not address oracles that could be available to the adversary in schemes that support membership revocation. We will discuss the additional constraints with regard to the security of such schemes informally.

2.2.4. Anonymity Definitions for Dynamic Schemes

In this section we introduce the anonymity definitions for dynamic group signatures schemes $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ building upon such requirements and associated experiments for static signature schemes introduced in Section 2.1.3.

Insider Anonymity

The notion of insider anonymity for dynamic signature schemes provides the adversary $\mathcal{A}$ with all public information about the deployed group signature scheme Γ and access to the oracles $\mathsf{AddU}(\cdot)$, $\mathsf{JoinM}(gmsk, \cdot)$, $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$, and $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$, whose behavior has been specified in Section 2.1.2. The associated experiment proceeds widely similar to that for static schemes. An important remark here is that since $\mathcal{A}$ is given access to the oracle $\mathsf{JoinM}(gmsk, \cdot)$ the experiment has to restrict the identities i_0 and i_1 output by $\mathcal{A}$ to be uncorrupted identities, previously admitted through the $\mathsf{AddU}(\cdot)$ oracle. Otherwise, if these identities would have been admitted through the $\mathsf{JoinM}(gmsk, \cdot)$ oracle then the experiment would not know the corresponding secret signing keys in order to compute the challenge signature.

Definition 2.11 (Insider Anonymity : Dynamic) A group signature scheme $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ provides *insider-anonymity* if for all PPT adversaries $\mathcal{A} =$

$(\mathcal{A}_1, \mathcal{A}_2)$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{I-AN}}_{\Gamma,\mathcal{A}}(1^\kappa) = \left| \Pr\left[\mathsf{Expt}^{\text{I-AN}}_{\Gamma,\mathcal{A}}(1^\kappa) = 1 \right] - \frac{1}{2} \right| .$$

The associated I-AN-*experiment* $\mathsf{Expt}^{\text{I-AN}}_{\Gamma,\mathcal{A}}(1^\kappa)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa)$ is executed to produce $(gpk, gmsk, \boldsymbol{reg})$.

Attack Stage I. Adversary $\mathcal{A}_1$ receives gpk.

1. $\mathcal{A}_1$ can submit queries to the oracles $\mathsf{AddU}(\cdot)$, $\mathsf{JoinM}(gmsk, \cdot)$, $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$, and $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$.
2. $\mathcal{A}_1$ stops and eventually outputs a tuple (st, i_0, i_1, m^*) containing some state information st, two *uncorrupted* identities $i_0, i_1 \in [1, n]$ previously admitted via corresponding $\mathsf{AddU}(\cdot)$ queries, and a challenge message m^*.

Challenge Stage. A bit $b \in \{0, 1\}$ is chosen at random and the signature generation algorithm $\mathsf{GSign}(\boldsymbol{gsk}[i_b], m^*)$ is executed to produce the challenge group signature σ^*.

Attack Stage II. Adversary $\mathcal{A}_2$ receives (st, σ^*).

1. $\mathcal{A}_2$ can submit queries to the oracles $\mathsf{AddU}(\cdot)$, $\mathsf{JoinM}(gmsk, \cdot)$, $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$, and $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$ as before, subject to the following restrictions:
   a) The corruption oracle $\mathsf{Corrupt}(\cdot)$ ignores queries of the form i_0 and i_1.
   b) The opening oracle $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$ ignores queries of the form (m^*, σ^*).
2. $\mathcal{A}_2$ stops and eventually outputs a bit b^*.

Output: If $b^* = b$ then the experiment outputs 1, otherwise it outputs 0. $\diamondsuit$

Full Anonymity

The strongest notion of full anonymity for dynamic schemes also remains widely identical to that of static schemes. The only difference is that since the scheme is dynamic the adversary $\mathcal{A}$ cannot be given all secret signing keys of users as input in advance. Hence, $\mathcal{A}$ is provided with an unrestricted access to the corruption oracle $\mathsf{Corrupt}(\cdot)$ from which it can obtain all these keys. We proceed with the formalization of full anonymity for dynamic schemes in Definition 2.12, which essentially resembles the definition given by Bichsel et al. [31].

Definition 2.12 (Full Anonymity : Dynamic) A group signature scheme $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ provides *full anonymity* if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{F-AN}}_{\Gamma,\mathcal{A}}(1^\kappa) = \left| \Pr\left[\mathsf{Expt}^{\text{F-AN}}_{\Gamma,\mathcal{A}}(1^\kappa) = 1 \right] - \frac{1}{2} \right| .$$

The associated F-AN-*experiment* $\mathsf{Expt}^{\text{F-AN}}_{\Gamma,\mathcal{A}}(1^\kappa, n)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa)$ is executed to produce $(gpk, gmsk, \boldsymbol{reg})$.

Attack Stage I. Adversary $\mathcal{A}_1$ receives gpk.

1. $\mathcal{A}_1$ can submit queries to the oracles $\mathsf{AddU}(\cdot)$, $\mathsf{JoinM}(gmsk, \cdot)$, $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$, and $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$.
2. $\mathcal{A}_1$ stops and eventually outputs a tuple (st, i_0, i_1, m^*) containing some state information st, two identities $i_0, i_1 \in [1, n]$ previously admitted via corresponding $\mathsf{AddU}(\cdot)$ queries, and a challenge message m^*.

Challenge Stage. A bit $b \in \{0, 1\}$ is chosen at random and the signature generation algorithm $\mathsf{GSign}(\boldsymbol{gsk}[i_b], m^*)$ is executed to produce the challenge group signature σ^*.

Attack Stage II. Adversary $\mathcal{A}_2$ receives (st, σ^*).

1. $\mathcal{A}_2$ can submit queries to the oracles $\mathsf{AddU}(\cdot)$, $\mathsf{JoinM}(gmsk, \cdot)$, $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$, and $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$ as before, subject to the following restriction:
   a) The opening oracle $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$ ignores queries of the form (m^*, σ^*).
2. $\mathcal{A}_2$ stops and eventually outputs a bit b^*.

Output: If $b^* = b$ then the experiment outputs 1, otherwise it outputs 0. ◊

2.2.5. Traceability Definitions for Dynamic Schemes

In this section we revisit definitions of traceability for dynamic group signature schemes. Recall that these attacks aim at generation of group signatures that cannot be opened by the group manager. Our exposition shows that not all flavors of traceability introduced for static schemes can be applied in the context of dynamic schemes. In particular, while the notion of insider traceability remains widely similar, the notion of full traceability cannot be achieved in the same strong sense as for static schemes.

Insider Traceability

The notion of insider traceability in dynamic group signature schemes considers traceability attacks by members of the group, while assuming that the group manager remains honest. We formalize this notion in Definition 2.13 by updating the traceability experiment for static schemes to fit the new adversarial model. The basic idea remains, however, the same. In particular, an adversary $\mathcal{A}$ can admit all group members and learn their secret signing keys through the $\mathsf{JoinM}(gmsk, \cdot)$ oracle. Thus, $\mathcal{A}$ can also generate valid group signatures. It can further ask the group manager to open any message-signature pair of its choice through the opening oracle $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$.

Definition 2.13 (Insider Traceability : Dynamic) A group signature scheme $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ provides *insider traceability* if for all probabilistic, polynomial-time adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{I-TR}}_{\Gamma,\mathcal{A}}(1^\kappa) = \Pr\left[\mathsf{Expt}^{\text{I-TR}}_{\Gamma,\mathcal{A}}(1^\kappa) = 1\right].$$

The associated I-TR-*experiment* $\mathsf{Expt}^{\text{I-TR}}_{\Gamma,\mathcal{A}}(1^\kappa)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa)$ is executed to produce $(gpk, gmsk, \boldsymbol{reg})$.

Attack Stage. Adversary $\mathcal{A}$ receives gpk.

1. $\mathcal{A}$ can submit queries to the oracles $\mathsf{JoinM}(gmsk, \cdot)$, and $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$.
2. $\mathcal{A}$ stops and eventually outputs a message-signature pair (m^*, σ^*).

Output. If $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$ and $\mathsf{Open}(gmsk, m^*, \sigma^*, \boldsymbol{reg}) = 0$ then the experiment outputs 1, otherwise it outputs 0. $\diamondsuit$

On Impossibility of Full Traceability for Dynamic Schemes

The strongest notion of full traceability, which extends the insider traceability by allowing the adversary $\mathcal{A}$ to additionally learn the secret key of the group manager $gmsk$ (and possibly the registration list $\boldsymbol{reg}$) *cannot* be achieved in dynamic schemes if one follows their specification from Definition 2.9. The reason is obvious: If $\mathcal{A}$ knows $gmsk$ it can pick any identity $i^* \in [1, n]$ which has not been admitted to the group yet and execute the corresponding joining protocol $(\mathsf{JoinM}(gmsk, i^*), \mathsf{JoinU}(gpk, i^*))$ to obtain $(\boldsymbol{reg}[i^*], \boldsymbol{gsk}[i^*])$, which would remain unknown to the experiment. $\mathcal{A}$ can then execute $\mathsf{GSign}(\boldsymbol{gsk}[i^*], m^*)$ on an arbitrary message m^* to compute a signature σ^*, and output (m^*, σ^*), which will pass the verification check $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$ but for which the (honestly executed) opening algorithm $\mathsf{Open}(gmsk, m^*, \sigma^*, \boldsymbol{reg})$ will output 0. The reason why (m^*, σ^*) cannot be opened is that the honest opening procedure is executed by the experiment, which does not know about the existence of a group member with identity i^*. That is the corresponding registration entry $\boldsymbol{reg}[i^*]$ does not exist in the registration list $\boldsymbol{reg}$ maintained by the experiment and the opening procedure will output 0 due to its correctness property (per Definition 2.10). There is not much one can do about this attack. Therefore, the strongest meaningful definition of traceability for dynamic schemes is that of insider traceability.

2.2.6. Non-Frameability Definitions for Dynamic Schemes

In this section we revisit the notion of non-frameability, which considers attacks performed by a coalition of group members aiming to produce a valid group signature which then opens to some other member of the group. We show how to adopt definitions for insider non-frameability and full non-frameability to the extended functionality and adversary model of dynamic schemes.

Insider Non-Frameability

In order to define insider non-frameability for dynamic group signature schemes we extend the corresponding experiment to fit the constraints of the dynamic setting. The experiment still captures meaningful framing attacks by prohibiting $\mathcal{A}$ from using the secret signing key $\boldsymbol{gsk}[i^*]$ directly to compute the message-signature pair (m^*, σ^*). In contrast to static schemes, where $\mathcal{A}$ could obtain $\boldsymbol{gsk}[i^*]$ from the respective $\mathsf{Corrupt}(\cdot)$ oracle only, a dynamic scheme offers $\mathcal{A}$ an additional possibility to learn $\boldsymbol{gsk}[i^*]$ by querying i^* to the $\mathsf{JoinM}(gmsk, \cdot)$ oracle. This is why the experiment has to ensure that no such query was submitted by $\mathcal{A}$ in the course of attack.

Definition 2.14 (Insider Non-Frameability: Dynamic) A group signature scheme $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ provides *insider non-frameability* if for all PPT adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{I-NF}}_{\Gamma,\mathcal{A}}(1^\kappa) = \Pr\left[\mathsf{Expt}^{\text{I-NF}}_{\Gamma,\mathcal{A}}(1^\kappa) = 1\right].$$

The associated I-NF-*experiment* $\mathsf{Expt}^{\text{I-NF}}_{\Gamma,\mathcal{A}}(1^\kappa)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa)$ is executed to produce $(gpk, gmsk, \boldsymbol{reg})$.

Attack Stage. Adversary $\mathcal{A}$ receives gpk.

1. $\mathcal{A}$ can submit queries to the oracles $\mathsf{AddU}(\cdot)$, $\mathsf{JoinM}(gmsk, \cdot)$, $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$, and $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$.
2. $\mathcal{A}$ stops and eventually outputs a message-signature pair (m^*, σ^*).

Output. If all of the following holds then the experiment outputs 1:

1. $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$ and $\mathsf{Open}(gmsk, m^*, \sigma^*, \boldsymbol{reg}) = i^*$ with $i^* \in [1, n]$
2. $\mathcal{A}$ did not submit i^* to $\mathsf{JoinM}(gmsk, \cdot)$
3. $\mathcal{A}$ did not submit i^* to $\mathsf{Corrupt}(\cdot)$
4. $\mathcal{A}$ did not submit (i^*, m^*) to $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$.

Otherwise it outputs 0. ◊

Full Non-Frameability

Our definition of full non-frameability for dynamic group signature schemes extends the corresponding experiment for static schemes from Section 2.1.5 to fit the dynamic model. This strong definition still makes sense for dynamic schemes since, intuitively, the additional knowledge of $gmsk$ (and $\boldsymbol{reg}$) should not allow the adversary to frame uncorrupted group members. In the corresponding experiment $\mathcal{A}$ is provided with the access to the $\mathsf{JoinU}(gpk, \cdot)$ oracle and can thus admit honest members to the group while misbehaving during the joining protocol. Therefore, there is no need to provide $\mathcal{A}$ with $\mathsf{AddU}(\cdot)$ and $\mathsf{JoinM}(gmsk, \cdot)$ oracles. Since $\mathcal{A}$ in the role of the corrupted group manager also obtains control over the registration list $\boldsymbol{reg}$ the opening oracle $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$ can be omitted, because $\mathcal{A}$ can open honestly generated signatures on its own. By giving such control over the registration list $\boldsymbol{reg}$ to $\mathcal{A}$ the experiment should nevertheless be able to evaluate the success of $\mathcal{A}$ in its attack. This evaluation requires that the experiment opens the message-signature pair (m^*, σ^*) output by $\mathcal{A}$ at the end of the attack stage. Given that $\mathcal{A}$ should have control over the registration list we require that in addition to (m^*, σ^*) the adversary outputs some list $\boldsymbol{reg}^*$, which will be used to evaluate its success. Another important aspect is that $\mathcal{A}$ could possibly admit i^* as a new group member such that the experiment remains unaware of this admission, e.g. when $\mathcal{A}$ executes both parts of the join protocol for i^*, which has not been admitted through the $\mathsf{JoinU}(gpk, \cdot)$ oracle before. This will allow $\mathcal{A}$ to succeed in a trivial way since $\mathcal{A}$ could specify the secret signing key of i^* on its own, without the experiment being able to recognize this as a corruption of i^*. Therefore, the experiment additionally checks that $\boldsymbol{gsk}[i^*] \neq \varepsilon$, i.e. that the secret signing key of i^* exists and is known to the experiment. This is the case only if i^* has been admitted through the $\mathsf{JoinU}(gpk, \cdot)$ oracle before. We proceed with the formalization of this notion in Definition 2.15.

Definition 2.15 (Full Non-Frameability : Dynamic) A group signature scheme $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open})$ provides *full non-frameability* if for all PPT adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa) = \Pr\left[\mathsf{Expt}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa) = 1\right].$$

The associated F-NF-*experiment* $\mathsf{Expt}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa)$ is executed to produce $(gpk, gmsk, \boldsymbol{reg})$.

Attack Stage. Adversary $\mathcal{A}$ receives $(gpk, gmsk, \boldsymbol{reg})$.

1. $\mathcal{A}$ can submit queries to the oracles $\mathsf{JoinU}(gpk, \cdot)$, $\mathsf{Corrupt}(\cdot)$, and $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$.
2. $\mathcal{A}$ stops and eventually outputs a tuple $(m^*, \sigma^*, \boldsymbol{reg}^*)$.

Output. If all of the following holds then the experiment outputs 1:

1. $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$ and $\mathsf{Open}(gmsk, m^*, \sigma^*, \boldsymbol{reg}^*) = i^*$ with $i^* \in [1, n]$
2. $\boldsymbol{gsk}[i^*] \neq \varepsilon$
3. $\mathcal{A}$ did not submit i^* to $\mathsf{Corrupt}(\cdot)$
4. $\mathcal{A}$ did not submit (i^*, m^*) to $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$.

Otherwise it outputs 0. ◊

2.3. Group Signature Schemes with Verifiable Opening

The opening procedure of a group signature scheme Γ can be executed by the manager of the group in order to identify the signer i for some given message-signature pair (m, σ). In ordinary group signatures this procedure outputs only the identity $i \in [1, n]$ (or 0 to indicate a failure). In scenarios where the group manager could attempt to falsely accuse some particular member i of having signed message m although i never did so, this basic opening procedure may not be sufficient. Indeed, nothing prevents a misbehaving group manager from outputting arbitrary identities i as a claimed result of the opening procedure. Therefore, if dishonest group managers represent a potential threat for the application then it would be desirable to have group signature schemes offering a *proof* τ that the output identity i of the opening procedure indeed belongs to the claimed signer of m, and some further verification algorithm that could *judge* whether τ is a correct proof for the statement that "identity i belongs to the group member who generated signature σ on the message m". In this section we extend definitions for ordinary group signature schemes towards schemes with verifiable opening (VO-schemes). We present definitions for dynamic VO-schemes, which can also be easily applied to static schemes.

2.3.1. Algorithms of VO-Schemes and Their Correctness Property

We define group signature schemes with verifiable opening in Definition 2.16 by modifying the opening and adding the judgement procedure. Since the modified opening procedure outputs a proof τ the corresponding algorithm Open is assumed to be randomized. Instead, the new algorithm Judge which either accepts or rejects the proof is deterministic. We explicitly omit definitions of the key generation algorithm GKg, join protocol Join, and the group signing / verifying algorithms GSign / GVrfy, which remain identical to the case of ordinary group signatures.

Definition 2.16 (Group Signature Scheme with Verifiable Opening) A group signature scheme with *verifiable opening* $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open}, \mathsf{Judge})$ is a group signature scheme with the modified algorithm Open and a new algorithm Judge defined in the following:

Opening procedure. The randomized *opening* algorithm Open takes as input the group manager's secret key $gmsk$, a message m, a signature σ (and if the scheme is dynamic the registration list $\boldsymbol{reg}$), and returns either a pair (i, τ) containing a signer's identity $i \in [1, n]$ and a proof τ, or $(0, \perp)$ (to indicate a failure).

Judgement procedure. The deterministic *judgement* algorithm Judge takes as input the group public key gpk, a message m, a signature σ, an identity $i \in [1, n]$, and a proof τ and returns either 1 (to indicate that the proof is valid) or 0 (otherwise). $\diamondsuit$

Note that syntax of the above algorithms and protocols can be extended to address membership revocation, similarly to the case of ordinary dynamic schemes, mentioned in Section 2.2.2.

The following correctness property of a VO-scheme says that any opening procedure, performed on a group signature, which was generated by some valid member of the group, results in the identity of that member and the corresponding proof that can be successfully verified by the judgement algorithm.

Definition 2.17 (Correctness : VO) A group signature scheme $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open}, \mathsf{Judge})$ is *correct* if for all group public keys gpk, group manager's secret keys $gmsk$, secret signing keys $\boldsymbol{gsk}[i]$, and registration entries $\boldsymbol{reg}[i]$ (generated as in Definition 2.10), and all messages $m \in \{0, 1\}^*$, all of the following holds:

1. $\mathsf{GVrfy}(gpk, m, \mathsf{GSign}(\boldsymbol{gsk}[i], m)) = 1$
2. $\mathsf{Open}(gmsk, m, \mathsf{GSign}(\boldsymbol{gsk}[i], m), \boldsymbol{reg}[i]) = (i, \tau)$ with $i > 0$
3. $\mathsf{Judge}(gpk, m, \sigma, i, \tau) = 1$. $\diamondsuit$

2.3.2. Optional Algorithms for User PKI

The above definition of group signatures with verifiable opening procedure implicitly assumes that identity i corresponds to some particular user that can be blamed of having produced the group signature σ. Many group signature schemes offering verifiable opening assume that proof τ contains some information that can be publicly linked to the claimed identity i. This, however, implies that information contained in τ must have been authenticated by the user i. Several group signature schemes make this binding more explicit by considering a public key infrastructure (PKI) for the admitted group members. This *user PKI* helps to actually verify, whether i is the purported signer of the group signature σ. The explicit use of such PKI in group signature schemes requires modifications to their syntax in order to accommodate the additional algorithms of the user PKI. In particular, the user PKI should be setup independently of the group signature scheme. That is, users should be able to generate their key pairs $(\boldsymbol{usk}[i], \boldsymbol{upk}[i])$ that can then be assumed to be bound to their respective identities i by the means of PKI certification. It is important to mention that certification of public keys $\boldsymbol{upk}[i]$ is assumed to be done by some certification authority, which is independent of any authorities (e.g. group manager) participating in the group signature scheme. Thus, group signature schemes with explicit user PKI should include the following algorithm:

User key generation. The *user key generation* algorithm UKg takes as input the security parameter 1^κ, $\kappa \in \mathbb{N}$ and outputs user's private/public key pair $(\boldsymbol{usk}[i], \boldsymbol{upk}[i])$.

The entire user PKI can represented by the list $\boldsymbol{upk}$ containing public keys of the users. It will be generally assumed that $\boldsymbol{upk}$ is public (but not modifiable by the adversary) and that any $\boldsymbol{upk}[i]$ can be retrieved using the corresponding identity i. That is, it suffices to give i to algorithms and protocols of the group signature scheme in order for them to obtain $\boldsymbol{upk}[i]$. Another alternative would be to explicitly add $\boldsymbol{upk}[i]$ as input to the respective algorithms. Note that in dynamic VO-schemes $\boldsymbol{upk}[i]$ is in general used by the algorithms Open and Judge as well as in the group manager's part JoinM of the joining protocol, whereas $\boldsymbol{usk}[i]$ is used in the user's part JoinU of the joining protocol.

Remark 2.3.1 Explicit consideration of user PKI can also be useful in static VO-schemes. In this case all members added to the group during the key generation procedure should be registered PKI users, that is the corresponding user key generation algorithm UKg for each prospective member i would have to be executed prior to GKg or as part of GKg. This would require some sort of interaction between the group manager and the prospective group member during the key generation procedure, unless the group manager is trusted to generate $(\boldsymbol{usk}[i], \boldsymbol{upk}[i])$ on behalf of every i and erase $\boldsymbol{usk}[i]$ after providing i with $\boldsymbol{usk}[i]$ (and $\boldsymbol{gsk}[i]$).

2.3.3. Adversary Model and Oracles for VO-Schemes

The adversary model for VO-schemes remains widely similar to ordinary group signatures. That is, we still consider PPT adversaries $\mathcal{A}$ and grant them access to the same oracles as before, except that the corresponding opening oracles — $\mathsf{Open}(gmsk, \cdot, \cdot)$ for static schemes and $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$ for dynamic schemes — on some query (m, σ) output the result of the opening algorithm of a VO-scheme. That is, the opening oracle on input some message-signature pair (m, σ) will output either (i, τ) or $(0, \perp)$. We refer to Sections 2.1.2 and 2.2.3 for the definitions of the other oracles that will be used in the security experiments for VO-schemes.

In case of VO-schemes, where user PKI is explicitly used as part of the specification, the adversary can also be given access to special oracles modeling attacks on the user PKI. In particular, the adversary should be able to introduce new users (under adversary's control) to the PKI and corrupt existing PKI users by obtaining their PKI-certified secret keys $\boldsymbol{usk}[i]$. The first ability of the adversary will be modeled through the new oracle AddPKI described below, while the ability to corrupt $\boldsymbol{usk}[i]$ will be modeled by the appropriate modification of the $\mathsf{Corrupt}$ oracles as described below.

Add PKI oracle. The *add PKI oracle* AddPKI takes as input an identity $i \in [1, n]$ and some string upk. If condition $\boldsymbol{upk}[i] = \varepsilon$ holds then the oracle sets $\boldsymbol{upk}[i] = upk$; otherwise it ignores the query. (Note that condition $\boldsymbol{upk}[i] = \varepsilon$ ensures that i is a new user. Any user added to PKI via this oracle will be treated as corrupted by the adversary. However, it is not necessary for the adversary to actually present or proof possession of some suitable secret key usk corresponding to upk.)

Corruption oracle. The *corruption oracle* $\mathsf{Corrupt}(\cdot)$ (defined for static schemes in Section 2.1.2 and for dynamic schemes in Section 2.2.3) is modified such that in addition to the secret signing key $\boldsymbol{gsk}[i]$ it also returns the PKI-certified secret key $\boldsymbol{usk}[i]$ (unless the corresponding public key $\boldsymbol{upk}[i]$ was established through a query to AddPKI in which case $\boldsymbol{usk}[i]$ is not known).

Add user and join oracles. In dynamic VO-schemes $\mathsf{AddU}(\cdot)$ and $\mathsf{JoinU}(gpk, \cdot)$ oracles (defined in Section 2.2.3) that can be used to admit an uncorrupted prospective member i to the group should be modified to check that $\boldsymbol{upk}[i] = \varepsilon$ holds and if so generate the PKI-certified key pair $(\boldsymbol{usk}[i], \boldsymbol{upk}[i]) \leftarrow_R \mathsf{UKg}(1^\kappa)$ for i before further processing the request. (This guarantees that every uncorrupted group member i has a PKI-certified key pair.) Furthermore, the $\mathsf{JoinM}(gmsk, \cdot)$ oracle that can be used by the adversary to admit member i under its control should also check whether $\boldsymbol{upk}[i] = \varepsilon$ holds, thus ensuring that such members also have been registered within the PKI.

In our security definitions for group signature schemes with verifiable opening procedure we will not consider user PKI as an explicit part of the scheme. Yet, where necessary, we will give remarks on how user PKI would affect these definitions.

2.3.4. Anonymity Definitions for VO-Schemes

The anonymity notions for group signatures with verifiable opening are the same as for ordinary group signatures schemes. The corresponding definitions of insider anonymity from Definitions 2.3 and 2.11 apply to static and dynamic VO-schemes, respectively. Also, our formal definitions of full anonymity from Definitions 2.4 and 2.12 can be used with VO-schemes.

If user PKI is explicitly used then definitions of full anonymity for the corresponding VO-scheme should allow the adversary to obtain all PKI-certified secret keys from $\boldsymbol{usk}$ (in addition to their secret signing keys in $\boldsymbol{gsk}$). In case of insider anonymity where corruption of signers i_0 and i_1 used in the generation of the challenge group signature σ^* is prohibited the adversary can still be given all keys from $\boldsymbol{usk}$. This is because, intuitively, secret signing keys $\boldsymbol{gsk}[i]$ used in the generation of group signatures should be independent of the PKI-certified keys.

2.3.5. Traceability Definitions for VO-Schemes

The modification of the opening procedure in VO-schemes towards the output of proof τ and the introduction of the publicly executable algorithm Judge influences definitions of traceability for static and dynamic schemes from Sections 2.1.4 and 2.2.5, respectively, where the success of the adversary was evaluated based on the outputs of the opening procedure. In particular, the corresponding experiments for insider traceability and full traceability (the latter notion is still meaningful only for static VO-schemes) have to be modified to account for traceability attacks, in which the judgement procedure fails although the opening procedure succeeds. Note that in schemes with explicit user PKI both definitions can remain as is since the adversary can obtain control over all users (and thus their PKI-certified keys) anyway, upon their admission to the group using the $\mathsf{JoinM}(gmsk, \cdot)$ oracle.

Insider Traceability

We modify definition of insider traceability to address dynamic VO-schemes. The corresponding definition for static schemes can be obtained in a similar way.

Definition 2.18 (Insider Traceability : VO) A group signature scheme $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open}, \mathsf{Judge})$ provides *insider traceability* if for all PPT adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{I-TR}}_{\Gamma,\mathcal{A}}(1^\kappa) = \Pr\left[\mathsf{Expt}^{\text{I-TR}}_{\Gamma,\mathcal{A}}(1^\kappa) = 1\right].$$

The associated I-TR-*experiment* $\mathsf{Expt}^{\text{I-TR}}_{\Gamma,\mathcal{A}}(1^\kappa)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa)$ is executed to produce $(gpk, gmsk, \boldsymbol{reg})$.

Attack Stage. Adversary $\mathcal{A}$ receives gpk.

1. $\mathcal{A}$ can submit queries to the oracles $\mathsf{JoinM}(gmsk, \cdot)$ and $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$.
2. $\mathcal{A}$ stops and eventually outputs a message-signature pair (m^*, σ^*).

Output. The experiment runs $\mathsf{Open}(gmsk, m^*, \sigma^*, \boldsymbol{reg})$ to obtain (i^*, τ^*). It then outputs 1 if all of the following holds:

1. $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$
2. $i^* = 0$ or $\mathsf{Judge}(gpk, m^*, \sigma^*, i^*, \tau^*) = 0$.

Otherwise, the experiment outputs 0. ◊

Full Traceability

Our arguments on the impossibility of full traceability of dynamic group signature schemes from Section 2.2.5 hold also for dynamic VO-schemes. In Definition 2.19 we thus define full traceability for static VO-schemes only.

Definition 2.19 (Full Traceability : Static VO) A group signature scheme $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open}, \mathsf{Judge})$ provides *full traceability* if for all PPT adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{F-TR}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = \Pr\ \ \mathsf{Expt}^{\text{F-TR}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = 1\ \ .$$

The associated F-TR-*experiment* $\mathsf{Expt}^{\text{F-TR}}_{\Gamma,\mathcal{A}}(1^\kappa, n)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa, n)$ is executed to produce $(gpk, gmsk, \boldsymbol{gsk})$.

Attack Stage. Adversary $\mathcal{A}$ receives $(gpk, \boldsymbol{gsk}, gmsk)$. $\mathcal{A}$ stops and eventually outputs a message-signature pair (m^*, σ^*).

Output. The experiment runs $\mathsf{Open}(gmsk, m^*, \sigma^*)$ to obtain (i^*, τ^*). It then outputs 1 if all of the following holds:

1. $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$
2. $i = 0$ or $\mathsf{Judge}(gpk, m^*, \sigma^*, i^*, \tau^*) = 0$.

Otherwise, the experiment outputs 0. ◊

2.3.6. Non-Frameability Definitions for VO-Schemes

A publicly executable Judge algorithm for verifying the validity of the opening procedure significantly strengthens the notion of non-frameability. The reason is that now in order to successfully accuse some group member i^* of having produced a signature σ^* on a message m^*, the adversary $\mathcal{A}$ has to output a tuple $(m^*, \sigma^*, i^*, \tau^*)$, which will pass the algorithm Judge and for which (m^*, σ^*) can be successfully verified using GVrfy.

Insider Non-Frameability

We define insider non-frameability for VO-schemes in Definition 2.20. In case of static VO-schemes the second output condition can be dropped. If the VO-scheme comes with explicit user PKI then the adversary can be given access to the $\mathsf{AddPKI}(\cdot, \cdot)$ oracle but the output conditions should also check that this oracle has not been used to register i^*.

Definition 2.20 (Insider Non-Frameability: VO) A group signature scheme $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open}, \mathsf{Judge})$ provides *insider non-frameability* if for all PPT adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{I-NF}}_{\Gamma,\mathcal{A}}(1^\kappa) = \Pr\ \ \mathsf{Expt}^{\text{I-NF}}_{\Gamma,\mathcal{A}}(1^\kappa) = 1\ \ .$$

The associated I-NF-*experiment* $\mathsf{Expt}^{\text{I-NF}}_{\Gamma,\mathcal{A}}(1^\kappa)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa)$ is executed to produce $(gpk, gmsk, \boldsymbol{reg})$.

Attack Stage. Adversary $\mathcal{A}$ receives gpk.

1. $\mathcal{A}$ can submit queries to the oracles $\mathsf{AddU}(\cdot)$, $\mathsf{JoinM}(gmsk, \cdot)$, $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$, and $\mathsf{Open}(gmsk, \cdot, \cdot, \boldsymbol{reg})$.
2. $\mathcal{A}$ stops and eventually outputs a tuple $(m^*, \sigma^*, i^*, \tau^*)$.

Output. If all of the following holds then the experiment outputs 1:

1. $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$ and $i^* \in [1, n]$ and $\mathsf{Judge}(gpk, m^*, \sigma^*, i^*, \tau^*) = 1$
2. $\mathcal{A}$ did not submit i^* to $\mathsf{JoinM}(gmsk, \cdot)$
3. $\mathcal{A}$ did not submit i^* to $\mathsf{Corrupt}(\cdot)$
4. $\mathcal{A}$ did not submit (i^*, m^*) to $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$.

Otherwise the experiment outputs 0. $\Diamond$

Full Non-Frameability

The notion of full non-frameability considers framing attacks by member coalitions which may also include the group manager. In VO-schemes a successful attack can be checked through the algorithm Judge, which does not take any secrets of the group manager. Therefore, the non-frameability experiment for VO-schemes can check whether a framing attack against some member i^* is successful without using $gmsk$ and $\boldsymbol{reg}$. In particular, the experiment for dynamic VO-schemes in Definition 2.21 does not require $\mathcal{A}$ to output a registration list $\boldsymbol{reg}^*$ (unlike the corresponding experiment for ordinary dynamic schemes in Definition 2.15). In case of static VO-schemes the second output condition can be dropped and in case of VO-schemes with explicit user PKI the adversary can be given access to the $\mathsf{AddPKI}(\cdot, \cdot)$ oracle but the output conditions should also check that this oracle has not been used to register i^*.

Definition 2.21 (Full Non-Frameability : VO) A group signature scheme $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open}, \mathsf{Judge})$ provides *full non-frameability* if for all PPT adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa) = \Pr\left[\mathsf{Expt}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa) = 1\right].$$

The associated F-NF-*experiment* $\mathsf{Expt}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa)$ is executed to produce $(gpk, gmsk, \boldsymbol{reg})$.

Attack Stage. Adversary $\mathcal{A}$ receives $(gpk, gmsk, \boldsymbol{reg})$.

1. $\mathcal{A}$ can submit queries to the oracles $\mathsf{JoinU}(gpk, \cdot)$, $\mathsf{Corrupt}(\cdot)$, and $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$.
2. $\mathcal{A}$ stops and eventually outputs a tuple $(m^*, \sigma^*, i^*, \tau^*)$.

Output. If all of the following holds then the experiment outputs 1:

1. $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$ and $i^* \in [1, n]$ and $\mathsf{Judge}(gpk, m^*, \sigma^*, i^*, \tau^*) = 1$
2. $\boldsymbol{gsk}[i^*] = \varepsilon$
3. $\mathcal{A}$ did not submit i^* to $\mathsf{Corrupt}(\cdot)$
4. $\mathcal{A}$ did not submit (i^*, m^*) to $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$.

Otherwise the experiment outputs 0. $\Diamond$

2.4. Group Signature Schemes with Distributed Authorities

In group signature schemes, where both admission of group members and revocation of signer's anonymity are performed by the group manager, a high amount of trust into the single party, which executes this role, becomes necessary. If the group manager's secret key *gmsk* is leaked then the anonymity of all signers is gone. The problem becomes even more severe in the dynamic setting, where the knowledge of *gmsk* would allow an adversary to introduce new group members at will, generate untraceable signatures (see the discussion on impossibility of full traceability for dynamic schemes in Section 2.1.4), and also reveal the anonymity of honestly admitted group members. This negative impact from the leakage of *gmsk* can be mitigated by separating the duties of the group manager into two authorities: The *issuer* with its own (secret) *issuing key ik* that is responsible solely for the admission of new members to the group, and the *opener* with its (secret) *opening key ok* that is given the sole ability to execute the opening procedure and identify signers. We refer to such schemes as DA-schemes.

We observe that separation of duties makes more sense for dynamic schemes, where the issuer must be available beyond the initial key generation procedure, in order to participate in subsequent runs of the joining protocol. In fact, static schemes could safely remove the issuer after the key generation procedure, e.g. by explicitly erasing its secret issuing key. In this section we will mainly discuss dynamic DA-schemes with verifiable opening procedure that were defined in Section 2.3. Our definitions can be easily extended with the additional mechanisms for the explicit use of user PKI and they can also be adopted to static DA-schemes with verifiable opening and ordinary group signature schemes (static and dynamic) without verifiable opening.

2.4.1. Algorithms of DA-Schemes and Their Correctness Property

We define group signatures with distributed authorities in Definition 2.22 by modifying the key generation algorithm GKg, join protocol Join, and open algorithm Open to take into account the two authorities — issuer and opener — with their respective secret keys. Note that the opening procedure in our definition outputs proof τ, in addition to the signer's identity i. That is our definition assumes dynamic schemes with verifiable opening. The corresponding definitions for schemes without verifiable opening can be obtained by omitting τ and making the opening procedure deterministic.

Definition 2.22 (Group Signature Scheme with Distributed Authorities) A group signature scheme with *distributed authorities* $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open}, \mathsf{Judge})$ is a group signature scheme, where the key generation algorithm GKg, join protocol Join, and opening algorithm Open are modified as follows:

Key generation. The randomized *group key generation* algorithm GKg takes as input the security parameter 1^κ, $\kappa \in \mathbb{N}$ and returns a tuple $(gpk, ik, \boldsymbol{reg}, ok)$, where gpk is the *group public key*, ik is the *secret issuing key*, $\boldsymbol{reg}$ is the initially empty *registration list*, and ok is the *secret opening key*. The issuer is given ik and full control over $\boldsymbol{reg}$, while the opener is given ok and read-only access to $\boldsymbol{reg}$.

Join protocol. The randomized Join protocol is a two-party protocol composed of two interactive algorithms $(\mathsf{JoinM}, \mathsf{JoinU})$. JoinM takes as input the secret issuing key ik and identity $i \in [1, n]$, and at the end of the interaction adds a *registration entry* $\boldsymbol{reg}[i]$ to the list $\boldsymbol{reg}$. JoinU takes as input the group public key gpk and identity $i \in [1, n]$, and at the end of the interaction outputs the *secret signing key* $\boldsymbol{gsk}[i]$. If either JoinM or JoinU fails then the respective output is set to $\perp$.

Opening procedure. The randomized *opening* algorithm Open takes as input the secret opening key ok, a message m, a signature σ, and the registration list $\boldsymbol{reg}$, and returns either a pair (i, τ) containing a signer's identity $i \in [1, n]$ and a proof τ, or $(0, \perp)$ (to indicate a failure). $\diamondsuit$

The separation of duties between the issuer and the opener is made explicit by using ik in the join protocol and ok in the opening procedure. The key generation procedure as defined above can be run either by a trusted party which computes the keys and provides them to the authorities, or through a secure two-party protocol between both authorities, depending on a concrete instantiation of the scheme.

The following correctness property of a DA-schemes says that any opening procedure executed by the opener on a group signature, which was generated by some valid member of the group, previously admitted by the issuer, results in the identity of that member and that the corresponding proof (generated by the opener) can be successfully verified by the judgement algorithm.

Definition 2.23 (Correctness : DA) A group signature scheme with distributed authorities $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open}, \mathsf{Judge})$ is *correct* if for all $\kappa, n \in \mathbb{N}$, all outputs $(gpk, ik, \boldsymbol{reg}, ok) \leftarrow \mathsf{GKg}(1^\kappa)$, all outputs $(\boldsymbol{reg}[i], \boldsymbol{gsk}[i]) \leftarrow (\mathsf{JoinM}(ik, i), \mathsf{JoinU}(gpk, i))$ for any $i \in [1, n]$, and all messages $m \in \{0, 1\}^*$, all of the following holds:

1. $\mathsf{GVrfy}(gpk, m, \mathsf{GSign}(\boldsymbol{gsk}[i], m)) = 1$
2. $\mathsf{Open}(ok, m, \mathsf{GSign}(\boldsymbol{gsk}[i], m), \boldsymbol{reg}[i]) = (i, \tau)$ with $i > 0$
3. $\mathsf{Judge}(gpk, m, \sigma, i, \tau) = 1.$ $\diamondsuit$

2.4.2. Adversary Model and Oracles for DA-Schemes

The adversarial model for DA-schemes requires some changes due to the use of two distinct authorities and their respective secret keys. In particular, the oracles $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot],\cdot)$, and $\mathsf{JoinU}(gpk,\cdot)$ remain as before. The changes mainly concern the following oracles:

Add user oracle. The *add user oracle* $\mathsf{AddU}(\cdot)$ takes as input an identity $i \in [1,n]$, which does not belong to any member of the group, executes $(\mathsf{JoinM}(ik,i),\mathsf{JoinU}(gpk,i))$ to compute $(\boldsymbol{reg}[i],\boldsymbol{gsk}[i])$.

Join oracle. The *join oracle* $\mathsf{JoinM}(ik,\cdot)$ takes as input an identity $i \in [1,n]$ and executes part $\mathsf{JoinM}(ik,i)$ of the Join protocol, which eventually leads to the computation of $\boldsymbol{reg}[i]$. This oracle is defined only if i does *not* already belong to some group member, previously admitted via the AddU, JoinM, or JoinU oracles.

Opening oracle. The *opening oracle* $\mathsf{Open}(ok,\cdot,\cdot,\boldsymbol{reg})$ takes as input a message-signature pair (m,σ), and returns the output of the opening algorithm $\mathsf{Open}(ok,m,\sigma,\boldsymbol{reg})$.

Remark 2.4.1 For DA-schemes with explicit user PKI the above modifications of the oracles $\mathsf{AddU}(\cdot)$ and $\mathsf{JoinM}(ik,\cdot)$ should further be augmented with appropriate mechanisms regarding the use of $(\boldsymbol{usk}[i],\boldsymbol{upk}[i])$ in a similar way as described in Section 2.3.3.

In addition to these modified oracles the adversary model should consider adversarial access to the registration list $\boldsymbol{reg}$. In DA-schemes security experiments can provide $\mathcal{A}$ with access to either ik or ok, or to both keys, depending on the assumed authority corruptions. If $\mathcal{A}$ is only given ik, meaning that it can act on behalf of the issuer, then $\mathcal{A}$ should also be able to modify entries in $\boldsymbol{reg}$. If $\mathcal{A}$ is only given ok then it should be able to read registration entries. This separation will be modeled through the following read and write oracles for the registration list $\boldsymbol{reg}$:

Read oracle. The *read oracle* $\mathsf{RReg}(\cdot)$ takes as input an identity $i \in [1,n]$ and outputs the corresponding registration entry $\boldsymbol{reg}[i]$ (which may also be empty).

Read oracle. The *write oracle* $\mathsf{WReg}(\cdot,\cdot)$ takes as input an identity-entry pair (i,ρ) with $i \in [1,n]$ and modifies $\boldsymbol{reg}[i]$ to be ρ.

2.4.3. Anonymity Definitions for DA-Schemes

The anonymity notions for DA-schemes are similar to those of group signatures with one group manager, except that the experiments consider malicious issuers. This models the distinguished property of DA-schemes, namely that anonymity can be revoked only by the opener. In our definitions of insider anonymity and full anonymity for DA-schemes we primarily consider dynamic schemes with verifiable opening. If the DA-scheme comes with explicit user PKI then these definitions can be further updated by providing the adversary with all PKI-certified secret keys in $\boldsymbol{usk}$.

Insider Anonymity

In Definition 2.24 we formalize insider anonymity for DA-schemes by updating the corresponding experiment. In particular, we provide the adversary $\mathcal{A}$ with the secret issuing key ik, write access to $\boldsymbol{reg}$, and access to the oracle $\mathsf{JoinU}(gpk, \cdot)$. In this way we model the sole ability of the opener to revoke anonymity of group members. Using $\mathsf{JoinU}(gpk, \cdot)$ oracle $\mathcal{A}$ can now admit honest group members while acting on behalf of the issuer. Recall that such oracle was not given to the anonymity adversary in dynamic schemes with one group manager as it would have allowed $\mathcal{A}$ to trivially open the challenge message-signature pair (m^*, σ^*). In DA-schemes $\mathcal{A}$ can still query the opening oracle $\mathsf{Open}(ok, \cdot, \cdot, \boldsymbol{reg})$ to test message-signature pairs of its choice. In order to answer these queries the experiment uses contents of $\boldsymbol{reg}$ obtained from the queries to the write oracle $\mathsf{WReg}(\cdot, \cdot)$.

Definition 2.24 (Insider Anonymity : DA) A group signature scheme with distributed authorities $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open}, \mathsf{Judge})$ provides *insider-anonymity* if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{I-AN}}_{\Gamma,\mathcal{A}}(1^\kappa) = \Pr\left[\mathsf{Expt}^{\text{I-AN}}_{\Gamma,\mathcal{A}}(1^\kappa) = 1\right] - \frac{1}{2} .$$

The associated I-AN-*experiment* $\mathsf{Expt}^{\text{I-AN}}_{\Gamma,\mathcal{A}}(1^\kappa)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa)$ is executed to produce $(gpk, ik, \boldsymbol{reg}, ok)$.

Attack Stage I. Adversary $\mathcal{A}_1$ receives $(gpk, ik, \boldsymbol{reg})$.

1. $\mathcal{A}_1$ can submit queries to the oracles $\mathsf{JoinU}(gpk, \cdot)$, $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$, $\mathsf{WReg}(\cdot, \cdot)$, and $\mathsf{Open}(ok, \cdot, \cdot, \boldsymbol{reg})$.
2. $\mathcal{A}_1$ stops and eventually outputs a tuple (st, i_0, i_1, m^*) containing some state information st, two *uncorrupted* identities $i_0, i_1 \in [1, n]$, previously admitted via corresponding $\mathsf{JoinU}(gpk, \cdot)$ queries, and a challenge message m^*.

Challenge Stage. A bit $b \in \{0, 1\}$ is chosen at random and the signature generation algorithm $\mathsf{GSign}(\boldsymbol{gsk}[i_b], m^*)$ is executed to produce the challenge group signature σ^*.

Attack Stage II. Adversary $\mathcal{A}_2$ receives (st, σ^*).

1. $\mathcal{A}_2$ can submit queries to the oracles $\mathsf{JoinU}(gpk, \cdot)$, $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$, $\mathsf{WReg}(\cdot, \cdot)$, and $\mathsf{Open}(ok, \cdot, \cdot, \boldsymbol{reg})$ as before, subject to the following restrictions:
   a) The corruption oracle $\mathsf{Corrupt}(\cdot)$ ignores queries of the form i_0 and i_1.
   b) The opening oracle $\mathsf{Open}(ok, \cdot, \cdot, \boldsymbol{reg})$ ignores queries of the form (m^*, σ^*).
2. $\mathcal{A}_2$ stops and eventually outputs a bit b^*.

Output: If $b^* = b$ then the experiment outputs 1, otherwise it outputs 0. ♢

Full Anonymity

In Definition 2.25 we model full anonymity of DA-schemes. In contrast to insider anonymity, we allow $\mathcal{A}$ to learn secret signing keys $\boldsymbol{gsk}[i]$ of all group members, including i_0 and i_1, by removing restrictions on the corruption oracle $\mathsf{Corrupt}(\cdot)$. Therefore, we can also remove the signing oracle $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$. Our definition of full anonymity is similar to the one proposed by Bellare, Shi, and Zhang [25].

Definition 2.25 (Full Anonymity : DA) A group signature scheme with distributed authorities $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open}, \mathsf{Judge})$ provides *full-anonymity* if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{F-AN}}_{\Gamma,\mathcal{A}}(1^\kappa) = \ \Pr \ \mathsf{Expt}^{\text{F-AN}}_{\Gamma,\mathcal{A}}(1^\kappa) = 1 \ - \frac{1}{2} \ .$$

The associated F-AN-*experiment* $\mathsf{Expt}^{\text{F-AN}}_{\Gamma,\mathcal{A}}(1^\kappa)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa)$ is executed to produce $(gpk, ik, \boldsymbol{reg}, ok)$.

Attack Stage I. Adversary $\mathcal{A}_1$ receives $(gpk, ik, \boldsymbol{reg})$.

1. $\mathcal{A}_1$ can submit queries to the oracles $\mathsf{JoinU}(gpk, \cdot)$, $\mathsf{Corrupt}(\cdot)$, $\mathsf{WReg}(\cdot, \cdot)$, and $\mathsf{Open}(ok, \cdot, \cdot, \boldsymbol{reg})$.
2. $\mathcal{A}_1$ stops and eventually outputs a tuple (st, i_0, i_1, m^*) containing some state information st, two identities $i_0, i_1 \in [1, n]$, previously admitted via corresponding $\mathsf{JoinU}(gpk, \cdot)$ queries, and a challenge message m^*.

Challenge Stage. A bit $b \in \{0, 1\}$ is chosen at random and the signature generation algorithm $\mathsf{GSign}(\boldsymbol{gsk}[i_b], m^*)$ is executed to produce the challenge group signature σ^*.

Attack Stage II. Adversary $\mathcal{A}_2$ receives (st, σ^*).

1. $\mathcal{A}_2$ can submit queries to the oracles $\mathsf{JoinU}(gpk, \cdot)$, $\mathsf{Corrupt}(\cdot)$, $\mathsf{WReg}(\cdot, \cdot)$, and $\mathsf{Open}(ok, \cdot, \cdot, \boldsymbol{reg})$ as before, subject to the following restriction:
    a) The opening oracle $\mathsf{Open}(ok, \cdot, \cdot, \boldsymbol{reg})$ ignores queries of the form (m^*, σ^*).
2. $\mathcal{A}_2$ stops and eventually outputs a bit b^*.

Output: If $b^* = b$ then the experiment outputs 1, otherwise it outputs 0. $\Diamond$

2.4.4. Traceability Definitions for DA-Schemes

The traceability notions for DA-schemes are similar to those of group signatures with one group manager, except that the experiments consider coalitions of corrupted openers and malicious group members. This is because in DA-schemes only issuers should be able to admit new group members due to the separation of duties amongst the both authorities. Our definitions consider primarily DA-schemes with verifiable opening. Note that if the scheme, additionally, comes with mechanisms for user PKI then definitions need not to be updated since the adversary obtains control over all users (and thus their PKI-certified keys $\boldsymbol{usk}[i]$) using the $\mathsf{JoinM}(ik, \cdot)$ oracle.

Insider Traceability

In Definition 2.24 we model insider traceability of DA-schemes by providing $\mathcal{A}$ with the opening key *ok* and the read oracle $\mathsf{RReg}(\cdot)$ to allow $\mathcal{A}$ to open signatures on its own. Additionally, we give $\mathcal{A}$ access to the $\mathsf{JoinM}(ik, \cdot)$ oracle that can be used to admit new group members while learning their secret signing keys.

Definition 2.26 (Insider Traceability : DA) A group signature scheme with distributed authorities $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open}, \mathsf{Judge})$ provides *insider traceability* if for all PPT adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{I-TR}}_{\Gamma,\mathcal{A}}(1^\kappa) = \Pr\left[\mathsf{Expt}^{\text{I-TR}}_{\Gamma,\mathcal{A}}(1^\kappa) = 1\right].$$

The associated I-TR-*experiment* $\mathsf{Expt}^{\text{I-TR}}_{\Gamma,\mathcal{A}}(1^\kappa)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa)$ is executed to produce $(gpk, ik, \boldsymbol{reg}, ok)$.

Attack Stage. Adversary $\mathcal{A}$ receives (gpk, ok).

1. $\mathcal{A}$ can submit queries to the oracles $\mathsf{JoinM}(ik, \cdot)$ and $\mathsf{RReg}(\cdot)$.
2. $\mathcal{A}$ stops and eventually outputs a message-signature pair (m^*, σ^*).

Output. The experiment executes the opening procedure $\mathsf{Open}(ok, m^*, \sigma^*, \boldsymbol{reg})$ to obtain (i^*, τ^*). It then outputs 1 if all of the following holds:

1. $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$
2. $i^* = 0$ or $\mathsf{Judge}(gpk, m^*, \sigma^*, i^*, \tau^*) = 0$.

Otherwise, the experiment outputs 0. $\Diamond$

Remark 2.4.2 If the DA-scheme Γ does not provide verifiable opening then the output conditions of the experiment $\mathsf{Expt}^{\text{I-TR}}_{\Gamma,\mathcal{A}}(1^\kappa)$ should check whether $\mathsf{Open}(ok, m^*, \sigma^*, \boldsymbol{reg}) = 0$ (as in Definition 2.13).

Note that insider traceability is the strongest notion that can be satisfied by a dynamic DA-scheme. In particular, full traceability where $\mathcal{A}$ would receive the issuing key *ik* and write access to the registration list $\boldsymbol{reg}$ cannot be satisfied for same arguments as in Section 2.2.5. The notion of full traceability is still meaningful for static DA-schemes. Its definition can be obtained by providing $\mathcal{A}$ with both keys *ik* and *ok*.

2.4.5. Non-Frameability Definitions for DA-Schemes

The notion of non-frameability in DA-schemes can have several flavors. As before we can think of insider non-frameability, where coalitions of malicious group members should not be able to generate signatures that will then be traced to some honest group member. The strongest notion is still full non-frameability where the adversary is given further access to the secret keys of both group authorities. However, DA-schemes may also allow coalitions of malicious group members and only one authority (either issuer or opener). These latter flavors are clearly weaker than full non-frameability.

Insider Non-Frameability for DA-Schemes

In Definition 2.27 we formalize the notion of insider non-frameability for DA-schemes. This definition is widely similar to full non-frameability of group signatures with one group manager, except that it uses two different secret keys (ik and ok). Note that if the DA-scheme comes with explicit user PKI then the adversary can be given access to the $\mathsf{AddPKI}(\cdot,\cdot)$ oracle but the output conditions should also check that this oracle has not been used to register i^*.

Definition 2.27 (Insider Non-Frameability : DA) A group signature scheme with distributed authorities $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open}, \mathsf{Judge})$ provides *insider non-frameability* if for all PPT adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{I-NF}}_{\Gamma,\mathcal{A}}(1^\kappa) = \Pr\left[\mathsf{Expt}^{\text{I-NF}}_{\Gamma,\mathcal{A}}(1^\kappa) = 1\right].$$

The associated I-NF-*experiment* $\mathsf{Expt}^{\text{I-NF}}_{\Gamma,\mathcal{A}}(1^\kappa)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa)$ is executed to produce $(gpk, ik, \boldsymbol{reg}, ok)$.

Attack Stage. Adversary $\mathcal{A}$ receives gpk.

1. $\mathcal{A}$ can submit queries to the oracles $\mathsf{AddU}(\cdot)$, $\mathsf{JoinM}(ik,\cdot)$, $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot],\cdot)$, and $\mathsf{Open}(ok,\cdot,\cdot,\boldsymbol{reg})$.
2. $\mathcal{A}$ stops and eventually outputs a tuple $(m^*, \sigma^*, i^*, \tau^*)$.

Output. If all of the following holds then the experiment outputs 1:

1. $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$ and $i^* \in [1, n]$ and $\mathsf{Judge}(gpk, m^*, \sigma^*, i^*, \tau^*) = 1$
2. $\mathcal{A}$ did not submit i^* to $\mathsf{JoinM}(ik,\cdot)$
3. $\mathcal{A}$ did not submit i^* to $\mathsf{Corrupt}(\cdot)$
4. $\mathcal{A}$ did not submit (i^*, m^*) to $\mathsf{GSign}(\boldsymbol{gsk}[\cdot],\cdot)$.

Otherwise it outputs 0. $\Diamond$

Remark 2.4.3 If the DA-scheme Γ does not provide verifiable opening then output conditions of the experiment $\mathsf{Expt}^{\text{I-NF}}_{\Gamma,\mathcal{A}}(1^\kappa)$ should check whether $\mathsf{Open}(ok, m^*, \sigma^*, \boldsymbol{reg}) = i^*$ (as in Definition 2.14).

The notion of insider non-frameability can be strengthened towards stronger coalitions of group members with corrupted issuers and/or openers as discussed below.

Full Non-Frameability

In Definition 2.28 we formalize the notion of full non-frameability of DA-schemes. This is done similarly to the earlier definitions, except that now $\mathcal{A}$ receives two keys ik and ok. In particular, the experiment ensures that successful framing attack should be performed against a group member i^* previously admitted through the $\mathsf{JoinU}(gpk,\cdot)$ oracle (check that $\boldsymbol{gsk}[i^*] = \varepsilon$)

and not corrupted thereafter. Also this definition can be adopted to schemes with explicit user PKI by granting $\mathcal{A}$ additional access to the $\mathsf{AddPKI}(\cdot,\cdot)$ oracle and prohibiting its use for the registration of i^*.

Definition 2.28 (Full Non-Frameability : DA) A group signature scheme with distributed authorities $\Gamma = (\mathsf{GKg}, (\mathsf{JoinM}, \mathsf{JoinU}), \mathsf{GSign}, \mathsf{GVrfy}, \mathsf{Open}, \mathsf{Judge})$ provides *full non-frameability* if for all PPT adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa) = \Pr\left[\mathsf{Expt}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa) = 1\right].$$

The associated F-NF-*experiment* $\mathsf{Expt}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa)$ is executed to produce $(gpk, ik, \boldsymbol{reg}, ok)$.

Attack Stage. Adversary $\mathcal{A}$ receives $(gpk, ik, ok, \boldsymbol{reg})$.

1. $\mathcal{A}$ can submit queries to the oracles $\mathsf{JoinU}(gpk,\cdot)$, $\mathsf{Corrupt}(\cdot)$, and $\mathsf{GSign}(\boldsymbol{gsk}[\cdot],\cdot)$.
2. $\mathcal{A}$ stops and eventually outputs a tuple $(m^*, \sigma^*, i^*, \tau^*)$.

Output. If all of the following holds then the experiment outputs 1:

1. $\mathsf{GVrfy}(gpk, m^*, \sigma^*) = 1$ and $i^* \in [1, n]$ and $\mathsf{Judge}(gpk, m^*, \sigma^*, i^*, \tau^*) = 1$
2. $\boldsymbol{gsk}[i^*] = \varepsilon$
3. $\mathcal{A}$ did not submit i^* to $\mathsf{Corrupt}(\cdot)$
4. $\mathcal{A}$ did not submit (i^*, m^*) to $\mathsf{GSign}(\boldsymbol{gsk}[\cdot],\cdot)$.

Otherwise it outputs 0. ◊

Remark 2.4.4 If the DA-scheme Γ does not have verifiable opening then output conditions of the experiment $\mathsf{Expt}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa)$ should check whether $\mathsf{Open}(ok, m^*, \sigma^*, \boldsymbol{reg}^*) = i^*$, assuming that $\mathcal{A}$ additionally outputs a registration list $\boldsymbol{reg}^*$ (as in Definition 2.15). Instead of requiring that $\mathcal{A}$ outputs $\boldsymbol{reg}^*$ we could also give $\mathcal{A}$ access to the write oracle $\mathsf{WReg}(\cdot,\cdot)$ and use the resulting registration list in the above check.

Finally, we observe that several intermediate flavors residing between insider and full non-frameability can be obtained by giving $\mathcal{A}$ partial access to the secret keys of both authorities. For example, $\mathcal{A}$ can be given as input the issuing key ik and write access to $\boldsymbol{reg}$, but not the opening key ok. In this case $\mathcal{A}$ would have to get the opening oracle but the output conditions of experiment $\mathsf{Expt}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa)$ would remain the same. Alternatively, $\mathcal{A}$ can be given the opening key ok and the read access to $\boldsymbol{reg}$, but not the issuing key ik. In this case one would remove the $\mathsf{JoinU}(gpk,\cdot)$ oracle and modify the second output condition according to the experiment for insider non-frameability from Definition 2.27.

3. Cryptographic Foundations and Hardness Assumptions

Foundations of modern cryptography include various *hardness assumptions* for proving security of cryptographic schemes and a broad spectrum of *cryptographic primitives* serving as building blocks for more advanced cryptographic constructs. In this chapter we introduce several hardness assumptions and provide an overview of primitives that will become relevant in our description of modern group signatures. We start with general assumptions that consider existence of certain types of abstractly defined functions and continue with the description of more concrete hardness assumptions based on number theory. Additionally, we will give an overview of several basic cryptographic building blocks and their security properties.

3.1. General Hardness Assumptions

We describe two general hardness assumptions — existence of one-way functions/permutations and existence of trapdoor permutations. These assumptions are foundational for many cryptographic primitives. As we will see the assumption on the existence of trapdoor permutations is important in the context of group signatures.

3.1.1. One-Way Functions

Let $f : \{0,1\}^* \mapsto \{0,1\}^*$ be some function. Intuitively, f is one-way if it is easy to compute but hard to invert. While "easy to compute" requires f to be computable in polynomial time, "hard to invert" assumes the absence of a polynomial-time algorithm for computing the pre-images of f. This intuition is formalized in Definition 3.1.

Definition 3.1 (One-Way Function/Permutation) A function $f : \{0,1\}^* \mapsto \{0,1\}^*$ is *one-way* if all of the following holds:

Easy to compute. There exists a polynomial-time algorithm that on input $x \in \{0,1\}^*$ computes $f(x)$.

Hard to invert. For all PPT algorithms $\mathcal{A}$ the following advantage function is negligible in κ:

$$\mathsf{Adv}^{\mathrm{OW}}_{f,\mathcal{A}}(\kappa) = \Pr\left[\begin{array}{c} x \in_R \{0,1\}^\kappa, y = f(x) \\ x^* \leftarrow \mathcal{A}(1^\kappa, y) \end{array} : f(x^*) = y\right].$$

Furthermore, if f is also length-preserving ($|f(x)| = |x|$ for all $x \in \{0,1\}^*$) and if its restriction to the domain $\{0,1\}^\kappa$ is a bijection then f is a *one-way permutation.* If f is a one-way permutation then any value y in its range uniquely determines the corresponding pre-image x. $\Diamond$

The existence of one-way functions is an unproven assumption. Nevertheless, cryptographic constructions often use computational problems like integer factorization or computation of discrete logarithms (cf. Section 3.2.2) as candidates for such functions due to the absence of (efficient) polynomial-time algorithms that can solve these problems for an appropriate choice of security parameters; despite of much attention that those problems have received in the past.

3.1.2. Trapdoor Permutations

Let $f : \{0,1\}^* \mapsto \{0,1\}^*$ be some function. Intuitively, f is a trapdoor permutation if f is a one-way permutation, for which there exists an associated (secret) trapdoor information td allowing to efficiently compute the pre-images of f. This intuition is formalized in Definition 3.2.

Definition 3.2 (Trapdoor Permutation) A function $f : \{0,1\}^* \mapsto \{0,1\}^*$ with an associated trapdoor information $td \in \{0,1\}^\kappa$, $\kappa \in \mathbb{N}$ is a *trapdoor permutation* if all of the following holds:

One-way permutation. If td is kept secret then f has the same properties as a one-way permutation from Definition 3.1.

Easy to invert with a trapdoor. There exists a PPT algorithm that for all $x \in \{0,1\}^*$ on input td and $y = f(x)$ outputs x. $\Diamond$

Note that the easy inversion with the trapdoor does not contradict the function's one-wayness, because the trapdoor is not part of the function's output and is therefore not part of $\mathcal{A}$'s input. Obviously, any trapdoor permutation is also a one-way permutation. On the other hand, not every one-way permutation can be associated with a trapdoor. Therefore, the assumption on the existence of trapdoor permutations is strictly stronger than on the existence of one-way functions. As we will see, existence of trapdoor permutations is likely to be the weakest assumption needed to construct secure group signatures, unlike ordinary signatures that can be constructed from one-way functions. Although existence of trapdoor permutations is an unproven assumption, a well-known candidate for a trapdoor permutation is the permutation used by the RSA cryptosystem (cf. Section 3.2.1).

3.2. Number-Theoretic Hardness Assumptions

While general assumptions are helpful to assess security of cryptographic schemes from the theoretical point of view many practical cryptographic constructions require assumptions based on number-theory. In this section we give an overview of three number-theoretic settings that have been used in the design of modern group signature schemes. These include the RSA setting, the DL setting, and the setting of bilinear maps, which can be seen as a special case of the DL setting with richer algebraic properties. In our description we will assume that the reader is familiar with basic number-theoretic concepts used in cryptography that can be found, for example, in the book of Shoup [171].

3.2.1. Assumptions in the RSA Setting

The RSA setting is based on an algorithm RSAGen that on input a security parameter 1^κ, $\kappa \in \mathbb{N}$ outputs a tuple of integers (N, p, q) such that $N = pq$ is of length κ, and p, q are prime numbers. N is called RSA modulus. Moreover, if p, q are safe primes, i.e. $p = 2p' + 1$ and $q = 2q' + 1$ with p' and q' being primes as well, then the RSA modulus N is called *safe*. It is widely assumed that factoring N, that is computing its prime factors p and q, is hard if these factors are sufficiently large. The RSA setting admits further hardness assumptions that we will use in our description of group signatures and introduce in the following.

Definition 3.3 (Strong RSA Assumption (SRSA)) Let RSAGen be an algorithm that outputs (N, p, q) with N being a (safe) RSA modulus and let $\mathbb{G} = \langle g \rangle$ denote a cyclic subgroup of $\mathbb{Z}_N^*$ of order Q with length $|Q| = \kappa$. The *Strong RSA (SRSA)* assumption says that for all PPT algorithms $\mathcal{A}$ the following advantage function is negligible in κ:

$$\mathsf{Adv}^{\mathrm{SRSA}}_{\mathsf{RSAGen},\mathcal{A}}(\kappa) = \Pr\left[\begin{array}{c}(N,p,q) \leftarrow \mathsf{RSAGen}(1^\kappa), z \in_R \mathbb{G}, \\ (u,e) \leftarrow \mathcal{A}(N,g,z)\end{array} : \begin{array}{c} u \in \mathbb{G}, e \in \mathbb{Z}_{>1} \\ u^e = z \pmod N\end{array}\right].$$

◊

A frequent choice for $\mathbb{G}$ in the context of group signatures is the group of *quadratic residues* modulo N, denoted $QR(N)$. This group of order $p'q'$ is generated by an element $g \in \mathbb{Z}_N$. An appropriate generator g can be chosen by picking $a \in \mathbb{Z}_N^*$ such that $gcd(a \pm 1, N) = 1$, in which case $g = a^2 \mod N$. Security of several group signatures, where the $QR(N)$ group is used, relies further on the following assumption.

Definition 3.4 (Decision Diffie-Hellman (DDH) Assumption in $QR(N)$) Let RSAGen be an algorithm that outputs (N, p, q) with N being a (safe) RSA modulus and let $QR(N) = \langle g \rangle$ denote the group of quadratic residues modulo N of order $p'q'$ of length κ. The *Decision Diffie-Hellman (DDH) assumption in* $QR(N)$ says that for all PPT algorithms $\mathcal{A}$ the following advantage function is negligible in κ:

$$\mathsf{Adv}^{\mathrm{DDH}}_{\mathsf{RSAGen},\mathcal{A}}(\kappa) = \left|\Pr\left[\begin{array}{c}(N,p,q) \leftarrow \mathsf{RSAGen}(1^\kappa), x, y, z \in_R \mathbb{Z}_{p'q'}, \\ g_0 = g^{xy}, g_1 = g^z, b \in_R \{0,1\}, b^* \leftarrow \mathcal{A}(N, g, g^x, g^y, g_b)\end{array} : b = b^*\right] - \frac{1}{2}\right|.$$

◊

The DDH assumption in $QR(N)$ groups is a special type of the more general DDH assumption, which can be defined over cyclic groups $\mathbb{G}$ of prime order (cf. Definition 3.6). Interestingly, in $QR(N)$ groups the DDH assumption is assumed to hold, irrespective of whether the factors p and q of the RSA modulus N are known or not.

3.2.2. Assumptions in the DL Setting

The Discrete Logarithm (DL) setting is based on an algorithm GenG that on input a security parameter 1^κ, $\kappa \in \mathbb{N}$ outputs the description of a cyclic group $(\mathbb{G}, g, Q)$ where g is the generator and the order Q is prime and of length κ. The DL setting admits the following hardness assumptions that we will refer to in the constructions of group signature schemes.

Definition 3.5 (Discrete Logarithm (DL) Assumption) Let GenG be an algorithm that outputs the description of a cyclic group $(\mathbb{G}, g, Q)$ with Q prime and $|Q| = \kappa$. The *Discrete Logarithm (DL)* assumption says that for all PPT algorithms $\mathcal{A}$ the following advantage function is negligible in κ:

$$\mathsf{Adv}^{\mathrm{DL}}_{\mathsf{GenG},\mathcal{A}}(\kappa) = \Pr\left[\begin{array}{c} (\mathbb{G}, g, Q) \leftarrow \mathsf{GenG}(1^\kappa), h \in_R \mathbb{G}, \\ x \leftarrow \mathcal{A}(\mathbb{G}, g, Q, h) \end{array} : \begin{array}{c} x \in \mathbb{Z}_Q \\ g^x = h \end{array}\right].$$

◇

A popular example of a suitable group, often used in cryptographic schemes, is a subgroup $\mathbb{G} \subset \mathbb{Z}_P$ of a prime order $Q|P-1$ where P is also prime.

Furthermore, the DDH assumption introduced for $QR(N)$ groups in the previous section (cf. Definition 3.4) can be generalized to cyclic groups of prime order. For example, IND-CPA security of the well-known ElGamal encryption scheme [88] relies on the DDH assumption.

Definition 3.6 (Decision Diffie-Hellman (DDH) Assumption in Groups of Prime Order) Let GenG be an algorithm that outputs the description of a cyclic group $(\mathbb{G}, g, Q)$ with Q prime and $|Q| = \kappa$. The *Decision Diffie-Hellman (DDH)* assumption in $\mathbb{G}$ says that for all PPT algorithms $\mathcal{A}$ the following advantage function is negligible in κ:

$$\mathsf{Adv}^{\mathrm{DDH}}_{\mathsf{GenG},\mathcal{A}}(\kappa) = \left|\Pr\left[\begin{array}{c} (\mathbb{G}, g, Q) \leftarrow \mathsf{GenG}(1^\kappa), x, y, z \in_R \mathbb{Z}_Q, \\ g_0 = g^{xy}, g_1 = g^z, b \in_R \{0,1\}, b^* \leftarrow \mathcal{A}(g, g^x, g^y, g_b) \end{array} : b = b^*\right] - \frac{1}{2}\right|.$$

◇

3.2.3. Assumptions in the Setting of Bilinear Maps

The cryptographic setting of bilinear maps, also known as *pairings* serves as a basis for many modern group signature schemes as it provides richer algebraic structure in comparison to the DL and RSA settings. The setting of bilinear maps is based on an algorithm GenBG that on input a security parameter 1^κ, $\kappa \in \mathbb{N}$ outputs the description of two cyclic groups $(\mathbb{G}_1, g_1, Q)$ and $(\mathbb{G}_2, g_2, Q)$ of prime order Q of length κ and respective generators g_1 and g_2 with an associated bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ where $\mathbb{G}_T$ is another cyclic group (called *target* group) of order Q. The corresponding groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are called bilinear if they satisfy the following definition.

Definition 3.7 (Bilinear Groups) Let GenBG be an algorithm that outputs the description of two cyclic groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ of prime order Q with $|Q| = \kappa$, where possibly

$\mathbb{G}_1 = \mathbb{G}_2$, and the description of $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ with $\mathbb{G}_T$ being another cyclic group of prime order Q. The group pair $(\mathbb{G}_1, \mathbb{G}_2)$ is called *bilinear* if the following holds:

1. Efficiency: The *bilinear map* $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ can be computed in polynomial-time.
2. Bilinearity: For all $u \in \mathbb{G}_1, v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_Q$: $e(u^a, v^b) = e(u, v)^{ab}$.
3. Non-degeneracy: $e(g_1, g_2) = 1$.

There can further exist an efficiently computable homomorphism ψ from $\mathbb{G}_2$ to $\mathbb{G}_1$ with $\psi(g_2) = g_1$. ◊

Depending on the choice of the input groups $(\mathbb{G}_1, \mathbb{G}_2)$ and existence of the homomorphism ψ, the associated pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ can be classified as follows (see also Galbraith, Paterson, and Smart [95]).

Definition 3.8 (Bilinear Maps: Classification) Let $(\mathbb{G}_1, \mathbb{G}_2)$ be bilinear groups and $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ the associated bilinear map according to Definition 3.7.

Type-1. e is of Type-1 if $\mathbb{G}_1 = \mathbb{G}_2$.

Type-2. e is of Type-2 if $\mathbb{G}_1 = \mathbb{G}_2$ and there exists an efficiently computable homomorphism $\psi : \mathbb{G}_2 \mapsto \mathbb{G}_1$.

Type-3. e is of Type-3 if $\mathbb{G}_1 = \mathbb{G}_2$ and there exists no efficiently computable homomorphism $\psi : \mathbb{G}_2 \mapsto \mathbb{G}_1$.

◊

In general, different pairing types may admit different hardness assumptions and result in more or less efficient implementations. For example, the DDH assumption does not hold in Type-1 pairings. Indeed, if $\mathbb{G}_1 = \mathbb{G}_2$ and the corresponding generator is g then for a given problem instance (g, g^x, g^y, g_b) one can easily distinguish the corresponding bit b by testing whether $e(g^x, g^y) = e(g_b, g)$. In Type-2 pairings the DDH assumption is assumed to hold only in the input group $\mathbb{G}_1$, whereas in Type-3 pairings the DDH assumption is assumed to hold in both input groups $\mathbb{G}_1$ and $\mathbb{G}_2$. From the efficiency point of view, Type-3 pairings admit the most efficient implementations, considering both bandwidth and computation costs.

In the following we focus on some number-theoretic hardness assumptions, frequently used in the design of group signatures. We start with the q-Strong Diffie-Hellman (q-SDH) assumption, which was introduced by Boneh and Boyen [35]. It is one of the most popular assumptions used to prove security of various signature and group signature schemes.

Definition 3.9 (q-Strong Diffie-Hellman (q-SDH) Assumption) Let GenBG be an algorithm that outputs a pair of bilinear groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ of prime order Q with $|Q| = \kappa$, and the associated bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$. The *q-Strong Diffie-Hellman (q-SDH)* assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ with $q \in \mathbb{N}$ says that for all PPT algorithms $\mathcal{A}$ the following advantage function is negligible in κ:

$$\mathsf{Adv}^{\mathrm{q-SDH}}_{\mathsf{GenBG},\mathcal{A}}(\kappa) = \Pr \begin{array}{c} \gamma \in_R \mathbb{Z}_Q \\ (g, x) \leftarrow \mathcal{A}(g_1, g_2, g_2^{\gamma}, g_2^{(\gamma^2)}, \ldots, g_2^{(\gamma^q)}) \end{array} : x \in \mathbb{Z}_Q^*, g = g_1^{\frac{1}{\gamma + x}} \quad .$$

◊

The following Decision Linear (DLIN) assumption was introduced by Boneh, Boyen, and Shacham [36]. It serves as a basis of the Linear Encryption scheme, which can be seen as an analog of the ElGamal encryption scheme in bilinear groups, most frequently in Type-1 pairings, where the original DDH problem is not necessarily hard. We define the DLIN assumption in the setting of bilinear maps, where it is traditionally used. However, we note that DLIN assumption can also be formulated in the standard DL setting using the corresponding algorithm GenG.

Definition 3.10 (Decision Linear (DLIN) Assumption) Let $\mathbb{G} = \langle g \rangle$ be one of the bilinear groups of prime order Q with $|Q| = \kappa$ output by an algorithm GenBG, and u, v, and h be arbitrary generators of $\mathbb{G}$. The *Decision Linear (DLIN)* assumption in $\mathbb{G}$ says that for all PPT algorithms $\mathcal{A}$ the following advantage function is negligible in κ:

$$\mathsf{Adv}^{\mathrm{DLIN}}_{\mathsf{GenBG},\mathcal{A}}(\kappa) = \left| \Pr\left[\begin{array}{c} u, v, h \in_R \mathbb{G},\ \alpha, \beta, \gamma \in_R \mathbb{Z}_Q,\ h_0 = h^{\alpha+\beta}, h_1 = h^{\gamma} \\ b \in_R \{0,1\}, b^* \leftarrow \mathcal{A}(u, v, h, u^{\alpha}, v^{\beta}, h_b) \end{array} : b = b^* \right] - \frac{1}{2} \right| .$$

◊

It can be shown that an algorithm breaking the DLIN Assumption in $\mathbb{G}$ can be used to solve the DDH problem in $\mathbb{G}$ while the converse is believed to be false.

The following Decision Bilinear Diffie-Hellman (DBDH) assumption was introduced by Boneh and Boyen [34].

Definition 3.11 (Decision Bilinear Diffie-Hellman (DBDH) Assumption) Let $\mathbb{G} = \langle g \rangle$ be one of the bilinear groups of prime order Q with $|Q| = \kappa$ output by an algorithm GenBG. The *Decision Bilinear Diffie-Hellman (DBDH)* assumption in $\mathbb{G}$ says that for all PPT algorithms $\mathcal{A}$ the following advantage function is negligible in κ:

$$\mathsf{Adv}^{\mathrm{DBDH}}_{\mathsf{GenBG},\mathcal{A}}(\kappa) = \left| \Pr\left[\begin{array}{c} a, b, c, d \in_R \mathbb{Z}_Q,\ h_0 = e(g,g)^{abc}, h_1 = e(g,g)^{d} \\ b \in_R \{0,1\}, b^* \leftarrow \mathcal{A}(g, g^a, g^b, g^c, h_b) \end{array} : b = b^* \right] - \frac{1}{2} \right| .$$

◊

The following LRSW assumption was introduced by Lysyanskaya et al. [135]. The LRSW assumption is defined for general prime order groups $\mathbb{G}$, however, we will use it mostly in the context of bilinear groups with $\mathbb{G}$ being one of the input groups (i.e. $\mathbb{G}_1$ or $\mathbb{G}_2$). Additionally, we mention the asymmetric version of the LRSW assumption, which explicitly takes inputs from both input groups $\mathbb{G}_1$ and $\mathbb{G}_2$.

Definition 3.12 (LRSW Assumption) Let $\mathbb{G} = \langle g \rangle$ be a group of prime order Q with $|Q| = \kappa$ and $X, Y \in \mathbb{G}$ with $X = g^x$ and $Y = g^y$. Let $O_{X,Y}(\cdot)$ be an oracle that, on input a value $m \in \mathbb{Z}_Q$, outputs a triple (a, a^y, a^{x+mxy}) for a randomly chosen $a \in \mathbb{G}$.

The *LRSW* assumption in $\mathbb{G}$ says that for all PPT algorithms $\mathcal{A}$ the following advantage function (where $\mathcal{Q}$ is the set of queries $\mathcal{A}$ poses to $O_{X,Y}(\cdot)$) is negligible in κ:

$$\mathsf{Adv}^{\mathrm{LRSW}}_{\mathcal{A}}(\kappa) = \Pr\left[\begin{array}{c} x \in_R \mathbb{Z}_Q, y \in_R \mathbb{Z}_Q, X = g^x, Y = g^y \\ (m, a, b, c) \leftarrow \mathcal{A}^{O_{X,Y}(\cdot)}(Q, \mathbb{G}, g, X, Y) \end{array} : \begin{array}{c} m \notin \mathcal{Q}, m \in \mathbb{Z}_Q, m = 0 \\ a \in \mathbb{G}, b = a^y, c = a^{x+mxy} \end{array} \right] .$$

The *asymmetric* version of the LRSW assumption employs two different groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$, both of prime order Q with $|Q| = \kappa$, such that $a \in \mathbb{G}_1$, whereas $X, Y \in \mathbb{G}_2$. $\diamondsuit$

The following SDLP assumption was introduced by Bichsel et al. [31].

Definition 3.13 (SDLP Assumption) Let $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$ be bilinear groups of prime order Q with $|Q| = \kappa$ with the associated bilinear map e.

The *SDLP* assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ says that for all PPT algorithms $\mathcal{A}$ the following advantage function is negligible in κ:

$$\mathsf{Adv}^{\mathrm{SDLP}}_{\mathcal{A}}(\kappa) = \Pr \quad \mu \in_R \mathbb{Z}_Q, \mu^* \leftarrow \mathcal{A}(Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, g_1^{\mu}, g_2^{\mu}) \ : \ \mu = \mu^* \quad .$$

$\diamondsuit$

3.3. Hash Functions

Cryptographic hash functions are functions that map binary strings of arbitrary length to a string of a constant length with the additional property of collision-resistance, meaning that it is difficult to find two distinct input strings that would result in the same output. Traditionally, cryptographic hash functions are defined in terms of families of functions where some particular function from the family is selected by specifying its public index.

Definition 3.14 (Collision-Resistant Hash Function) A family of *hash functions* $\{\mathsf{Hash}_k : \{0,1\}^* \to \{0,1\}^{\ell(k)} \mid k \leftarrow \mathsf{Kg}(1^\kappa)\}$ is called *collision-resistant* if the following advantage function is negligible in κ:

$$\mathsf{Adv}^{\mathrm{COL}}_{\mathsf{Hash},\mathcal{A}}(\kappa) = \Pr \quad (x, y) \leftarrow \mathcal{A}(k) \ : \ x = y \ \text{ and } \ \mathsf{Hash}_k(x) = \mathsf{Hash}_k(y) \quad .$$

Using notation $\mathsf{Hash} : \{0,1\}^* \to \{0,1\}^\kappa$ we implicitly assume that the length of the output corresponds to the security parameter 1^κ of the function. $\diamondsuit$

In the context of digital signatures that we discuss in Section 3.4 and also group signatures, as we will see in later sections, cryptographic hash functions are typically used in the so-called "hash-then-sign" approach, where a hash function Hash is first applied to some message m of an arbitrary length to obtain the constant-length hash value $\mathsf{Hash}(m)$, which is then processed by the signing algorithm of the scheme.

3.3.1. The Random Oracle Model

Many cryptographic constructions, including various group signature schemes, that utilize a hash function $\mathsf{Hash} : \{0,1\}^* \mapsto \{0,1\}^\kappa$ as one of their building blocks can be proven secure only in the so-called *Random Oracle Model (ROM)* introduced by Bellare and Rogaway [24]. This model makes a non-standard assumption on the computation and distribution of hash values. In particular, it assumes that computation of $\mathsf{Hash}(m)$ cannot be performed by an algorithm on its own but requires assistance of some "magic" party, called *random oracle*, that on input m outputs $\mathsf{Hash}(m)$, and that these outputs are uniformly distributed in $\{0,1\}^*$ while preserving the deterministic nature of the hash function, in that any two different invocations of the random oracle on the same input result in the same output. This setting essentially assumes that Hash is the random oracle. Obviously, this assumption is very strong since given the description of some (practical) hash function Hash one could easily compute $\mathsf{Hash}(m)$ on any given input m. Therefore, ROM is primarily seen as methodology for proving security of cryptographic schemes, that is in the security proof hash function Hash used in the construction is replaced with the random oracle. In particular, this means that the adversary against some security property of the scheme can neither compute $\mathsf{Hash}(m)$ without querying the random oracle Hash nor derive any information about m when given only $\mathsf{Hash}(m)$. A more detailed information about the ROM methodology can also be found in the book by Katz and Lindell [116]. It should, furthermore, be noted that not every cryptographic scheme that uses hash functions automatically requires ROM. Security proofs, in which Hash need not be a random oracle but a collision-resistant hash function are typically referred to as being performed in the *standard* model. On the other hand, using ROM to prove security of a cryptographic scheme is still better than claiming security of the scheme without any proof.

3.4. Digital Signatures

Digital signatures can be used for the purpose of authentication and identification. These cryptographic primitives are frequently used as building blocks in cryptographic protocols, including group signatures. In the following we provide basic definitions of digital signature schemes and their security. For a more detailed treatment of many known digital signature schemes and their security properties we refer to the book of Katz [115].

Definition 3.15 (Digital Signature) A *digital signature scheme* $\Sigma = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vrfy})$ consists of three polynomial-time algorithms:

Key generation. The randomized *key generation algorithm* Kg takes as input a security parameter 1^κ, $\kappa \in \mathbb{N}$, and outputs a private/public key pair (sk, pk).

Signature generation. The *signature generation algorithm* Sign takes as input a secret key sk and a message $m \in \{0,1\}^*$, and outputs signature σ.

Verification procedure. The deterministic *verification algorithm* Vrfy takes as input a public key pk, a message $m \in \{0,1\}^*$, and a candidate signature σ and outputs 1 (indicating that σ is valid) or 0.

A signature scheme Σ is *correct* if for all $\kappa \in \mathbb{N}$, all $(sk, pk) \leftarrow \mathsf{Kg}(1^\kappa)$, and all messages $m \in \{0,1\}^*$:

$$\mathsf{Vrfy}(pk, m, \mathsf{Sign}(sk, m)) = 1.$$

A digital signature scheme must satisfy at least the following security requirement of (existential) unforgeability, which is modeled through the experiment, in which a PPT adversary $\mathcal{A}$ is allowed to obtain valid signatures on messages of its choice (through the signing oracle $\mathsf{Sign}(sk, \cdot)$) and its goal is to come up with a valid σ^* on some message m^*, possibly of $\mathcal{A}$'s choice, that has not been signed before.

Definition 3.16 (Unforgeability of Signatures) A digital signature scheme $\Sigma = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vrfy})$ provides *unforgeability* if for all PPT adversaries $\mathcal{A}$ the following advantage function is negligible in κ:

$$\mathsf{Adv}^{\text{UNF}}_{\Sigma,\mathcal{A}}(\kappa) = \Pr\left[\begin{array}{c} (sk, pk) \leftarrow \mathsf{Kg}(1^\kappa), \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(pk) \end{array} : \begin{array}{c} \mathsf{Vrfy}(pk, m^*, \sigma^*) = 1 \\ \mathcal{A} \text{ did not submit } m^* \text{ to } \mathsf{Sign}(sk,\cdot) \end{array}\right].$$

$\Diamond$

As already noticed, signing of arbitrarily long messages can be realized using the "hash-then-sign" approach. The collision-resistance property of the hash function contributes in this case to the unforgeability requirement of the scheme.

3.5. Public-Key Encryption

In order to allow confidential communication and exchange of data cryptography offers a variety of encryption schemes. In the context of group signatures of prime interest are public key encryption schemes.

Definition 3.17 (Public Key Encryption (PKE)) A *public key encryption (PKE) scheme* $\Theta = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ consists of three polynomial-time algorithms:

Key generation. The randomized *key generation algorithm* Kg takes as input a security parameter 1^κ, $\kappa \in \mathbb{N}$, and outputs a private/public key pair (sk, pk).

Encryption procedure. The randomized *encryption algorithm* Enc takes as input a public key pk and a message $m \in \{0,1\}^*$, and outputs ciphertext c.

Decryption procedure. The deterministic *decryption algorithm* Dec takes as input a private key sk and a ciphertext c and outputs either m (indicating that m could be correctly decrypted) or $\perp$ (if some failure occurred).

A PKE scheme Θ is *correct* if for all $\kappa \in \mathbb{N}$, all $(sk, pk) \leftarrow \mathsf{Kg}(1^\kappa)$, and all messages $m \in \{0,1\}^*$:

$$\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m.$$

A PKE scheme Θ typically satisfies one of the following notions of security, which prevents information leakage about the encrypted message m. Both notions can only be achieved if the encryption algorithm is randomized.

Definition 3.18 (IND-CCA/IND-CPA Security) A public key encryption scheme $\Theta = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ provides *indistinguishability against chosen ciphertext attacks* (IND-CCA) if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ the following advantage function is negligible in κ:

$$\mathsf{Adv}^{\text{IND-CCA}}_{\Theta,\mathcal{A}}(\kappa) = \Pr\left[\begin{array}{c} (sk, pk) \leftarrow \mathsf{Kg}(1^\kappa), \\ (st, m_0, m_1) \leftarrow \mathcal{A}_1^{\mathsf{Dec}(sk,\cdot)}(pk) \\ b \in_R \{0,1\}, c \leftarrow \mathsf{Enc}(pk, m_b) \\ b^* \leftarrow \mathcal{A}_2^{\mathsf{Dec}(sk,\cdot)}(st, c) \end{array} : \begin{array}{c} b^* = b \\ \mathcal{A} \text{ did not submit } c \text{ to } \mathsf{Dec}(sk,\cdot) \end{array}\right] - \frac{1}{2}.$$

If in the above experiment $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is not given access to the decryption oracle $\mathsf{Dec}(sk, \cdot)$ and the corresponding advantage function $\mathsf{Adv}^{\text{IND-CPA}}_{\Theta,\mathcal{A}}(\kappa)$ remains negligible in κ then Θ is said to provide *indistinguishability against chosen plaintext attacks* (IND-CPA).

There also exists a weaker notion of IND-CCA security often called IND-CCA1 (with the above definition being IND-CCA2), where only $\mathcal{A}_1$ is given access to the decryption oracle $\mathsf{Dec}(sk, \cdot)$. $\Diamond$

3.6. Commitment Schemes

Cryptographic commitment schemes, defined in the following, are useful building blocks for applications where a party should commit to some value without actually revealing information about this value and then at some later stage disclose this value without being able to modify it. A typical example for the use of commitment schemes can be found in auctions where during the bidding phase competitors submit commitments on their bids and later disclose their bids in order to estimate the winner.

Definition 3.19 ((Non-Interactive) Commitment Schemes) A (non-interactive) *commitment scheme* $\Xi = (\mathsf{Kg}, \mathsf{Commit}, \mathsf{Vrfy})$ consists of three polynomial-time algorithms defined as follows:

Key generation. The randomized *key generation algorithm* Kg takes as input a security parameter 1^κ,$\kappa \in \mathbb{N}$, and outputs the *commitment public key* ck. It is implicitly assumed that ck determines the message space $\mathcal{M}$ and a randomizer space $\mathcal{R}$.

Commitment generation. The randomized *commitment algorithm* Commit takes as input ck, message $m \in \mathcal{M}$, and randomizer $r \in_R \mathcal{R}$, and outputs a *commitment* c.

Verification procedure. The randomized *verification algorithm* Vrfy takes as input ck, $m \in \mathcal{M}$, $r \in \mathcal{R}$, and a candidate commitment c, and outputs either 1 (indicating that commitment c is valid for the message m) or 0.

A commitment scheme Ξ is *correct* if for all $\kappa \in \mathbb{N}$, all $ck \leftarrow \mathsf{Kg}(1^\kappa)$, all messages $m \in \mathcal{M}$, all randomizers $r \in \mathcal{R}$ and $c \leftarrow \mathsf{Commit}(m, r)$: $\mathsf{Vrfy}(ck, m, r, c) = 1$.

$\mathsf{Commit}(m, r)$ will denote the output of the commitment algorithm with ck as an implicit input. Sometimes we will write $c = \mathsf{Commit}(m, r)$ instead of $\mathsf{Vrfy}(c, m, r)$ assuming that c can be recomputed from the *opening* (m, r). $\Diamond$

There are two main security properties that should be satisfied by any commitments scheme: (1) *hiding* which protects leakage of information about the committed message m from the corresponding commitment c, and (2) *binding* which ensures that some particular commitment c can only be opened to the originally committed message m.

A simple example of a commitment scheme is due to Pedersen [159] that uses some cyclic group $\mathbb{G} = \langle g \rangle$ of prime order Q, possibly in the DL setting or in $QR(N)$ groups with safe RSA modulus N. The public commitment key is $ck = (g, h)$ for some $h \in_R \mathbb{G}$ for which the discrete logarithm $\log_g h$ remains unknown. In order to commit to some message $m \in \mathbb{Z}_Q$ the algorithm chooses a randomizer $r \in_R \mathbb{Z}_Q$ and outputs the commitment $c = g^m h^r$. To open this commitment c the algorithm outputs the initially used message-randomizer pair (m, r), allowing the verification algorithm to recompute c and compare it with the initially received commitment.

In group signature schemes commitments are widely used during the signature generation procedure and in combination with (non-interactive) zero-knowledge proofs and signatures of knowledge which we briefly introduce in the next section.

3.7. Zero-Knowledge Proofs and Signatures of Knowledge

Let R be a polynomial time decidable binary relation for pairs $(C, w) \in R$ consisting of a statement C and a witness w. Let L_R be the NP-language of statements C that have witnesses w of length κ in R, i.e. $L_R = \{C \mid \exists w, |w| = \kappa \text{ and } (C, w) \in R\}$. Statements may have several witnesses, i.e. there might exist a set of witnesses $w(C)$ such that $(C, w) \in R$ for all $w \in w(C)$.

3.7.1. Zero-Knowledge Proofs of Knowledge (ZKPoK)

The main goal behind zero-knowledge proofs of knowledge (ZKPoK) is to allow some prover P being in possession of some secret w to prove to some verifier V that P knows the secret and that this secret satisfies some relation. An illustrative example is where a prover is in possession of some private/public key pair (sk, pk) and wishes to prove the knowledge of sk to a verifier V who knows pk. Clearly, a ZKPoK proof for this task should not reveal sk nor should it put any third party (an adversary) in position where it can produce convincing proof without knowing sk. We proceed with the notations and definitions of ZKPoK proofs.

ZKPoK protocols are defined between two PPT algorithms P and V, in which P tries to convince V to accept the proof. ZKPoK protocols should satisfy several properties as defined in Definition 3.20. The *correctness* property means that V accepts proofs for valid statements. The *soundness* property prevents P from cheating, which is the case if $C \notin L_R$. The *proof of knowledge (PoK)* property requires that if V accepts the proof then P must have known the corresponding witness. The *zero-knowledge (ZK)* property roughly means that the proof does not leak any information to the verifier, i.e. everything a verifier can compute through an interaction with the honest prover, can also be computed from the same inputs without this interaction.

Definition 3.20 (Zero-Knowledge Proof of Knowledge) Let L_R be an NP-language and R the corresponding binary relation. A pair of interactive PPT algorithms (P, V) is called *zero-knowledge proof of knowledge (ZKPoK)* if it satisfies the following properties:

Completeness. For all $C \in L_R$ and $w \in w(C)$, the probability that the verifier $V(C)$ accepts in its interaction with the prover $P(C, w)$ is 1.

Soundness. For all $C \notin L_R$ the probability that the verifier $V(C)$ accepts in its interaction with a PPT prover P^* is negligible (in κ).

PoK property. For all $C \in L_R$, if the verifier $V(C)$ accepts the proof in its interaction with a PPT prover P^* then there exists a PPT *extractor* E that on input C interacts with P^*, and outputs $w' \in w(C)$ with probability being non-negligible (in κ).

ZK property. For all $(C, w) \in R$ and any PPT verifier V^* interacting with the prover $P(C, w)$, there exists a PPT *simulator* S that on input C produces outputs that are indistinguishable from the outputs of V^*.
The ZK property is called: (i) *perfect* if the probability of distinguishing the above outputs is 0 even for an unbounded distinguisher; (ii) *statistical* if this probability is $\epsilon > 0$ even for an unbounded distinguisher, and (iii) *computational* if this probability is negligible (in κ) for a probabilistic polynomial-time distinguisher.

Since relations R and statement C can vary depending on the actual ZKPoK protocol we will be using the following notation for ZKPoK proofs on input a secret witness w testifying that $(C, w) \in R$ for some public statement C and relation R:

$$\mathsf{ZKPoK}\ \big(\ \text{witness } w\ \big)\ :\quad \text{relation}\quad R\quad \text{that}\quad (C, w)\quad \text{must satisfy}$$

For example, $\mathsf{ZKPoK}\ \big(\ x\ \big)\ :\ \ y = g^x$ denotes a zero-knowledge proof that the prover knows the discrete logarithm of y with respect to the base g. In this case g and y are public values, seen as part of the statement C, whereas x corresponds to the private witness w, R is given through the equality relation.

Although ZKPoK protocols can be constructed for all NP languages L_R, practical group signature schemes usually require proofs for languages that are believed to be in NP, i.e. where the intractability of a witness w from a statement $C \in L_R$ relies on some number-theoretic assumption, like the above proof for the equality of discrete logarithms. ZKPoK protocols for such languages typically require three protocol messages and are called Σ-protocols: The first message CMT (commitment) is computed by P on input (C, r) where r is some random string. The second message CH (challenge) is chosen by V in an unpredictable way and independent of CMT. The final message RSP (response) is computed by P on input (w, r, CMT, CH). At the end of the interaction V, given (C, CMT, CH, RSP), can verify the validity of the proof. In order to obtain perfect ZK property against malicious verifiers V^*, the challenge CH is usually set to be a random bit b. Indeed, this allows for the construction of a simulator S for the ZK property that can generate indistinguishable protocol transcripts with probability $\frac{1}{2}$. In this case, however, the soundness property holds also with probability $\frac{1}{2}$ since this is the probability with which a malicious prover P^* can guess CH. Therefore, in order to preserve soundness such protocol has to be executed between P and V sequentially κ-times, thus decreasing a successful attack of malicious provers P^* against the soundness property to $2^{-\kappa}$. Executing protocols κ-times is rather undesirable in practice. Therefore, challenge CH is typically chosen to be a random string from $\{0, 1\}^\kappa$. This change results, however, in a simulation problem for the ZK property. The reason is that a malicious verifier V^* might choose CH not truly at random but according to some distribution, which is not known to S. Therefore, Σ protocols satisfy a weaker ZK property, termed *honest-verifier* ZK, where the ZK property holds only against verifiers V that choose their challenges truly at random (as required by the protocol). Such honest-verifier ZKPoK proofs are still sufficient for many applications, including group signatures.

As an illustrative example we consider the above mentioned proof ZKPoK $x \quad : \quad y = g^x$, where we assume that g, y are elements of a cyclic group $\mathbb{G} = \langle g \rangle$ of prime order Q. The (honest-verifier) ZKPoK protocol runs as follows: Prover P picks random $r_x \in_R \mathbb{Z}_Q$ and sends commitment $CMT = g^{r_x}$ to the verifier V, which in turn replies with a random challenge $CH \in_R \mathbb{Z}_Q$ (of length κ). The prover P then computes its response as $RSP = r_x - CH \cdot x \bmod Q$, allowing V to check its validity through the comparison $CMT = g^{RSP}y^{CH}$.

3.7.2. Non-Interactive Zero-Knowledge Proofs of Knowledge (NIZKPoK)

Non-interactive zero-knowledge proofs of knowledge (NIZKPoK) are protocols where the prover P can compute the proof without involvement of the verifier V. That is, V receives a statement-proof pair (C, π) from P and should be able to check its validity without further communication with the prover. NIZKPoK proofs are formulated in the *common reference string (CRS)* model, which assumes that both algorithms P and V have access to some universal random string ρ. Therefore, NIZKPoK protocols have an additional generation algorithm K which on input some security parameter 1^κ, $\kappa \in \mathbb{N}$ outputs ρ. In Definition 3.21 we state main requirements on a NIZKPoK proof. These are similar to those of interactive ZKPoK proofs, except for some modifications regarding the CRS string.

Definition 3.21 (Non-Interactive ZKPoK) Let L_R be an NP-language and R the corresponding binary relation. A tuple of PPT algorithms (K, P, V) is called *non-interactive zero-knowledge proof of knowledge (NIZKPoK)* if it satisfies the following properties:

Completeness. For all CRS $\rho \leftarrow_R K(1^\kappa)$, all statement-witness pairs $(C, w) \in R$, and all proofs $\pi \leftarrow_R P(\rho, C, w)$, the probability that $V(\rho, C, \pi)$ accepts is 1.

Soundness. For all CRS $\rho \leftarrow_R K(1^\kappa)$ and all statement-candidate proofs $(C, \pi^*) \leftarrow_R P^*(\rho)$ with $C \notin L_R$, the probability that $V(\rho, C, \pi^*)$ accepts is negligible (in κ).

PoK property. There exists a PPT *extractor* $E = (E_1, E_2)$ with the following two properties:

a) Algorithm E_1 on input 1^κ generates pairs (ρ, st) such that for all PPT provers P^*, the probability with which P^* distinguishes, whether ρ was computed by $E_1(1^\kappa)$ or by $K(1^\kappa)$ is negligible (in κ).

b) For all outputs (ρ, st) generated by $E_1(\kappa)$, all PPT provers P^* that on input ρ output a pair (C, π^*) with $C \in L_R$, algorithm E_2 on input (ρ, st, C, π^*) outputs $w \in w(C)$ with probability being non-negligible (in κ).

ZK property. There exists a PPT *simulator* $S = (S_1, S_2)$ with the following properties:

a) Algorithm S_1 on input 1^κ generates pairs (ρ, st) such that for all PPT verifiers V^*, the probability with which V^* distinguishes, whether ρ was computed by $S_1(1^\kappa)$ or by $K(1^\kappa)$ is negligible (in κ).

b) For all (ρ, st) generated by $S_1(\kappa)$, all PPT verifiers V^*, the probability that V^* with input ρ, for any statement-witness pairs $(C, w) \in R$ distinguishes between outputs of the prover $P(\rho, C, w)$ and outputs of the algorithm $S_2(\rho, st, C)$ is negligible (in κ).

Also NIZKPoK proofs exist for all NP languages L_R. However, we will be mainly concerned with proofs that can be obtained from interactive Σ-protocols mentioned above. Indeed, any Σ-protocol can be converted to a corresponding NIZKPoK proof using the Fiat-Shamir transformation [89]. This transformation uses a collision-resistant hash function $\mathsf{Hash} : \{0,1\}^* \mapsto \{0,1\}^\kappa$ to compute the challenge message CH out of the statement C and commitment CMT. This prevents a malicious prover P^* from being able to modify the commitment CMT after obtaining the challenge CH, which is exactly the property ensured in Σ-protocols through interaction. The security of NIZKPoK proofs obtained through Fiat-Shamir transformation requires, however, an additional assumption, namely that the hash function Hash behaves like a random oracle (cf. Section 3.3.1). Note that any NIZKPoK protocol obtained in this way has thus its interactive counterpart, which is an honest-verifier ZKPoK, whose security does not necessarily require random oracles.

Similarly to ZKPoK proofs, we will use the following notation while referring to non-interactive zero-knowledge proofs of knowledge for some $(C, w) \in R$:

$$\mathsf{NIZKPoK}\quad \text{witness } w \quad : \quad \text{relation} \quad R \quad \text{that} \quad (C, w) \quad \text{must satisfy}$$

We implicitly assume that such NIZKPoK proof contains all information about C and R that is necessary to check its validity.

Examples of NIZKPoK Proofs

In the following we list several examples of NIZKPoK proofs to give an impression of how different types of relations can be handled in practice. We assume that $\mathbb{G} = \langle g \rangle$ is a cyclic group, generated by g and that the order of $\mathbb{G}$ is of length $\kappa_1 \in \mathbb{N}$. By $\epsilon > 1$ we denote another security parameter, that is used to control the so-called *tightness* of the statistical zero-knowledge property of these protocols. As mentioned before, NIZKPoK proofs obtained from the use of Fiat-Shamir transformation [89] use a hash function $\mathsf{Hash} : \{0,1\}^* \mapsto \{0,1\}^{\kappa_2}$, which must be modeled as a random oracle.

Definition 3.22 (NIZKPoK of a Discrete Logarithm) Let $g, y \in \mathbb{G}$. A pair $(c, s) \in \{0,1\}^{\kappa_2} \times \pm\{0,1\}^{\epsilon(\kappa_1+\kappa_2)+1}$ satisfying $c = \mathsf{Hash}(g, y, g^s y^c)$ is a NIZKPoK proof for the knowledge of the discrete logarithm of y with respect to the base g, denoted

$$\mathsf{NIZKPoK}\quad x \quad : \quad y = g^x \quad .$$

This proof can be computed by choosing a random $r_x \in_R \pm\{0,1\}^{\epsilon(\kappa_1+\kappa_2)}$ and then computing

$$c = \mathsf{Hash}(g, y, g^{r_x}) \quad \text{and} \quad s = r_x - cx.$$

◊

Definition 3.23 (NIZKPoK of a Representation) Let $g_1, \dots, g_n, y \in \mathbb{G}$. A tuple $(c, s_1, \dots, s_n) \in \{0,1\}^{\kappa_2} \times (\pm\{0,1\}^{\epsilon(\kappa_1+\kappa_2)+1})^n$ satisfying $c = \mathsf{Hash}(g_1, \dots, g_n, y, g_1^{s_1} \cdots g_n^{s_n} \cdot y^c)$ is a NIZKPoK proof for the the discrete logarithm-based representation of y_1 with respect to the bases $g_1, \dots, g_n$, denoted

$$\mathsf{NIZKPoK} \quad x_1, \dots, x_n \quad : \quad y = g_1^{x_1} \cdots g_n^{x_n} \quad .$$

This proof can be computed by choosing random $r_{x_i} \in_R \pm\{0,1\}^{\epsilon(\kappa_1+\kappa_2)}$ for all $i = 1, \dots, n$, and then producing

$$c = \mathsf{Hash}(g_1, \dots, g_n, y, g_1^{r_{x_1}} \cdots g_n^{r_{x_n}}) \ \text{ and } \ s_i = r_{x_i} - cx_i.$$

$\Diamond$

Definition 3.24 (NIZKPoK of Equality of Discrete Logarithms) Let $g_1, g_2, y_1, y_2 \in \mathbb{G}$. A tuple $(c, s) \in \{0,1\}^{\kappa_2} \times \pm\{0,1\}^{\epsilon(\kappa_1+\kappa_2)+1}$ satisfying $c = \mathsf{Hash}(g_1, g_2, y_1, y_2, g_1^s y_1^c, g_2^s y_2^c)$ is a NIZKPoK proof for the equality of the discrete logarithm of y_1 with respect to the base g_1 and of the discrete logarithm of y_2 with respect to the base g_2, denoted

$$\mathsf{NIZKPoK} \quad x \quad : \quad y_1 = g_1^x \ \text{ and } \ y_2 = g_2^x \quad .$$

This proof can be computed by choosing random $r_x \in_R \pm\{0,1\}^{\epsilon(\kappa_1+\kappa_2)}$ and then producing

$$c = \mathsf{Hash}(g_1, g_2, y_1, y_2, g_1^{r_x}, g_2^{r_{x_n}}) \ \text{ and } \ s = r_x - cx.$$

$\Diamond$

Definition 3.25 (NIZKPoK of a Discrete Logarithm in an Interval) Let $g, y \in \mathbb{G}$. A pair $(c, s) \in \{0,1\}^{\kappa_2} \times \pm\{0,1\}^{\epsilon(\kappa_1+\kappa_2)+1}$ satisfying $c = \mathsf{Hash}(g, y, g^{s-cX} y^c)$ is a NIZKPoK proof for the knowledge of the discrete logarithm of y with respect to the base g that lies in an interval $]X - 2^{\epsilon(\kappa_1+\kappa_2)}, X + 2^{\epsilon(\kappa_1+\kappa_2)}[$, denoted

$$\mathsf{NIZKPoK} \quad x \quad : \quad y = g^x \ \text{ and } \ x \in]X - 2^{\epsilon(\kappa_1+\kappa_2)}, X + 2^{\epsilon(\kappa_1+\kappa_2)}[\quad .$$

This proof can be computed by choosing a random $r_x \in_R \pm\{0,1\}^{\epsilon(\kappa_1+\kappa_2)}$ and then producing

$$c = \mathsf{Hash}(g, y, g^{r_x}) \ \text{ and } \ s = r_x - c(x - X).$$

$\Diamond$

Remark 3.7.1 The above examples consider that the order of the group $\mathbb{G}$ is unknown, which would be typically the case for cyclic groups in the RSA setting. If the group order is known, i.e. prime Q of length κ, which is the case in the DL setting and if prime order bilinear groups are deployed, then in all proofs computations on the exponents can be performed modulo Q and using a hash function $\mathsf{Hash} : \{0,1\}^* \mapsto \mathbb{Z}_Q$. In particular, random elements r_x used in the respective proofs can be chosen simply from $\mathbb{Z}_Q$, rather than from a larger space $\pm\{0,1\}^{\epsilon(\kappa_1+\kappa_2)}$, which will also lead to shorter proofs in comparison to those in the RSA setting.

3.7.3. Signatures of Knowledge (SoK)

The notion Signature of Knowledge (SoK) has been introduced by Camenisch [45] in the context of group signatures and later formalized by Chase and Lysyanskaya [67]. SoK signatures combine the properties of digital signatures presented in Section 3.4 and NIZKPoK proofs from Section 3.7.2. A SoK signature σ on a message m with respect to a pair (C, w) satisfying some relation R can be easily obtained from a ZKPoK Σ-protocol that proves the relation $(C, w) \in R$ using the Fiat-Shamir transformation [89], similarly to the construction of the corresponding NIZKPoK proof for $(C, w) \in R$, except that now the hash function $\mathsf{Hash} : \{0,1\}^* \mapsto \{0,1\}^\kappa$ takes as input m in addition to the statement C and the commitment message CMT. This also ensures the unforgeability of SoK signatures in the Random Oracle Model. Note that SoK signatures satisfy the NIZKPoK requirements of completeness, soundness, PoK and ZK properties, when viewed as a NIZKPoK proof for $(C, w) \in R$. In fact, when viewed from the perspective of (traditional) digital signatures, (C, R) can be seen as a public key and witness w as the corresponding private key.

Similarly to ZKPoK and NIZKPoK proofs, we will use the following notation while referring to SoK signatures on a message m with an associated NIZKPoK proof for some $(C, w) \in R$:

$$\mathsf{SoK} \quad \text{witness } w \quad : \quad \text{relation} \quad R \quad \text{that} \quad (C, w) \quad \text{must satisfy} \qquad \text{message } m$$

We implicitly assume that such SoK signature contains all information about C and R that is necessary to check its validity. Note that concrete examples of different SoK signatures can easily be obtained from the examples of NIZKPoK proofs in the previous section, by including m as one of the inputs to the hash function Hash; as shown below for case of knowledge of a discrete logarithm.

Definition 3.26 (SoK of a Discrete Logarithm) Let $g, y \in \mathbb{G}$ and $m \in \{0,1\}^*$. A pair $(c, s) \in \{0,1\}^{\kappa_2} \times \pm\{0,1\}^{\epsilon(\kappa_1+\kappa_2)+1}$ satisfying $c = \mathsf{Hash}(g, y, g^s y^c, m)$ is a SoK signature on m for the knowledge of the discrete logarithm of y with respect to the base g, denoted

$$\mathsf{SoK} \quad x \quad : \quad y = g^x \qquad m \quad .$$

This signature can be computed by choosing a random $r_x \in_R \pm\{0,1\}^{\epsilon(\kappa_1+\kappa_2)}$ and then computing

$$c = \mathsf{Hash}(g, y, g^{r_x}, m) \quad \text{and} \quad s = r_x - cx.$$

$\diamondsuit$

4. Group Signatures based on General Assumptions

In order to get intuition about the cryptographic design of group signature schemes it is advisable to first consider generic solutions that build upon cryptographic primitives used in a black-box way. Such generic constructions are usually less efficient than concrete solutions based on number-theory since they do not necessarily admit optimizations on the algorithmic level. Nevertheless, such generic constructions are interesting from several points of view: They shed light on the general hardness assumptions needed to prove security of group signature schemes, and they motivate and explain design principles of group signature schemes built from concrete number-theoretic assumptions. In this section we first present a generic construction of a static group signature scheme introduced by Bellare, Micciancio, and Warinschi [22]. Then, we describe its dynamic version proposed by Bellare, Shi, and Zhang [25]. These schemes apply the so-called **"sign-and-encrypt-and-prove" paradigm**, also used in the construction of many other group signature schemes.

4.1. The Bellare-Micciancio-Warinschi Scheme — "Sign-and-Encrypt-and-Prove" Paradigm

In addition to the first general formal model for static group signature schemes that has also been used to motivate our definitions in Section 2.1, Bellare, Micciancio, and Warinschi [22] gave a suitable construction based on a digital signature scheme, a public key encryption scheme, and non-interactive zero-knowledge proofs of knowledge. We refer to their scheme as BMW scheme. From the historic perspective, the concept underlying the design of the BMW scheme is related to the notion of verifiable encryption (of digital signatures) used previously by Camenisch and Michels [60].

4.1.1. The BMW Scheme

The BMW scheme utilizes the following generic cryptographic building blocks:

- A digital signature scheme $\Sigma = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vrfy})$ specified in Section 3.4.
- A public-key encryption scheme $\Theta = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ specified in Section 3.5.
- A non-interactive zero-knowledge proof of knowledge (NIZKPoK) scheme (K, P, V) specified in Section 3.7.2. The underlying NP relation R contains statement-witness pairs of the form $((pk_e, pk_s, m, c), (i, pk_i, \mathrm{cert}_i, s))$ with $\mathsf{Vrfy}(pk_s, (i, pk_i), \mathrm{cert}_i) = 1$, $\mathsf{Vrfy}(pk_i, m, s) =$

1, and $\mathsf{Enc}(pk_e, (i, pk_i, \mathrm{cert}_i, s)) = c$, where pk_e is a public key suitable for Θ, pk_s is a public key suitable for Σ, pk_i is the public key of user i suitable for Σ, cert_i and s are digital signatures, m is a message, and c is a ciphertext.

The NP relation underlying the NIZKPoK proof motivates at a high level the actual construction of the BMW scheme. Namely, the secret signing key of a member i consists of a secret key sk_i and a digital signature cert_i issued by the group manager on the pair (i, pk_i). That is cert_i can be seen as group manager's certificate on the member's public key pk_i. In order to produce a group signature σ on some message m, member i encrypts $(i, pk_i, \mathrm{cert}_i)$ under the group manager's public key pk_e, computes an ordinary signature s on m using own secret key sk_i, and proves the correctness of these computations as well as the possession of a valid cert_i through the NIZKPoK proof. In the following we detail the BMW construction, following the specification from [22].

Key generation. The key generation algorithm GKg on input the security parameter 1^κ, $\kappa \in \mathbb{N}$ and the number of members $n \in \mathbb{N}$ performs the following steps:

1. Compute the common reference string $\rho \leftarrow_R K(1^\kappa)$.
2. Compute the private/public key pair $(sk_s, pk_s) \leftarrow_R \Sigma.\mathsf{Kg}(1^\kappa)$ for the digital signature scheme Σ.
3. Compute the private/public key pair $(sk_e, pk_e) \leftarrow_R \Theta.\mathsf{Kg}(1^\kappa)$ for the public key encryption scheme Θ.
4. For $i = 1$ to n: compute $(sk_i, pk_i) \leftarrow_R \Sigma.\mathsf{Kg}(1^\kappa)$ and $\mathrm{cert}_i \leftarrow_R \mathsf{Sign}(sk_s, (i, pk_i))$.
5. Output $(gpk, gmsk, \boldsymbol{gsk})$ such that:
    - group public key $gpk = (\rho, pk_s, pk_e)$
    - group manager's secret key $gmsk = (gpk, sk_e)$
    - member i's secret signing key $\boldsymbol{gsk}[i] = (gpk, sk_i, pk_i, \mathrm{cert}_i)$.

Algorithms K and Kg are assumed to use internally security parameters that are polynomially related to the overall security parameter κ. The key generation algorithm is executed in a trusted way. In particular, the private key sk_s should be erased as it is not part of $gmsk$. Alternatively, one can assume two distributed authorities, the issuer and the opener, where the issuer would generate (sk_s, pk_s) and issue certificates cert_i, while the opener would generate (sk_e, pk_e). This distributed approach is explicitly used in the dynamic version of the scheme, which we introduce in Section 4.2. Within the static BMW scheme such issuer would be required during the key generation procedure only.

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, sk_i, pk_i, \mathrm{cert}_i)$ of member i, where $gpk = (\rho, pk_s, pk_e)$, and a message $m \in \{0,1\}^*$, and proceeds as follows:

1. Compute $s \leftarrow_R \mathsf{Sign}(sk_i, m)$ and $c \leftarrow_R \mathsf{Enc}(pk_e, (i, pk_i, \mathrm{cert}_i, s))$.

2. Compute π as a non-interactive zero-knowledge proof of knowledge

$$\mathsf{NIZKPoK}\left[\begin{array}{ccc} & & \mathsf{Vrfy}(pk_s, (i, pk_i), \mathrm{cert}_i) = 1 \\ i, pk_i, \mathrm{cert}_i, s & : & \mathsf{Vrfy}(pk_i, m, s) = 1 \\ & & \mathsf{Enc}(pk_e, (i, pk_i, \mathrm{cert}_i, s)) = c \end{array}\right].$$

3. Output group signature $\sigma = (c, \pi)$.

The ciphertext c encrypts the certificate cert_i, the public key pk_i of member i and his signature s under the public key pk_e of the group manager. Additionally, the NIZKPoK proof π proves in zero-knowledge fashion that c encrypts these values and that, furthermore, cert_i is a valid signature on (i, pk_i) issued by the group manager and that member's signature s can be indeed verified using the encrypted public key pk_i, thus implicitly meaning that s was computed by i using the corresponding private key sk_i. This is the essence of the "sign-and-encrypt-and-prove" paradigm that is realized by many modern group signature schemes using concrete cryptographic settings and number theory.

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (\rho, pk_s, pk_e)$, a message m, and a candidate group signature σ, and proceeds as follows:

1. Parse σ as (c, π).

2. If π is a valid NIZKPoK proof then output 1; otherwise output 0.

Opening procedure. The opening algorithm Open takes as input the group manager's secret key $gmsk = (gpk, sk_e)$, message m, and a group signature σ, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.

2. Decrypt $(i, pk_i, \mathrm{cert}_i, s) \leftarrow \mathsf{Dec}(sk_e, c)$.

3. If $n < i$ or $\mathsf{Vrfy}(pk_i, m, s) = 0$ or $\mathsf{Vrfy}(pk_s, (i, pk_i), \mathrm{cert}_i) = 0$ then output 0; otherwise output i.

The opening procedure verifies that the decrypted tuple $(i, pk_i, \mathrm{cert}_i, s)$ is correctly formed. In particular, that cert_i is indeed a valid certificate on (i, pk_i) and that signature s is a valid signature under pk_i.

Remark 4.1.1 The BMW scheme comes without verifiability of the opening procedure. It seems, however, possible to extend the scheme towards this property using an additional NIZKPoK proof for the NP relation given by statement-witness pairs $((pk_e, c, i, pk_i, \mathrm{cert}_i, s), sk_e)$ for which $\mathsf{Dec}(sk_e, c) = (i, pk_i, \mathrm{cert}_i, s)$. In this case the opening algorithm would output, in addition to i, a proof τ containing $(pk_i, \mathrm{cert}_i, s, J)$ where J would be the mentioned NIZKPoK proof. The judgement algorithm would then verify the validity of J, in addition to the validity of the group signature σ, cert_i and s (as in the opening procedure). In fact this approach was used by Bellare, Shi, and Zhang [25] in the context of their dynamic group signature scheme, which we describe in Section 4.2. Furthermore, as mentioned by Galindo et al. [96], verifiability of the opening procedure in group signature schemes that follow the design principle of the BMW scheme and its dynamic variant from [25] can be obtained by using a different flavor of public key encryption schemes that in addition to being IND-CCA secure have the property of *non-interactive opening* (the so-called PKENO schemes). These schemes implicitly allow the decrypting party, being in possession of sk_e, to actually prove that some message was encrypted in the ciphertext in a non-interactive way.

4.1.2. Security of the BMW Scheme

Bellare, Micciancio, and Warinschi [22] proved that their construction satisfies the security notions of full anonymity and full traceability for static group signature schemes, whereby their notion of full traceability slightly differs from our notion as explained in Remark 2.1.4. We thus discuss the security of the BMW scheme in the light of our definitions from Section 2.1.

Anonymity. The BMW scheme satisfies the full anonymity notion from Definition 2.4 assuming the IND-CCA security of the encryption scheme Θ (see Definition 3.18), the computational ZK property of the NIZKPoK proof π, and its simulation-soundness property [165]. In particular, even if $\mathcal{A}$ learns all secret signing keys in $\boldsymbol{gsk}$ it cannot distinguish which signer produced the challenge group signature $\sigma^* = (c^*, \pi^*)$ due to the security of the NIZKPoK proof.

Traceability. The BMW scheme provides full traceability from Definition 2.6 assuming the unforgeability of the digital signature scheme Σ (see Definition 3.16) and the soundness property of the NIZKPoK scheme (K, P, V) for relation R. Observe that in order to break full traceability the adversary would have to output a pair (m^*, σ^*) with $\sigma^* = (c^*, \pi^*)$ that would pass the verification procedure, yet result in the opening algorithm outputting 0. The soundness property behind the NIZKPoK proof π^* implies that c^* encrypts $(i, pk_i, \mathrm{cert}_i, s)$ where cert_i is a valid certificate on (i, pk_i) and s is a valid signature on m that can be verified using pk_i. Hence, in order to break traceability the adversary would have to either break the soundness property regarding π^* or to come up with a forged certificate cert_i that has never been issued by the group manager.

Non-frameability. The BMW scheme guarantees the full non-frameability notion from Definition 2.8 under the sole assumption that the digital signature scheme Σ is unforgeable. In order to break the full non-frameability property the adversary that now knows $gmsk$ would have to output a pair (m^*, σ^*) with $\sigma^* = (c^*, \pi^*)$ that would pass the verification procedure and result in the opening algorithm outputting some identity $i^* \in [1, n]$, yet without knowing $\boldsymbol{gsk}[i^*]$ and without having previously obtained a group signature on m^* through the signing oracle. The knowledge of $gmsk$ allows the adversary to reveal cert_{i^*}, $i^* \in [1, n]$ from any valid group signature of i^*. On the other hand, $gmsk$ does not include sk_s, thus preventing the adversary from issuing rogue certificates cert_{i^*} on public keys of its own choice. The only secret information that remains hidden from the adversary is the corresponding private key sk_{i^*}. Therefore, in order to break full non-frameability the adversary would have to come up with a forgery for the signature s on behalf of i^*.

Remark 4.1.2 The entire security of the BMW scheme can be reduced to the sole assumption that trapdoor permutations exist. The reason is that this assumption is the minimal assumption, which is necessary to show existence of an IND-CCA secure public key encryption scheme Θ. Moreover, this assumption suffices to prove existence of simulation-sound NIZKPoK proofs for arbitrary NP relations. The existence of an unforgeable digital signature scheme Σ can, however, be shown based on a weaker assumption, namely the existence of one-way functions [163]. A natural question in this context is, whether assuming existence of trapdoor permutations is necessary for a static group signature scheme to be fully anonymous, fully traceable, and fully non-frameable? In the follow-up work Abdalla and Warinschi [2] were able to prove that existence of such group signature schemes implies existence of IND-CCA secure public key encryption. Hence, it is unlikely that one can construct a group signature scheme with the above mentioned security properties, assuming solely the existence of one-way functions. This makes group signature schemes more complex primitives than e.g. ordinary signature schemes.

4.2. The Bellare-Shi-Zhang Scheme

Bellare, Shi, and Zhang [25] gave a generic construction of a dynamic group signature scheme that also provides verifiable opening procedure and splits the role of the group manager into two distributed authorities — the issuer and the opener. Furthermore, their scheme, which we refer to as BSZ, involves user PKI for potential group members. In the following we describe algorithms and protocols of the BSZ scheme and discuss its security.

4.2.1. The BSZ Scheme

The BSZ scheme uses the same cryptographic building blocks as the static BMW scheme, namely an unforgeable digital signature scheme $\Sigma = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vrfy})$, a public key encryption scheme $\Theta = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$, and two non-interactive zero-knowledge proof of knowledge (NIZKPoK) schemes: (1) (K_1, P_1, V_1) for the NP relation R_1 containing statement-witness pairs of the form $((pk_e, pk_s, m, c), (i, pk_i, \mathrm{cert}_i, s))$ identical to those used in the BMW scheme, and (2) (K_2, P_2, V_2) for the NP relation R_2 containing statement-witness pairs of the form $((pk_e, c, i, pk_i, \mathrm{cert}_i, s), sk_e)$ for which $\mathsf{Dec}(sk_e, c) = (i, pk_i, \mathrm{cert}_i, s)$ as described in Remark 4.1.1. We now proceed with the specification of algorithms and protocols of the BSZ scheme.

Key generation. The key generation algorithm GKg on input the security parameter 1^κ, $\kappa \in \mathbb{N}$ proceeds as follows:

1. Compute common reference strings $\rho_1 \leftarrow_R K_1(1^\kappa)$ and $\rho_2 \leftarrow_R K_2(1^\kappa)$.
2. Compute the private/public key pair $(sk_s, pk_s) \leftarrow_R \Sigma.\mathsf{Kg}(1^\kappa)$ for the digital signature scheme Σ.
3. Compute the private/public key pair $(sk_e, pk_e) \leftarrow_R \Theta.\mathsf{Kg}(1^\kappa)$ for the public key encryption scheme Θ.
4. Output $(gpk, ik, ok, \boldsymbol{reg})$ such that:

- group public key $gpk = (\rho_1, \rho_2, pk_s, pk_e)$
- secret issuing key $ik = (gpk, sk_s)$
- secret opening key $ok = (gpk, sk_e)$
- initially empty registration list $\boldsymbol{reg}$.

User key generation. The user key generation algorithm UKg on input the security parameter 1^κ, $\kappa \in \mathbb{N}$ computes and returns the private/public key pair $(\boldsymbol{usk}[i], \boldsymbol{upk}[i]) \leftarrow_R \Sigma.\mathsf{Kg}(1^\kappa)$ for the digital signature scheme Σ.

As noticed in Section 2.3.2 user PKI is modeled here through public (read) access to the list of registered public keys $\boldsymbol{upk}$.

Join protocol. The join protocol Join is executed between the issuer with input $ik = (gpk, sk_s)$ and a prospective member i with input $gpk = (\rho_1, \rho_2, pk_s, pk_e)$ and own PKI-certified key pair $(\boldsymbol{usk}[i], \boldsymbol{upk}[i])$. It proceeds as follows:

1. Member i generates $(sk_i, pk_i) \leftarrow_R \Sigma.\mathsf{Kg}(1^\kappa)$ and computes $sig_i \leftarrow_R \mathsf{Sign}(\boldsymbol{usk}[i], (i, pk_i))$. Member i sends (pk_i, sig_i) to the issuer.
2. The issuer proceeds if $\mathsf{Vrfy}(\boldsymbol{upk}[i], (i, pk_i), sig_i) = 1$. The issuer sends $\mathrm{cert}_i \leftarrow_R \mathsf{Sign}(sk_s, (i, pk_i))$ back to i and stores $\boldsymbol{reg}[i] = (i, pk_i, sig_i)$.
3. Member i checks whether $\mathsf{Vrfy}(pk_s, (i, pk_i), \mathrm{cert}_i) = 1$ and if so stores $\boldsymbol{gsk}[i] = (gpk, sk_i, pk_i, \mathrm{cert}_i)$ as its secret signing key.

The joining procedure should be executed over a secure channel in order to protect the transmission of (pk_i, sig_i) and cert_i. Furthermore, it is implicitly assumed PKI-certified public keys $\boldsymbol{upk}[i]$ of joining members are known to the issuer, who is also supposed to check their validity. In fact, user PKI can also be helpful for the establishment of the required secure channel. Note also that the registration information $\boldsymbol{reg}[i]$ contains sig_i which can be verified using PKI-certified $\boldsymbol{upk}[i]$ and can thus be used later by the opener, who has read access to $\boldsymbol{reg}$.

Signature generation. The signing algorithm GSign of the BSZ scheme proceeds identically to the signing algorithm of the BMW scheme from Section 4.1. The resulting group signature on a message m generated by the signer i is thus $\sigma = (c, \pi)$ where c, being a ciphertext under pk_e, is part of the statement for the NP relation R_1, which encrypts the witness $(i, pk_i, \mathrm{cert}_i, s)$, and π is the corresponding NIZKPoK proof.

Signature verification. The signature verification algorithm GVrfy of the BSZ scheme is also identical to the verification algorithm of the BMW scheme from Section 4.1. The verification of $\sigma = (c, \pi)$ on a message m boils down to the verification of the corresponding NIZKPoK proof π.

Opening procedure. The opening algorithm Open takes as input the secret opening key $ok = (gpk, sk_e)$, message m, group signature σ, and the registration list $\boldsymbol{reg}$, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output $(0, \perp)$.
2. Decrypt $(i, pk_i, \mathrm{cert}_i, s) \leftarrow \mathsf{Dec}(sk_e, c)$.
3. If $n < i$ or $\mathsf{Vrfy}(pk_i, m, s) = 0$ or $\mathsf{Vrfy}(pk_s, (i, pk_i), \mathrm{cert}_i) = 0$ then output $(0, \perp)$.
4. Find i such that $\boldsymbol{reg}[i] = (i, pk_i, sig_i)$.
5. Compute J as $\mathsf{NIZKPoK} \quad sk_e \quad : \quad \mathsf{Dec}(sk_e, c) = (i, pk_i, \mathrm{cert}_i, s) \quad .$
6. Output (i, τ) where $\tau = (pk_i, \mathrm{cert}_i, s, J, sig_i)$.

The opening procedure of the BSZ scheme is similar to that of the BMW scheme from Section 4.1, except that in addition to the identity i of the signer it outputs proof τ that includes the decrypted tuple $(i, pk_i, \mathrm{cert}_i, s)$ together with the NIZKPoK proof J testifying that decryption was performed correctly and signature sig_i that links the signer i to the PKI-certified public key $\boldsymbol{upk}[i]$.

Judgement procedure. The judgement algorithm Judge takes as input the group public key $gpk = (\rho_1, \rho_2, pk_s, pk_e)$, message m, group signature σ, identity i, and proof τ, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.
2. Parse τ as $(pk_i, \mathrm{cert}_i, s, J, sig_i)$.
3. If all of the following holds then output 1; otherwise output 0:
   - J is a valid NIZKPoK proof
   - Retrieve $\boldsymbol{upk}[i]$
   - $\mathsf{Vrfy}(\boldsymbol{upk}[i], (i, pk_i), sig_i) = 1$

The judgement procedure first ensures the validity of the group signature, which implicitly provides assurance for the validity of the encrypted tuple $(i, pk_i, \mathrm{cert}_i, s)$. Through the additional verification of the NIZKPoK proof J it obtains confidence that decryption was performed correctly. The actual identification of the signer i is then verified using the signer's PKI-certified public key $\boldsymbol{upk}[i]$ and the signature sig_i. It is implicitly assumed that identity i points to the candidate public key $\boldsymbol{upk}[i]$ used in this final verification step.

4.2.2. Security of the BSZ Scheme

Bellare, Shi, and Zhang [25] were able to prove that security of their scheme relies on the existence of trapdoor permutations (which also underlies the security of the static BMW scheme). Towards this statement they analyzed anonymity, traceability, and non-frameability of the scheme using definitions that in part correspond to our definitions for DA-schemes with verifiable opening and user PKI from Section 2.4.

Anonymity. The BSZ scheme satisfies the full anonymity notion for DA-schemes (with verifiable opening) from Definition 2.25 assuming the IND-CCA security of the encryption scheme Θ (see Definition 3.18), the ZK property of the NIZKPoK proof (K_1, P_1, V_1) for relation R_1, and its simulation-soundness property [165]. In particular, even if $\mathcal{A}$ corrupts all members of the group, obtaining all secret signing keys in $\boldsymbol{gsk}$ and all PKI-certified secret keys in $\boldsymbol{usk}$, it cannot distinguish which signer produced the challenge group signature $\sigma^* = (c^*, \pi^*)$ due to the security of the corresponding NIZKPoK proof.

Traceability. The BSZ scheme being dynamic cannot provide full traceability. However, it can be shown that BSZ scheme satisfies the weaker notion of insider traceability from Definition 2.26 assuming the unforgeability of the digital signature scheme Σ (see Definition 3.16) and the soundness property of the NIZKPoK scheme (K_1, P_1, V_1) for relation R_1. Observe that in order to break insider traceability the adversary would have to output a pair (m^*, σ^*) with $\sigma^* = (c^*, \pi^*)$ that would pass the verification procedure, yet result either in the opening algorithm outputting $(0, \perp)$ or in the judgement algorithm outputting 0. The soundness property of the NIZKPoK proof behind π^* implies that c^* encrypts $(i, pk_i, \mathrm{cert}_i, s)$ where cert_i is a valid certificate on (i, pk_i) and s is a valid signature on m that can be verified using pk_i. Hence, in order for the opening algorithm to output $(0, \perp)$ the adversary would have to either break the soundness property regarding π^* or to come up with a forged certificate cert_i that has never been issued by the issuer. Since the judgement algorithm verifies the validity of the group signature it will accept the corresponding proof τ^* output by the opening algorithm.

Non-frameability. The BSZ scheme guarantees the full non-frameability notion for DA-schemes (with verifiable opening) from Definition 2.28 under the assumption that the digital signature scheme Σ is unforgeable and the NIZKPoK scheme (K_2, P_2, V_2) for relation R_2 is sound. In order to break the full non-frameability property the adversary that now knows ik and ok would have to output $(m^*, \sigma^*, i^*, \tau^*)$ with $\sigma^* = (c^*, \pi^*)$ that would pass the verification procedure and (i^*, τ^*) that would pass the judgement procedure, yet without knowing $\boldsymbol{gsk}[i^*]$ and $\boldsymbol{usk}[i^*]$ and without having previously obtained a group signature on m^* through the signing oracle of i^*. Note that τ^* contains sig_{i^*} and J^* that are verified by the judgement algorithm. The knowledge of ik allows the adversary to create rogue certificates cert_{i^*} on public keys pk_{i^*} of its own choice. Hence, in order to break full non-frameability the adversary would have to either forge signature sig_{i^*} under $\boldsymbol{upk}[i^*]$ or break the soundness property regarding J^*.

5. Group Signatures in the RSA Setting

Many group signature schemes were constructed using the RSA setting. Earlier schemes, including RSA-based constructions proposed in the seminal work of Chaum and van Heyst [71] suffered in particular from long sizes of group public keys and signatures, and could only partially satisfy the security requirements, which group signatures are expected to fulfill today. The first (dynamic) group signature scheme in the RSA setting with constant key and signature lengths was introduced by Camenisch and Stadler [61] and more efficient constructions were proposed soon thereafter by Ateniese and Tsudik [16] for dynamic groups and by Camenisch and Michels [58] for static groups with verifiable opening procedure. The first group signature scheme in the RSA setting that was also provably secure against coalitions of signers (with the group manager) was constructed by Ateniese, Camenisch, Joye, and Tsudik [11]. None of these schemes considered revocation and so several extensions were proposed later to deal with this problem. For example, Bresson and Stern [42] showed how to modify the scheme from [61] in order to revoke signing rights without sacrificing anonymity. Ateniese, Song, and Tsudik [15] proposed revocation solution for [11] that, however, required zero-knowledge proofs for special type of relations that cannot be realized efficiently. A better revocation approach for the scheme from [11] was proposed by Camenisch and Lysyanskaya [56] based on dynamic accumulators. This revocation mechanism has been then modified by Tsudik and Xu [178], who also proposed another group signature scheme for dynamic schemes with better efficiency regarding the admission procedure. Nakanishi and Sugiyama [149], based on previous two approaches, designed a group signature scheme for dynamic groups with distributed authorities and reduced costs for the public membership information and computations of the group manager. This scheme has been further improved by Nakanishi et al. [148] towards more efficient update of secret signing key performed by the (unrevoked) group members, yet with higher amount of work for the group manager. Camenisch and Groth [49] came up with several efficient schemes in the RSA setting for static and dynamic groups that can also handle revocation. Kiayias and Yung [119, 121] proposed another dynamic group signature scheme, using some ideas from [11], which they however analyzed in a formal model similar to [22, 25] and for which they could prove anonymity against malicious group managers. They also gave intuition on how to adopt the scheme to the setting of distributed authorities.

In the following we give a detailed overview of several group signature schemes in the RSA setting. We start with the Ateniese-Camenisch-Joye-Tsudik (ACJT) scheme [11], for which we also describe the revocation mechanism based on dynamic accumulators from [56]. We then focus on the revocable dynamic group signature scheme by Tsudik and Xu [178] and on the schemes by Camenisch and Groth [49]. The last group signature in the RSA setting which we address is by Kiayias and Yung [119, 121].

5.1. The Ateniese-Camenisch-Joye-Tsudik Scheme

In this section we present the group signature scheme proposed by Ateniese, Camenisch, Joye, and Tsudik [11]. This scheme, which we refer to as ACJT, is dynamic and offers verifiable opening. According to our classification it thus belongs to group signature schemes that we defined in Section 2.3. Historically, the ACJT scheme can be seen as evolution of several previous proposals by Camenisch and Stadler [61] (some weaknesses of their scheme were later identified in [17]), Camenisch and Michels [58], and Ateniese and Tsudik [16]. The main advantage of the ACJT scheme was that it was the first practical scheme to generate constant-length group signatures and public group keys, while also offering coalition-resistance property, which is subsumed by our definitions of traceability and non-frameability.

5.1.1. The ACJT Scheme

The ACJT group signature scheme is dynamic and offers verifiable opening. It uses RSA setting and has several security parameters, described in the following: Let $\epsilon > 1$ and ${}_c$, ${}_p \in \mathbb{N}$, where ${}_p$ determines the size of the modulus N and ${}_c$ denotes the output length of the used hash function. Let $\lambda_1, \lambda_2, \gamma_1, \gamma_2$ denote lengths and Λ, Γ denote two integral ranges satisfying the following conditions:

- $\lambda_1 > \epsilon(\lambda_2 + {}_c) + 2$ and $\lambda_2 > 4\ {}_p$
- $\Lambda =]2^{\lambda_1 - \lambda_2}, 2^{\lambda_1 + \lambda_2}[$
- $\gamma_1 > \epsilon(\gamma_2 + {}_c) + 2$ and $\gamma_2 > \lambda_1 + 2$
- $\Gamma =]2^{\gamma_1 - \gamma_2}, 2^{\gamma_1 + \gamma_2}[$

The two values λ_1, λ_2 stretch the integral range Λ for the members secret x_i. The same shall apply for γ_1, γ_2, stretching Γ for the random prime values e_i. Thus the values $\lambda_1, \lambda_2, \gamma_1, \gamma_2$ determine the hardness of the underlying strong RSA assumption.

In the following we specify the core algorithms and protocols of the ACJT scheme. Our description follows the specification from [11], except that we also provide specification of the judgement procedure that has been omitted in the original work.

Key generation. The key generation algorithm GKg on input ${}_p$ performs the following steps:

1. Compute safe RSA modulus N as a product of two safe primes $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are primes of length ${}_p$.
2. Choose random elements $a, a_0, g, h \in_R QR(N)$ (of order $p'q'$).
3. Pick random $x \in_R \mathbb{Z}^*_{p'q'}$, compute $y = g^x \mod N$.
4. Output $(gpk, gmsk, \boldsymbol{reg})$ such that:
    - group public key $gpk = (N, a, a_0, y, g, h)$

- group manager's secret key $gmsk = (gpk, p', q', x)$
- registration list ***reg*** is empty.

It is assumed that key generation is performed in a trusted way. In particular, this means that the RSA modulus N is indeed safe and that elements a, a_0, g, h are chosen independently at random from $QR(N)$. This assumption is necessary to ensure trust into the group public key gpk. In case that this trust is unavailable the correct generation of gpk can be ensured through costly NIZKPoK proofs proposed by Camenisch and Michels in [59].

Join protocol. The join protocol Join is executed between the group manager GM with input $gmsk = (gpk, p', q', x)$ and a prospective member i with input $gpk = (N, a, a_0, y, g, h)$. It proceeds as follows:

1. Member i picks random $\tilde{x}_i \in_R]0, 2^{\lambda_2}[$ and $\tilde{r} \in_R]0, N^2[$, computes $C_1 = g^{\tilde{x}_i} h^{\tilde{r}_i} \mod N$, and sends C_1 to GM together with a proof
$$\mathsf{ZKPoK}\left[\tilde{x}_i, \tilde{r}_i \;:\; C_1 = g^{\tilde{x}_i} h^{\tilde{r}_i} \mod N\right].$$
2. GM proceeds if $C_1 \in QR(N)$ and the above proof was correct. GM picks random $\alpha_i, \beta_i \in_R]0, 2^{\lambda_2}[$ and sends (α_i, β_i) back to i.
3. Member i computes $x_i = 2^{\lambda_1} + (\alpha_i \tilde{x}_i + \beta_i \mod 2^{\lambda_2})$, and sends $C_2 = a^{x_i} \mod N$ to GM together with a proof
$$\mathsf{ZKPoK}\left[x_i, u, v, w \;:\; \begin{array}{c} C_2 = a^{x_i} \text{ and } x_i \in \Lambda \\ a^u = C_2 / a^{2^{\lambda_1}} \text{ and } u \in]-2^{\lambda_2}, 2^{\lambda_2}[\\ C_1^{\alpha_i} g^{\beta_i} = g^u (g^{2^{\lambda_2}})^v h^w \end{array} \right].$$
4. GM proceeds if $C_2 \in QR(N)$ and the above proof was correct. GM picks random prime $e_i \in_R \Gamma$ and computes $A_i = (C_2 a_0)^{1/e_i} \mod N$, sets $\boldsymbol{reg}[i] = (A_i, e_i, \mathtt{trans}_i)$ where $\mathtt{trans}_i$ is the communication transcript, and sends (A_i, e_i) back to i.
5. Member i checks whether $a^{x_i} a_0 = A_i^{e_i} \mod N$ and if so stores $\boldsymbol{gsk}[i] = (gpk, x_i, A_i, e_i)$ as its secret signing key.

The above join protocol should be performed over a secure channel in order to prevent leakage of the pair (A_i, e_i) which is sent in clear to member i. Moreover, communication transcripts $\mathtt{trans}_i$ that the group manager stores in ***reg*** are assumed to be authenticated by i in a way that any publication of $\mathtt{trans}_i$ uniquely identifies i who participated in the join protocol. These assumptions can be realized using PKI in which candidate members have certified private/public key pairs. That is i can sign the transcript using those keys.

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, x_i, A_i, e_i)$ of member i, where $gpk = (N, a, a_0, y, g, h)$, and a message $m \in \{0,1\}^*$, and proceeds as follows:

1. Pick random $w \in_R \{0,1\}^{2\ell_p}$ and compute
$$T_1 = A_i y^w \mod N, \quad T_2 = g^w \mod N, \quad T_3 = g^{e_i} h^w \mod N.$$

2. Compute S as a signature of knowledge
$$\mathsf{SoK}\left[\ \alpha,\beta,\gamma,\delta,\varepsilon \quad : \quad \begin{array}{c} a_0 = T_1^{\alpha} \ \frac{1}{a}^{\ \beta} \left(\frac{1}{y}\right)^{\gamma} \text{ and } 1 = T_2^{\alpha}\left(\frac{1}{g}\right)^{\gamma} \text{ and } T_2 = g^{\delta} \\ T_3 = g^{\alpha}h^{\varepsilon} \text{ and } \alpha \in \Lambda \text{ and } \beta \in \Gamma \end{array}\ \right] \quad m \quad .$$

3. Output group signature $\sigma = (S, T_1, T_2, T_3)$.

In the above signature generation algorithm (T_1, T_2) is ElGamal encryption [88] of A_i under the public key y in the $QR(N)$ group. The additional value T_3 is a Pedersen commitment [159] to the exponent e_i. The SoK signature thus proves that the signer is in possession of a pair (A_i, e_i) such that $A_i = (a^{x_i} a_0)^{1/e_i} \mod N$; thus, proving that the signer has a valid signing key $\boldsymbol{gsk}[i]$ and that σ can be opened by the group manager, who can decrypt A_i from the ciphertext.

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (N, a, a_0, y, g, h)$, a message m, and a candidate group signature σ, and proceeds as follows:

1. Parse σ as (S, T_1, T_2, T_3).

2. If S is a valid SoK signature on message m then output 1; otherwise output 0.

Opening procedure. The opening algorithm Open takes as input the group manager's secret key $gmsk = (gpk, p', q', x)$, message m, group signature σ, and registration list $\boldsymbol{reg} = \{(A_i, e_i, \mathtt{trans}_i)_{i\in[1,n]}\}$, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.

2. Compute $A_i = \frac{T_1}{T_2^x} \mod N$.

3. Find i for which $\boldsymbol{reg}[i] = (A_i, e_i, \mathtt{trans}_i)$.

4. Compute J as $\mathsf{NIZKPoK} \quad x \quad : \quad y = g^x \text{ and } \frac{T_1}{A_i} = T_2^x \quad .$

5. Output (i, τ), where $\tau = (J, \mathtt{trans}_i)$.

Note that opening proof τ contains the NIZKPoK of the group manager that the group signature's components T_1 and T_2 encrypt value A_i, which is part of the disclosed communication transcript $\mathtt{trans}_i$.

Judgement procedure. The judgement algorithm Judge takes as input the group public key $gpk = (N, a, a_0, y, g, h)$, message m, signature σ, identity i, and proof τ, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.
2. Parse τ as (A_i, J).
3. If all of the following holds then output 1; otherwise output 0:
   - J is a valid NIZKPoK proof
   - A_i is included in $\texttt{trans}_i$
   - $\texttt{trans}_i$ is authenticated by i.

The validity of the NIZKPoK proof J ensures that the group manager correctly decrypted the claimed value A_i from the ElGamal ciphertext (T_1, T_2). The actual proof that A_i identifies i as a signer of σ stems from the assumption on the join protocol that transcripts $\texttt{trans}_i$ (containing A_i) have been authenticated by i.

5.1.2. Security of the ACJT Scheme

At time when ACJT scheme appeared, formal security models for group signatures were not yet established. The original security proofs for the ACJT scheme, therefore, focused on several building blocks of the scheme and proved separate requirements of unforgeability, anonymity, unlinkability, exculpability, and coalition-resistance. In particular, Ateniese et al. [11] proved that secret signing keys $\boldsymbol{gsk}[i]$ are unforgeable under the Strong RSA assumption, even if the adversary can admit new members and learn their secret signing keys by interacting with the group manager. They also proved that the interactive protocol underlying the SoK signature S is a statistical honest-verifier ZKPoK protocol.

In the following we intuitively discuss security of the ACJT scheme in the light of our definitions for anonymity, traceability, and non-frameability from Sections 2.2 and 2.3.

Anonymity. The ACJT scheme seems to satisfy the full anonymity notion from Definition 2.12 in the Random Oracle Model under the DDH assumption in $QR(N)$ groups (see Definition 3.4). The knowledge of secret signing keys $\boldsymbol{gsk}[i]$ of all group members does not help the adversary $\mathcal{A}$ in distinguishing, whether the challenge signature $\sigma^* = (S^*, T_1^*, T_2^*, T_3^*)$ has been produced by signer i_0 or signer i_1, since the corresponding SoK signature has statistical ZK property, i.e., it does not reveal any information about $\boldsymbol{gsk}[i_b]$, and even if $\boldsymbol{gsk}[i_0]$ and $\boldsymbol{gsk}[i_1]$ are known to $\mathcal{A}$ (which full anonymity implicitly assumes), distinguishing whether these values have been used to compute T_1^*, T_2^*, and T_3^* is hard under the DDH assumption in $QR(N)$ groups.

Traceability. The ACJT scheme is dynamic and cannot satisfy full traceability as mentioned in Section 2.2.4. The ACJT scheme does, however, seems to offer insider traceability from Definition 2.18. Since the group manager remains honest and the join protocol assumes that each joining member i authenticates the transcript $\texttt{trans}_i$, the adversary $\mathcal{A}$ has to come up with the group signature $\sigma^* = (S^*, T_1^*, T_2^*, T_3^*)$ on some message m^* such that σ^* is valid but the algorithm Open outputs either an identity $i^* = 0$ or a pair (i^*, τ^*), which is then rejected by the algorithm Judge. The soundness property of the SoK signature S^* ensures that if σ^* is valid then its components T_1^* and T_2^* encrypt A_{i^*} and T^* is a commitment to e_{i^*} satisfying the relation

$A_{i^*} = (a^{x_{i^*}} a_0)^{1/e_{i^*}} \mod N$ for some x_{i^*} known to the signer. Since forging $(x_{i^*}, A_{i^*}, e_{i^*})$ is computationally infeasible under the Strong RSA assumption (see Definition 3.3) and for each secret signing key issued by the group manager there is a corresponding transcript stored in the ***reg*** list, the probability that Open outputs $i^* = 0$ is negligible. The probability that Judge algorithm rejects the output pair (i^*, τ^*) is negligible since the NIZKPoK proof J is assumed to be sound and the presented transcript $\mathtt{trans}_{i^*}$ is assumed to uniquely identify i^*.

Non-Frameability. The ACJT scheme seems to satisfy the notion of full non-frameability from Definition 2.21. The reason is that Judge algorithm first ensures that (m^*, σ^*) can be successfully verified. The PoK property of the SoK signature S^* (which is part of σ^*) implies that σ^* has been produced with knowledge of $(x_{i^*}, A_{i^*}, e_{i^*})$. Although the adversary $\mathcal{A}$ may learn (A_{i^*}, e_{i^*}) by acting as the group manager in the join protocol, it is hard for $\mathcal{A}$ to actually obtain x_{i^*} due to the hardness of the DL problem in either $\mathbb{Z}_p^*$ or $\mathbb{Z}_q^*$ (recall that $\mathcal{A}$ knows factors of N) and the ZK property of the ZKPoK protocol executed in Step 3 of Join and of the SoK signatures. $\mathcal{A}$ could still generate (m^*, σ^*) on behalf of some corrupted group member and attempt to provide a proof $\tau^* = (\mathtt{trans}_{i^*}, J)$ that this pair opens to some honest member i^*. This is, however, prevented by the soundness property of the NIZKPoK proof J and the assumption that each transcript uniquely identifies the corresponding member.

5.2. The Camenisch-Lysyanskaya Revocation Mechanism for the ACJT Scheme

The original ACJT scheme does not support revocation of group members. There exist several proposals for adding this useful property to the ACJT scheme and its predecessors. For example, Bresson and Stern [42] extended the signing procedure of the Camenisch-Stadler group signature scheme from [61] towards a zero-knowledge proof for proving that the identity of the signer does not appear in the public list of revoked identities. Unfortunately, this technique results in the linear growth of the length of group signatures with the number of revoked signers. Ateniese, Song, and Tsudik [15] introduced three revocation mechanisms for the ACJT scheme. Two of these mechanisms require the group manager to change the group public key gpk and re-issue new pairs (A_i, e_i), which are part of the secret signing key $\boldsymbol{gsk}[i]$ to all remaining group members i, whereby their first approach assumes that the group manager re-computes all these pairs and communicates them to the remaining group members i by means of some secure broadcast channel, while the second approach off-loads the computation of the updated (A_i, e_i) pair to member i at the cost that the opening procedure requires linear number of public-key operations in the number of remaining group members to identify the signer. Their third mechanism is based on the certificate revocation lists and keeps the signature length and the amount of work for the signers constant, yet it employs special zero-knowledge proofs for proving knowledge of double discrete logarithms [61], which are known to be very inefficient.

In this section we focus on the far more efficient extension of the ACJT scheme towards the revocation property proposed by Camenisch and Lysyanskaya [56]. Their extension uses *dynamic accumulators.* The notion of accumulators has been introduced by Benaloh and de

Mare [26], and later studied by Barić and Pfitzmann [20]. Accumulators can combine a set of elements into a short value while also offering short witnesses for each combined (accumulated) value. The main security requirement on such accumulators is that it should be infeasible to find a witness for some value that was not accumulated. Camenisch and Lysyanskaya [56] introduced dynamic accumulators for the accumulation of prime numbers, and showed, how these can be used to revoke members in the ACJT scheme where prime numbers appear as part of the secret signing keys. In the following we provide definitions for dynamic accumulators and present the modified version of the ACJT scheme that can handle membership revocation more efficiently then earlier approaches.

5.2.1. Dynamic Accumulators and Group Management

Dynamic accumulators offer efficient operations for addition and deletion of accumulated elements and corresponding update of witnesses. In Definition 5.1 we specify main algorithms behind dynamic accumulators that allow for public addition of new elements but make the deletion process dependent on some private key. Another important property is that after the removal of some elements from the accumulator, the update of witnesses for remaining elements can be done publicly.

Definition 5.1 (Dynamic Accumulator) A *dynamic accumulator* ($\mathsf{AccGen}, \mathsf{AccAdd}, \mathsf{AccDel}, \mathsf{AccUpdW}, \mathsf{AccVrfy}$) consists of the following five PPT algorithms:

Key generation. The randomized *key generation algorithm* AccGen takes as input a security parameter 1^κ, $\kappa \in \mathbb{N}$ and outputs a private/public key pair (sk, pk), and an empty *accumulator* acc_X, with $X = \emptyset$, where $X \subset \mathcal{X}$ denotes the set of accumulated elements. It is assumed that AccGen determines the space of elements $\mathcal{X}$.

Add elements. The *add algorithm* AccAdd takes as input the public key pk, an element x, an accumulator acc_X, where $x \in \mathcal{X} \setminus X$, and outputs an updated accumulator $acc_{X \cup \{x\}}$ and a *witness* w_x for x.

Delete elements. The *delete algorithm* AccDel takes as input the secret key sk, element x, and an accumulator acc_X, where $x \in X$, and outputs an updated accumulator $acc_{X \setminus \{x\}}$.

Update witness. The *witness update algorithm* AccUpdW takes as input a witness w_x, an element x, an accumulator acc_X, where w_x is a witness for $x \in X$, an element $y = x$, a label $l \in \{\mathtt{add}, \mathtt{del}\}$, and an updated accumulator acc_Y, where Y is either $X \setminus \{y\}$ or $X \cup \{y\}$, and outputs an updated witness w_x for $x \in Y$.

Verification. The deterministic *verification algorithm* AccVrfy takes as input the public key pk, a candidate witness w, an element x, and an accumulator acc_X and outputs either 1 (to indicate that w is a valid witness for $x \in X$) or 0.

In order to be usable for the purpose of membership revocation dynamic accumulators should be quasi-commutative, in the sense that for a given correctly computed accumulator acc_X, and elements $x_1, x_2 \in \mathcal{X} \setminus X$ the following should hold:

$$\mathsf{AccAdd}(pk, x_2, (\mathsf{AccAdd}(pk, x_1, acc_X)) = \mathsf{AccAdd}(pk, x_1, (\mathsf{AccAdd}(pk, x_2, acc_X)).$$

Furthermore, for any PPT adversary $\mathcal{A}$ the probability that $\mathcal{A}$ given the public key pk and access to the delete oracle $\mathsf{AccDel}(sk, \cdot, \cdot)$ outputs a tuple (w^*, x^*, X^*) such that $x^* \notin X^*$ but $\mathsf{AccVrfy}(pk, w^*, x^*, acc_{X^*}) = 1$, where acc_{X^*} is the accumulator obtained by accumulating the elements of X^* using the AccAdd algorithms, is negligible in κ.

A dynamic accumulator ($\mathsf{AccGen}, \mathsf{AccAdd}, \mathsf{AccDel}, \mathsf{AccUpdW}, \mathsf{AccVrfy}$) can be used to implement membership revocation in a dynamic group signature scheme. Roughly, the management of the accumulator is performed by the group manager, who extends its group public key gpk with the public key pk for the accumulator and an accumulator value $acc_{\emptyset}$. Recall from Section 2.2 that group signature schemes with membership revocation maintain an additional update information $\boldsymbol{upd}$, which is assumed to be public. Upon the admission of a new group member i the group manager updates the accumulator acc_X with a fresh value $x_i \in \mathcal{X}$ and provides member i with the corresponding witness w_{x_i}. Additionally, it updates $\boldsymbol{upd}$ with an entry $(x_i, \mathtt{add})$, where $\mathtt{add}$ is a label indicating that x_i belongs to a recently admitted member. Similarly, if some member i is revoked then the corresponding entry $(x_i, \mathtt{add})$ in $\boldsymbol{upd}$ is replaced with $(x_i, \mathtt{del})$. It is assumed that $\boldsymbol{upd}$ implicitly allows parties to recognize all recent updates. In both cases unrevoked group members can use information from $\boldsymbol{upd}$ to directly update their witnesses w_{x_i}. The described procedure ensures that only current group members i are in possession of witnesses w_{x_i} for corresponding elements x_i that are accumulated into the current accumulator acc_X, which is part of the group public key. Security of the accumulator ensures that it is computationally hard to come up with an acceptable pair (w_{x_i}, x_i) for some acc_X in gpk if $x_i \notin X$. Before using this approach for membership revocation in group signature schemes there are two further problems that should be solved first.

The first problem is that elements x_i of admitted group members are made public through the update information $\boldsymbol{upd}$ and thus linkable to a particular group member i. Hence, in order to preserve anonymity the signer should somehow be able to prove that he possesses a witness w_{x_i} for some element x_i in the accumulator acc_X without pointing out, which element this is. One solution to this problem is that the signer commits to x_i using some commitment scheme Commit from Definition 3.19 and the dynamic accumulator offers support for a ZKPoK proof of the form:

$$\mathsf{ZKPoK}\left\{ w_{x_i}, x_i, r \;:\; C = \mathsf{Commit}(x_i, r) \text{ and } \mathsf{AccVrfy}(pk, w_{x_i}, x_i, acc_X) = 1 \right\}.$$

The existence of efficient ZKPoK proofs of this form surely depends on the actual construction of the accumulator.

The second problems is more subtle: Security of the accumulator says nothing about the secrecy of witnesses. It is thus possible that some revoked member i^* obtains a valid pair (w_{x_i}, x_i) for some unrevoked member i and performs the above ZKPoK proof. Therefore, witnesses and/or elements used in the accumulator should somehow be bound to the secret signing keys of members. In particular, it should not be possible to use $\boldsymbol{gsk}[i^*]$ together with a pair (w_i, x_i) unless $i^* = i$. Again, whether an accumulator can be bound to the group signature scheme depends on their actual instantiations.

5.2.2. The Camenisch-Lysyanskaya Accumulator for Prime Numbers

The dynamic accumulator introduced by Camenisch and Lysyanskaya [56] has been designed to accumulate prime numbers. Its construction is given by the following algorithms:

Key generation. The algorithm AccGen takes as input 1^κ, $\kappa \in \mathbb{N}$, runs $\mathsf{RSAGen}(1^\kappa)$ to obtain a safe RSA modulus $N = pq$ of length κ, where $p = 2p' + 1$ and $q = 2q' + 1$ with p', q', p, q being primes. The algorithm also chooses a generator g of the group $QR(N)$. The space of elements is set to

$$\mathcal{X} = \left\{ e_i \mid e_i \text{ is prime } \text{ and } \ e_i = p', q' \text{ and } \ 2 \leq A \leq e_i \leq B < A^2 \right\},$$

where A and B depend polynomially on κ. The algorithm outputs public key $pk = (N, A, B)$, secret key $sk = (p, q)$, and the initial accumulator $acc = g$.

Add element. The algorithm AccAdd takes as input (pk, e, acc), and outputs witness $w = acc$ and updated accumulator $acc^e \mod N$. Note that the previous accumulator is returned as a witness for the added element.

Delete element. The algorithm AccDel takes as input (sk, e, acc), and outputs updated accumulator $acc^{e^{-1} \bmod (p-1)(q-1)} \mod N$. Note that knowledge of factors p and q of N helps to invert e.

Update witness. The algorithm $\mathsf{AccUpdW}$ takes as input $(w, e, acc, \tilde{e}, l, \widetilde{acc})$. If $l = \texttt{add}$ then the algorithm outputs updated witness $w^{\tilde{e}} \mod N$. If $l = \texttt{del}$ then the algorithm computes integers a and b such that $ae + b\tilde{e} = 1$ and outputs updated witness $w^b \cdot \widetilde{acc}^a$. Note that a, b should be computed through the extended Euclidean algorithm (see, e.g. [171]). The existence of such values is guaranteed since $e, \tilde{e} \in \mathcal{X}$ are co-prime.

Verification. The algorithm $\mathsf{AccVrfy}$ takes as input (pk, w, e, acc), and outputs 1 if $w^e = acc \pmod N$; otherwise the algorithm outputs 0.

This accumulator construction ensures that addition of primes $e_1, \ldots, e_n$ results in the accumulator $acc = g^{\prod_i e_i} \in QR(N)$. It is easy to see that if multiple elements should be accumulated by a party which knows sk, then this can be done more efficiently by first computing their product modulo $(p-1)(q-1)$ and then raising the current accumulator acc to the power of that product. Similar efficiency improvement is possible for batch-wise deletion of accumulated elements. The security of this accumulator has been proven under the Strong RSA assumption (Definition 3.3) in [56].

Another property of this accumulator, which becomes necessary for handling the revocation in the ACJT scheme is the existence of an efficient ZKPoK proof for proving knowledge of (w, e) satisfying the verification equation $w^e = acc \mod N$ for some given accumulator acc when e is additionally committed to in $\mathfrak{C}_e = \mathfrak{g}^e \mathfrak{h}^r$ (Pedersen commitment in some suitable group $\mathfrak{G}$ of prime order $\mathfrak{q}$; we use Fraktur letters here to indicate that the setting of commitments differs from that of the accumulator). Highly simplified, the public key pk of the accumulator is extended with two further randomly chosen elements $g, h \in QR(N)$. The prover in possession of (w, e) for accumulator acc and $\mathfrak{C}_e = \mathfrak{g}^e \mathfrak{h}^r$ first picks random $r_1, r_2, r_3 \in \mathbb{Z}_{\lfloor N/4 \rfloor}$, computes $C_e = g^e h^{r_1}$, $C_{acc} = w h^{r_2}$, and $C_r = g^{r_2} h^{r_3}$, and sends $(\mathfrak{C}_e, C_e, C_{acc}, C_r)$ to the verifier that already knows acc. Then, prover and verifier execute the following ZKPoK protocol (for those details we refer to [56]):

$$\mathsf{ZKPoK}\left[\begin{array}{ccc} & & \mathfrak{C}_e = \mathfrak{g}^{\alpha}\mathfrak{h}^{\beta} \text{ and } \mathfrak{g} = \frac{\mathfrak{C}_e}{g}^{\gamma}\mathfrak{h}^{\delta} \\ \alpha, \beta, \gamma, \delta, \varepsilon, \zeta, \varphi, \psi, \eta, \varsigma, \xi & : & \mathfrak{g} = (\mathfrak{g}\mathfrak{C}_e)^{\varepsilon}\mathfrak{h}^{\zeta} \text{ and } C_e = g^{\alpha}h^{\varphi} \text{ and } C_r = g^{\psi}h^{\eta} \\ & & acc = C_{acc}^{\alpha}\frac{1}{h}^{\varsigma} \text{ and } 1 = C_r^{\alpha}\frac{1}{g}^{\varsigma}\frac{1}{h}^{\xi} \end{array}\right].$$

5.2.3. The ACJT Scheme with Membership Revocation

In the following we discuss how the Strong-RSA based dynamic accumulator from the previous section can support revocation when deployed in the original ACJT scheme. The key generation of the original ACJT scheme should be extended with the public parameters of the accumulator. In order to avoid confusion with parameters belonging to the original scheme and parameters belonging to the accumulator we will use Fraktur letters $\mathfrak{N}$, $\mathfrak{p}$, $\mathfrak{q}$, $\mathfrak{g}$, and $\mathfrak{h}$ to indicate those parameters that belong to the accumulator construction. The actual binding between the accumulator and then ACJT scheme is performed by using primes e_i of the ACJT scheme as elements that will be accumulated in acc.

Key generation. The key generation algorithm GKg runs additionally $\mathsf{AccGen}(1^{\kappa})$ and adds public values $(\mathfrak{N}, acc_{\emptyset}, \mathfrak{g}, \mathfrak{h})$ to the group public key gpk. It then extends $gmsk$ with primes $\mathfrak{p}, \mathfrak{q}$ (factors of the accumulator's RSA modulus $\mathfrak{N}$). Finally, it publishes the initially empty update information $\boldsymbol{upd}$.

Join protocol. In the protocol Join, the group manager sends $(A_i, e_i, acc_{\mathrm{i}})$ with $acc_{\mathrm{i}} = acc$ to i in the fourth protocol step. Using the prime e_i (from the ACJT member's certificate) the group manager updates the accumulator acc in gpk to $acc^{e_i} \mod \mathfrak{N}$. Finally, the group manager adds an entry $(e_i, \mathtt{add})$ to $\boldsymbol{upd}$. At the end of the protocol i runs $\mathsf{AccVrfy}$ to check whether acc_i is a valid witness for e_i for the updated accumulator. (Note that accumulator acc that was valid before the protocol execution is given to i as a witness for its prime e_i with respect to the new accumulator.)

Revocation procedure. The new revocation algorithm Revoke of the ACJT scheme proceeds as follows. In order to revoke a member i the group manager runs $\mathsf{AccDel}((\mathfrak{p}, \mathfrak{q}), e_i, acc)$ to compute the updated accumulator acc in the gpk. Note that e_i is kept in $\boldsymbol{reg}[i]$. Finally, the group manager replaces the entry $(e_i, \mathtt{add})$ in $\boldsymbol{upd}$ with $(e_i, \mathtt{del})$.

Update procedure. The new update algorithm UpdM of the ACJT scheme proceeds as follows. An unrevoked group member i in possession of $(A_i, e_i, acc_{\mathrm{i}})$ obtains the recent changes from $\boldsymbol{upd}$ (consisting of entries of the form $(e_j, \mathtt{del})$ and $(e_j, \mathtt{add})$) and the current accumulator

acc from gpk. For each entry $(e_j, \mathtt{del})$ and $(e_j, \mathtt{add})$ member i executes the corresponding witness update algorithm $\mathsf{AccUpdW}$ that results in an updated witness acc_{i} for e_i with respect to the new accumulator acc.

Signature generation. The signing algorithm GSign takes as input (x_i, A_i, e_i), witness acc_i of member i, and some message $m \in \{0,1\}^*$. In addition to computing T_1, T_2, and T_3 as in Step 1 of the original ACJT signing procedure, the algorithm further picks random $r_1, r_2, r_3 \in \mathbb{Z}_{\lfloor \mathfrak{N}/4 \rfloor}$, and computes values

$$\mathfrak{C}_e = \mathfrak{g}^{e_i}\mathfrak{h}^{r_1}, \quad \mathfrak{C}_{acc_i} = acc_i\mathfrak{h}^{r_2}, \quad \mathfrak{C}_r = \mathfrak{g}^{r_2}\mathfrak{h}^{r_3}.$$

Additionally, the algorithm computes modified SoK signature S as follows:

$$\mathsf{SoK}\left[\ \alpha,\beta,\gamma,\delta,\varepsilon,\zeta,\phi,\psi,\eta,\varsigma \ : \begin{array}{c} a_0 = T_1^{\alpha}\ \frac{1}{a}^{\ \beta}\ \frac{1}{y}^{\ \gamma} \ \text{ and } \ 1 = T_2^{\alpha}\ \frac{1}{g}^{\ \gamma} \ \text{ and } \ T_2 = g^{\delta} \\ T_3 = g^{\alpha}h^{\varepsilon} \ \text{ and } \ \alpha \in \Lambda \ \text{ and } \ \beta \in \Gamma \\ \mathfrak{C}_e = \mathfrak{g}^{\alpha}\mathfrak{h}^{\varphi} \ \text{ and } \ \mathfrak{C}_r = \mathfrak{g}^{\psi}\mathfrak{h}^{\eta} \ \text{ and } \ acc = \mathfrak{C}_{acc}^{\alpha}\ \frac{1}{\mathfrak{h}}^{\ \varsigma} \\ 1 = \mathfrak{C}_r^{\alpha}\ \frac{1}{\mathfrak{g}}^{\ \varsigma}\ \frac{1}{\mathfrak{h}}^{\ \xi} \end{array}\ \right] m \ .$$

The output of the algorithm is the group signature $\sigma = (S, T_1, T_2, T_3, \mathfrak{C}_e, \mathfrak{C}_{acc_i}, \mathfrak{C}_r)$.

In the modified signature generation algorithm value T_3 acts as a commitment to the element e_i. Therefore, the modified ACJT scheme does not require any further groups beyond $QR(N)$ (for the original scheme) and $QR(\mathfrak{N})$ (for the accumulator). Note also that algorithms GVrfy, Open, and Judge do not need any further modifications since the accumulator public key is considered to be part of gpk. The use of the accumulator makes the original scheme, however, almost twice as less efficient. Nevertheless, it provides revocation handling without sacrificing security of the original schemes with regard to full-anonymity, insider traceability, and full non-frameability, despite that values e_i of all revoked and unrevoked members are now made public.

5.3. The Tsudik-Xu Scheme

Tsudik and Xu [178] introduced a variant of the dynamic version of the ACJT scheme where revocation is performed through a different dynamic accumulator that accumulates composite numbers of the form $e = e_1e_2$ where e_1 and e_2 are prime numbers of sufficient length known only to the user who generates e. At a high level, the idea of their signature scheme is to let each new group member choose own composite number e within the joining procedure and let the group manager accumulate e into the accumulator acc, which is part of the group public key gpk. The knowledge of the factorization of e is then used by the group member to generate group signatures. In the following we detail the algorithms and protocols of their scheme, which we refer to as the TX scheme.

5.3.1. The TX Scheme

The TX group signature scheme is dynamic and offers verifiable opening. It uses RSA setting and has several security parameters described in the following: Let $\epsilon > 1$ and $\,_c, \,_p \in \mathbb{N}$, where $\,_p$ determines the size of the modulus N and $\,_c$ denotes the output length of the used hash function. Let λ_1, λ_2 denote lengths and $\Lambda_1, \Lambda_2, \Gamma$ be three integral ranges satisfying the following conditions:

- $\,_p > \lambda_1 > \epsilon(\lambda_2 + \,_c) + 2$
- $\Lambda_1 =]2^{\lambda_1} - 2^{\lambda_2}, 2^{\lambda_1} + 2^{\lambda_2}[$
- $\Lambda_2 =]2^{\lambda_1}, 2^{\lambda_1+1} - 1[$
- $\Gamma =] - 2^{2\lambda_1+1}, 2^{2\lambda_1+1}[$

These conditions imply that $2^{\lambda_1} - 2^{\lambda_2} > 4$ and $(2^{\lambda_1} - 2^{\lambda_2})^3 > (2^{\lambda_1} + 2^{\lambda_2})(2^{\lambda_1+1} - 1)$. The TX scheme has an implicit accumulator that accumulates composite numbers from the following set:

$$\mathcal{X} = \{e_1 e_2 \mid e_1 \text{ is prime and } e_1 \in \Lambda_1 \text{ and } e_2 \text{ is prime and } e_2 \in \Lambda_2\},$$

which is seen as a subset of $\mathcal{X}' \subseteq \{5, \ldots, (2^{\lambda_1} - 2^{\lambda_2})^3 - 1\}$.

Furthermore, the TX scheme assumes trusted setup in that it relies on the existence of the public common reference string $(\mathfrak{N}, \mathfrak{g}, \mathfrak{h})$ such that $\mathfrak{N} = (2\mathfrak{p}'+1)(2\mathfrak{q}'+1)$ is a safe RSA modulus for some unknown prime numbers $\mathfrak{p}', \mathfrak{q}'$ of length $\,_p$ and $\mathfrak{g}, \mathfrak{h}$ are random elements from $QR(\mathfrak{N})$ with an unknown discrete logarithm of $\mathfrak{g}$ and $\mathfrak{h}$ to each other. Note that trusted generation of the common reference string is necessary to prevent framing attacks of a dishonest group manager.

In the following we specify the core algorithms and protocols of the TX scheme. Our description follows the specification from [178], except that we also provide a judgement procedure that has been omitted in the original work.

Key generation. The key generation algorithm GKg on input $\,_p$, $\,_c$, and the common reference string $(\mathfrak{N}, \mathfrak{g}, \mathfrak{h})$ performs the following steps:

1. Compute safe RSA modulus N as a product of two safe primes $p = 2p'+1$ and $q = 2q'+1$, where p' and q' are primes of length $\,_p$.
2. Pick random $x \in_R \mathbb{Z}^*_{p'q'}$ and $g \in_R QR(N)$. Compute $y = g^x \mod N$.
3. Pick random t such that t is prime and its length $|t| > \,_c$.
4. Pick random $acc_\emptyset \in_R QR(N)$ to initialize the accumulator.
5. Output $(gpk, \boldsymbol{upd}, gmsk, \boldsymbol{reg})$ such that:
   - group public key $gpk = (N, g, y, t, acc_\emptyset, \mathfrak{N}, \mathfrak{g}, \mathfrak{h})$
   - initially empty public update information $\boldsymbol{upd}$

- group manager's secret key $gmsk = (gpk, p', q', x)$
- initially empty secret registration list $\boldsymbol{reg}$.

It is assumed that key generation is performed in a trusted way to ensure the validity of the group public key gpk. That is the RSA modulus N is assumed to be indeed safe and elements g and $acc_\emptyset$ to be indeed chosen independently at random from $QR(N)$. Note that this assumption can be lifted using the costly NIZKPoK proofs from [59].

Join protocol. The join protocol Join is executed between the group manager GM with input $gmsk = (gpk, p', q', x)$ and a prospective member i with input $gpk = (N, g, y, t, acc_\emptyset, \mathfrak{N}, \mathfrak{g}, \mathfrak{h})$. It proceeds as follows:

1. Member i picks random prime numbers $e_{i,1} \in_R \Lambda_1$ and $e_{i,2} \in_R \Lambda_2$, computes $e_i = e_{i,1}e_{i,2}$ (in $\mathbb{Z}$), and sends e_i to GM.
2. GM proceeds if e_i is odd, e_i does not appear in the current update information $\boldsymbol{upd}$, and e_i is in the range $(2^{\lambda_1} - 2^{\lambda_2})2^{\lambda_1} < e_i < (2^{\lambda_1} + 2^{\lambda_2})(2^{\lambda_1+1} - 1)$. Let acc_i denote the current accumulator value acc. GM updates acc in gpk to $acc_{i+1} = acc_i^{e_i} \mod N$ and adds an entry $(e_i, \texttt{add})$ to the public update information $\boldsymbol{upd}$. Then, GM sends (acc_i, e_i) back to i.
3. Member i checks whether $acc_i^{e_i} = acc \mod N$ and if so stores $\boldsymbol{gsk}[i] = (gpk, e_i, acc_i, e_{i,1}, e_{i,2})$ as its secret signing key.
4. GM stores $\boldsymbol{reg}[i] = (acc_i, e_i, \texttt{trans}_i)$ where $\texttt{trans}_i$ is the communication transcript authenticated by i.

It is implicitly assumed that authentication of $\texttt{trans}_i$ performed by member i uniquely identifies i. Similar to the ACJT scheme [11], this can be realized using PKI in which candidate members possess certified private/public key pairs used to sign the transcript. For this reason the join protocol has to be executed over a secure channel.

Note that the secret signing key $\boldsymbol{gsk}[i]$ contains the factorization of e_i, i.e., prime numbers $e_{i,1}$ and $e_{i,2}$. These are assumed to be chosen by i in a trusted way such that factoring e_i remains hard. As discussed by Tsudik and Xu [178], since $e_{i,1} \in \Lambda_1$ and $e_{i,2} \in \Lambda_2$ the underlying factorization assumption should hold even if $(\lambda_1 - \lambda_2)$ higher-order bits of $e_{i,1}$ are potentially known to the adversary. The use of composite numbers e_i states the difference to the Camenisch-Lysyanskaya revocation mechanisms for the ACJT scheme [56] where e_i being a public prime number could not have been used to generate (unforgeable) group signatures alone and thus another secret element A_i was required. On the other hand, the dynamic accumulation of composite numbers e_i used in the TX scheme proceeds similarly to the Camenisch-Lysyanskaya accumulator. In particular, the addition, deletion, and update functions remain the same except that e_i being composite must also be odd. This is necessary for the later opening procedure discussed below. Yet, the use of composite numbers imposes another inconvenience: Given two accumulated values $e_i = e_{i,1}e_{i,2}$ and $e_j = e_{j,1}e_{j,2}$ it is possible to come up with another accumulated value, e.g. $e_{i,1}e_{j,2}$ or $e_{j,1}e_{i,2}$. This means that two members i and j can collude and produce a forged membership certificate. The TX scheme cannot prevent such abuses, which as a consequence results in a weaker form of traceability. As we shall see in the opening procedure, since the update information $\boldsymbol{upd}$, which is updated on each joining and revocation procedure, keeps track on all admitted group members and their accumulated composite values, the group manager in the TX scheme can identify at least one of the colluding members.

Revocation procedure. The revocation algorithm Revoke takes as input the group manager's secret key $gmsk = (gpk, p', q', x)$, the identity $i \in [1, n]$ of a member to be revoked, the registration entry $\boldsymbol{reg}[i] = (acc_i, e_i, \mathtt{trans}_i)$, and the current update information $\boldsymbol{upd}$. The group manager updates the accumulator acc in gpk to $acc^{{e_i}^{-1} \bmod 4p'q'} \bmod N$ and replaces the entry $(e_i, \mathtt{add})$ in $\boldsymbol{upd}$ with $(e_i, \mathtt{del})$.

Update procedure. The update algorithm UpdM takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, e_i, acc_i, e_{i,1}, e_{i,2})$ of an unrevoked group member i and the public update information $\boldsymbol{upd}$. It is assumed that i obtains the recent changes from $\boldsymbol{upd}$ consisting of entries of the form $(e_j, \mathtt{del})$ and $(e_j, \mathtt{add})$ and that gpk includes the current accumulator value acc. The algorithm proceeds as follows:

- For all new entries $(e_j, \mathtt{del})$: Find integers a, b such that $ae_i + b\prod_j e_j = 1$.
  Update acc_i to $acc_i^b \cdot acc^a \bmod N$.

- For all new entries $(e_j, \mathtt{add})$: Update acc_i to $acc_i^{\;\;j\, e_j} \bmod N$.

Note that since each accumulated composite value e_j is odd it holds with high probability that values e_i, $\prod_j e_j$, and $4p'q'$ are co-prime. Therefore, the existence of values a and b which can be found using the extended Euclidean algorithm (see, e.g. [171]) is guaranteed with said high probability.

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, e_i, acc_i, e_{i,1}, e_{i,2})$ of member i, where $gpk = (N, g, y, t, acc, \mathfrak{N}, \mathfrak{g}, \mathfrak{h})$, and a message $m \in \{0, 1\}^*$, and proceeds as follows:

1. Pick random $r_1 \in_R \mathbb{Z}_N^*$ and compute $T_1 = (1 + e_i N) r_1^t \bmod N^2$.

2. Pick random $r_2 \in_R \pm\{0, 1\}^{2\ell_p + \ell_c}$ and compute
$$T_2 = g^{r_2} \bmod N \quad \text{and} \quad T_3 = acc_i y^{r_2} \bmod N.$$

3. Pick random $r_3 \in_R \pm\{0, 1\}^{2\ell_p + \ell_c}$ and compute
$$T_4 = \mathfrak{g}^{r_3} \mathfrak{h}^{e_{i,1}} \bmod \mathfrak{N} \quad \text{and} \quad T_5 = T_4^{e_{i,2}} \bmod \mathfrak{N}.$$

4. Compute S as a signature of knowledge

$$\mathsf{SoK}\left[\begin{pmatrix}\alpha,\beta,\gamma,\delta,\\ \varepsilon,\zeta,\varphi,\psi\end{pmatrix} : \begin{array}{c} T_1 = (1+n)^{\alpha}\beta^{t} \mod N^2 \text{ and } T_2 = g^{\gamma} \mod N \\ acc = T_3^{\alpha}\ \frac{1}{y}^{\delta} \mod N \text{ and } 1 = T_2^{\alpha}\ \frac{1}{g}^{\delta} \mod N \\ T_5 = \mathfrak{g}^{\varepsilon}\mathfrak{h}^{\alpha} \mod \mathfrak{N} \text{ and } T_4 = \mathfrak{g}^{\zeta}\mathfrak{h}^{\varphi} \mod \mathfrak{N} \\ T_5 = T_4^{\psi} \mod \mathfrak{N} \text{ and } \alpha \in \Gamma \text{ and } \varphi \in \Lambda_1 \text{ and } \psi \in \Lambda_2 \end{array}\right] m \quad .$$

5. Output group signature $\sigma = (S, T_1, T_2, T_3, T_4, T_5)$.

In the above signature generation algorithm T_1 is an encryption of e_i under the public key (N, t) according to the public key encryption scheme of Paillier [156] and its modification by Catalano et al. [65]; (T_2, T_3) is ElGamal encryption of acc_i in the $QR(N)$ group under the public key y; T_4 is a variant of the Damgård-Fujisaki commitment [80] to $e_{i,1}$ under the public key $(\mathfrak{N}, \mathfrak{g}, \mathfrak{h})$, whereas T_5 can be seen as a corresponding commitment to e_i. The SoK signature S thus proves that the signer is in possession of (acc_i, e_i) satisfying the relationship $acc = acc_i^{e_i} \mod N$ and furthermore knows the factorization of e_i given by $e_{i,1}$ and $e_{i,2}$, all of which are in the appropriate integral ranges. Moreover, S proves that σ can be opened by the group manager, who can decrypt e_i.

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (N, g, y, t, acc, \mathfrak{N}, \mathfrak{g}, \mathfrak{h})$, a message m, and a candidate group signature σ, and proceeds as follows:

1. Parse σ as $(S, T_1, T_2, T_3, T_4, T_5)$.

2. If S is a valid SoK signature on message m and $T_5 = 1 \mod N$ and $T_4 = T_5^b \mod N$ for $b = \pm 1$ then output 1; otherwise output 0.

Opening procedure. The opening algorithm Open takes as input the group manager's secret key $gmsk = (gpk, p', q', x)$, message m, group signature σ, registration list $\boldsymbol{reg} = \{(acc_i, e_i, \mathtt{trans}_i)_{i\in[1,n]}\}$, and current update information $\boldsymbol{upd}$, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.

2. Compute $r_1 = (T_1 \mod N)^{1/t} \mod N$ and $e_i = \frac{(T_1 r_1^{-t} \mod N^2)-1}{N}$.

3. Compute $acc_i = \frac{T_3}{T_2^x} \mod N$.

4. If $(e_i, \mathtt{add}) \in \boldsymbol{upd}$ then:
   - Find i for which $\boldsymbol{reg}[i] = (acc_i, e_i, \mathtt{trans}_i)$.
   - Compute J as $\mathsf{NIZKPoK} \quad x \quad : \quad y = g^x \text{ and } \frac{T_3}{acc_i} = T_2^x \quad .$
   - Output (i, τ), where $\tau = (r_1, J, \mathtt{trans}_i)$.

5. Else if $(e_i, \mathtt{add}) \notin \boldsymbol{upd}$ then:

- Find an entry $(e_j, \texttt{add}) \in \boldsymbol{upd}$ for which $e_j > gcd(e_i, e_j) > 1$.
- Find j for which $\boldsymbol{reg}[j] = (acc_j, e_j, \texttt{trans}_j)$.
- Output (j, τ), where $\tau = (r_1, e_i, \texttt{trans}_j)$.

The above opening procedure distinguishes between two cases depending on the occurrence of decrypted e_i in the update information $\boldsymbol{upd}$: (1) If e_i belongs to some previously admitted unrevoked group member i then the group manager can identify that member and provide the corresponding proof τ, which includes value r_1 that was used as randomness in the computation of T_1, a NIZKPoK proof for the valid decryption of acc_i from the ElGamal ciphertext (T_2, T_3), and transcript $\texttt{trans}_i$ authenticated by member i which contains (acc_i, e_i); alternatively, (2) e_i is a value that is accumulated in acc but does not belong to any group member. In this case e_i must have been computed as a product of at most two (due to the integral checks) factors of other legitimately accumulated values. That is e_i must divide the product $\prod_j e_j$ where each e_j corresponds to some entry $(e_j, \texttt{add}) \in \boldsymbol{upd}$. The opening procedure thus finds e_j for which one of its factors ($e_{j,1}$ or $e_{j,2}$) was used to compute e_i and provides a proof τ of this fact. This proof includes again the randomness r_1 used to compute the ciphertext T_1, the decrypted value e_i, and the transcript $\texttt{trans}_j$, which contains, in particular, e_j authenticated by member j. Tsudik and Xu [178] suggest that in this case member j should also be revoked.

Judgement procedure. The judgement algorithm Judge takes as input the group public key $gpk = (N, g, y, t, acc, \mathfrak{N}, \mathfrak{g}, \mathfrak{h})$, message m, signature σ, identity i, and proof τ, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.
2. Parse σ as $(S, T_1, T_2, T_3, T_4, T_5)$.
3. If τ contains $(r_1, J, \texttt{trans}_i)$ then if all of the following holds then output 1; otherwise output 0:
   - $T_1 = (1 + e_i N) r_1^t \mod N^2$ and J is a valid NIZKPoK proof (both are checked using (acc_i, e_i) from $\texttt{trans}_i$)
   - $\texttt{trans}_i$ is authenticated by i.
4. Else if τ contains $(r_1, e_j, \texttt{trans}_i)$ then if all of the following holds then output 1; otherwise output 0:
   - $T_1 = (1 + e_j N) r_1^t \mod N^2$
   - $e_i > gcd(e_j, e_i) > 1$ using e_i from $\texttt{trans}_i$
   - $\texttt{trans}_i$ is authenticated by i.

The judgement procedure depends on whether the opening procedure identified the actual signer or some member whose secret key was partially involved in the generation of the signature. In both cases the algorithm checks that decryption of e_i resp. e_j from T_1 was performed correctly by recomputing the ciphertext T_1. This check holds since the encryption operation from [65] is used as a commitment to the encrypted value e_i resp. e_j that can be verified using the randomness r_1 used in the generation of the ciphertext (commitment) T_1. Then, depending on the nature of the proof τ, either the validity of the NIZKPoK proof J is checked or the relationship $e_i > gcd(e_j, e_i) > 1$. In both cases the actual identity of member i is publicly verified using the authenticated transcript $\mathtt{trans}_i$.

5.3.2. Security of the TX Scheme

The original security proofs for the TX scheme addressed several building blocks of the scheme and proved separate requirements of unforgeability, anonymity, unlinkability, exculpability, and coalition-resistance. In particular, Tsudik and Xu [178] gave a separate construction of the dynamic accumulator for composite numbers and proved its security under the Strong RSA assumption. Furthermore, they proved that the interactive version of the protocol underlying the SoK signature S in the algorithm GSign has an (honest-verifier) statistical ZK property whereas its soundness and PoK properties hold under the assumptions that the encryption schemes by Catalano et al. [65] used to compute T_1 and ElGamal encryption used to compute T_2 and T_3 offer IND-CPA security, while the commitment scheme by Damgård and Fujisaki [80] offers computational binding, and factoring of composite numbers e_i is computationally hard, even if a small fraction of higher-order bits of one of the primes is known. Observe that: (1) IND-CPA security of the ElGamal encryption scheme in $QR(N)$ groups holds under the respective DDH assumption (see Definition 3.4); (2) IND-CPA security of the Catalano et al. encryption scheme holds under the Decision Small Residuosity (DSR) assumption (see [65] for further details); (3) Damgård-Fujisaki commitment schemes offer computational binding under a slightly more general Strong RSA assumption (see [80] for further details).

In the following we intuitively revisit the security of the TX scheme in the light of our definitions for anonymity, traceability, and non-frameability from Sections 2.2 and 2.3.

Anonymity. The TX scheme does not satisfy the full anonymity notion from Definition 2.12 for the following reason: If the adversary knows all secret signing keys, which include factorizations $(e_{i,1}, e_{i,2})$ of e_i then for the challenge group signature σ^*, containing values T_4^* and T_5^*, the following test whether $T_5^* = {T^*}_4^{e_{i,2}} \mod \mathfrak{N}$, performed for all $e_{i,2}$ would immediate reveal the signer. In contrast, the TX scheme may still satisfy the notion of insider anonymity (which assumes that at least two group members remain uncorrupted) in the Random Oracle Model, assuming the IND-CPA security of ElGamal encryption in $QR(N)$, the IND-CPA security of the encryption scheme by Catalano et al. [65], and hardness of factoring e_i. Note also that knowledge of secret signing keys $\boldsymbol{gsk}[i]$ of all other group members would not help the adversary $\mathcal{A}$ in distinguishing, whether the challenge group signature σ^* has been produced by uncorrupted signers i_0 or signer i_1, due to the statistical ZK property of the corresponding SoK signature.

Traceability. Since TX scheme is dynamic and cannot provide full traceability, as mentioned in Section 2.2.4. The TX scheme does, however, seem to offer insider traceability from Definition 2.18, but only in a weaker sense. Since the group manager remains honest and each joining member i authenticates the transcript $\mathtt{trans}_i$ the adversary $\mathcal{A}$ has to come up with the SoK signature S^* which is valid for $(T_1^*, T_2^*, T_3^*, T_4^*, T_5^*)$ and message m^*, while the algorithm Open executed by the group manager on σ^* outputs $i = 0$ or some pair (i^*, τ^*) that is rejected by the algorithm Judge. The soundness property of the SoK signature ensures that if σ^* is valid then components T_1^*, T_2^*, and T_3^* encrypt (e_i, acc_i) for which $acc = acc_i^{e_i} \mod N$. The security of the dynamic accumulator guarantees furthermore that Open, in addition to the proof τ, will output either identity $i^* = i$ whom decrypted e_i belongs to or identity $i^* = j$ for which one of the factors of e_j has been used in the computation of e_i. In any case, the probability that Open outputs $i^* = 0$ would remain negligible. Nonetheless, the group manager may not be able to identify all (colluding) members that were involved in the generation of σ^*. The probability that Judge algorithm rejects the output pair (i^*, τ^*) seems still negligible due to the following assumptions: In case where $i^* = i$ is the signer of σ^*, the acceptance of τ^* is guaranteed primarily by the binding property of the encryption scheme from [65], when viewed as a commitment scheme, and the soundness property of the NIZKPoK proof J. Whereas, if $i^* = j$ is the identity of one of the colluding members j, the acceptance of τ^* is guaranteed by the security of the dynamic accumulator, in addition to the binding property of the encryption scheme from [65], when viewed as a commitment scheme.

Non-Frameability. The TX scheme seems to satisfy the notion of full non-frameability from Definition 2.21. The reason is that Judge algorithm first ensures that (m^*, σ^*) can be successfully verified. The PoK property of the SoK signature S^* (which is part of σ^*) guarantees that σ^* has been produced with knowledge of factors $e_{i^*,1}$ and $e_{i^*,2}$ of the corresponding composite value e_{i^*}. Although the adversary $\mathcal{A}$ may learn e_{i^*} by acting as the group manager in the join protocol, it is hard for $\mathcal{A}$ to actually obtain these factors due to the hardness of the factorization assumption. Note that $\mathcal{A}$ could still generate (m^*, σ^*) on behalf of some corrupted group member or a coalition of such members and attempt to provide a proof τ^* that this pair opens to some honest member i^*. This is, however, prevented by the assumption that each transcript uniquely identifies the group member (who authenticated this transcript before) and the soundness property of the NIZKPoK proof J.

5.4. The Camenisch-Groth Scheme

In this section we present the group signature scheme and its extensions proposed by Camenisch and Groth [49]. This scheme, which we refer to as CG, basically provides two different types. The basic CG scheme is static while the offered extensions are fully dynamic, providing a joining and a revocation algorithm. So the basic scheme belongs to the class of schemes that were defined in Section 2.1 while the extended scheme belongs to the class of schemes that were defined in Section 2.2. The CG scheme is a further development of the previously described ACJT scheme (Section 5.1). It is significantly faster than the ACJT scheme also because it has a more efficient joining protocol. In the following we provide the description of the basic CG scheme and its two extensions.

5.4.1. The Basic CG Scheme

The basic CG group signature scheme is static. It uses RSA setting in the random oracle model and a DL assumption. Let $_c$, $_e$, $_s$, $_E$, $_Q$, $_N$, $_P$ denote lengths derived from the security parameter κ satisfying the following conditions:

- $_c$ denotes the length of the output of the used hash function.
- For every integer a, an integer r of the length $|a| + {}_s$ can be chosen randomly, so that $a+r$ and r are statistically indistinguishable. That is, for all distinguishers $\mathcal{A}$ the probability that $\mathcal{A}$ correctly distiguishes between r and $r+a$ is smaller than some fixed $\epsilon > 0$.
- $_e$ large enough to assign different numbers to all members and to make all E_i constructed during key generation prime.
- ${}_c + {}_e + {}_s + 1 < {}_Q$
- ${}_c + {}_Q + {}_s + 1 < {}_E < {}_N/2$

The relations between the lengths are chosen according to the Camenisch-Lysyanskaya signature scheme [55, 133]. In the following we specify the core algorithms and protocols of the basic CG scheme. Our description follows the specification from [49].

Key generation. The key generation algorithm GKg on input 1^κ and n where $n \in \mathbb{N}$ denotes the total number of group members performs the following steps:

1. Compute safe RSA modulus N using the $\mathsf{RSAGen}(1^{\ell_N})$ algorithm as defined in Section 3.2.1.
2. Choose random elements $a, g, h \in_R QR(N)$.
3. Choose random primes Q, P of length $_Q$, $_P$ with $Q|P-1$.
4. Pick $X_G, X_H \in_R \mathbb{Z}_Q$ with $G = F^{X_G} \bmod P$ and $H = F^{X_H} \bmod P$ where $F \in_R \mathbb{Z}_P^*$ of order Q.
5. Select $x_1, \ldots, x_n \in_R \mathbb{Z}_Q$ and $r_1, \ldots, r_n \in_R \mathbb{Z}_N$.
6. Choose $e_1, \ldots, e_n \in_R \{0,1\}^{\ell_e}$ such that $E_1 = 2^{\ell_E} + e_1, \ldots, E_n = 2^{\ell_E} + e_n$ are primes.
7. Compute $y_1, \ldots, y_n$ with $y_1^{E_1} = ag^{x_1}h^{r_1} \bmod N, \ldots, y_n^{E_n} = ag^{x_n}h^{r_n} \bmod N$.
8. Compute $Y_1 = G^{x_1} \bmod P, \ldots, Y_n = G^{x_n} \bmod P$.
9. Output $(gpk, gmsk, \boldsymbol{gsk})$ such that:
   - group public key $gpk = (N, a, g, h, F, G, H, P, Q)$
   - group manager's secret key $gmsk = (gpk, X_G, Y_1, \ldots, Y_k)$
   - n-vector of secret signing keys of member i: $\boldsymbol{gsk}[i] = (gpk, x_i, y_i, e_i, r_i)$

It is assumed that key generation is performed in a trusted way. In particular, this means that the RSA modulus N is indeed safe and that elements a, g, h are chosen independently at random from $QR(N)$. This assumption is necessary to ensure trust into the group public key *gpk*. The CG scheme is based on the Camenisch-Lysyanskaya signature scheme [55, 133] and the discrete logarithm in $\langle F \rangle$. The members secret keys contain in fact an ordinary Camenisch-Lysyanskaya signature (y_i, e_i, r_i) on the member's secret x_i. The values a, g, h, x_i also come from the Camenisch-Lysyanskaya signature scheme. The construction of E_i ensures that $2^{\ell_E} > E_i > 2^{\ell_E - 1}$. But instead of storing the large exponent E_i, it is sufficient to store the smaller value e_i. Since the discrete logarithm X_G of G is included into *gmsk* the group manager is able to decrypt Y_i and identify the signer. The discrete logarithm X_H of H is of no interest for the opening procedure as only G is used to generate Y_i. In contrast, H is needed to simulate group signatures on behalf of a member i, this ability is used to prove the anonymity property of the scheme.

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, x_i, y_i, e_i, r_i)$ of member i, where $gpk = (N, a, g, h, F, G, H, P, Q)$ and a message $m \in \{0,1\}^*$ and proceeds as follows:

1. Pick random $r \in_R \{0,1\}^{\ell_N/2}$ and $R \in_R \mathbb{Z}_Q$ and compute
$$T_1 = h^r y_i \bmod N,\ T_2 = F^R \bmod P,\ T_3 = G^{R+x_i} = G^R Y_i \bmod P,\ T_4 = H^{R+e_i} \bmod P.$$
2. Compute S as a signature of knowledge
$$\mathsf{SoK}\left[\ \xi, \rho, \varepsilon, \tau \quad : \quad \begin{array}{c} a = T_1^{2^{\ell_E}+\varepsilon} g^{-\xi} h^{\rho} \bmod N \ \text{ and } \ T_2 = F^{\tau} \bmod P \\ T_3 = G^{\tau+\xi} \bmod P \ \text{ and } \ T_4 = H^{\tau+\varepsilon} \bmod P \\ \varepsilon \in \{-2^{\ell_e+\ell_c+\ell_s}, +2^{\ell_e+\ell_c+\ell_s}\} \\ \xi \in \{-2^{\ell_Q+\ell_c+\ell_s}, +2^{\ell_Q+\ell_c+\ell_s}\} \end{array}\ \right] m \quad .$$
3. Output group signature $\sigma = (S, T_1, T_2, T_3, T_4)$.

The pair (T_2, T_3) is an ElGamal encryption of Y_i and the SoK signature proves additionally that the signer knows the corresponding x_i. Using the values T_1 and T_4 the SoK signature proves further that the signer is in possession of a valid signature (y_i, e_i, r_i).

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (N, a, g, h, F, G, H, P, Q)$, a message m and a candidate group signature σ and proceeds as follows:

1. Parse σ as (S, T_1, T_2, T_3, T_4).
2. If S is a valid SoK signature on message m then output 1; otherwise output 0.

Opening procedure. The opening algorithm Open takes as input the group manager's secret key $gmsk = (gpk, X_G, Y_1, \ldots, Y_k)$, the registration information $\boldsymbol{reg}$, a message m and a group signature σ, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.
2. Parse σ as (S, T_1, T_2, T_3, T_4).
3. Compute $Y_i = T_3/T_2^{X_G} \bmod P$.
4. If there exists i with $\boldsymbol{reg}[i]$ contains Y_i, then output i; otherwise output 0.

This ensures that only the group manager, knowing the Y_i and X_G is able to open a signature.

5.4.2. Security of the Basic CG Scheme

Camenisch and Groth proved that their basic group signature scheme has full traceability and full anonymity according to the definitions of Bellare et al. [22]. In the following we discuss its security in the light of our notions for anonymity, traceability and non-frameability from Section 2.1.

Anonymity. The basic CG scheme seems to offer full anonymity from Definition 2.4 in the Random Oracle Model under the DL assumption from Definition 3.5 in $QR(N)$ groups and under the strong RSA assumption in $\mathbb{Z}_N^*$ groups from Definition 3.3. The knowledge of secret signing keys $\boldsymbol{gsk}[i]$ of all group members does not help the adversary $\mathcal{A}$ in distinguishing, whether the challenge signature $\sigma = (S, T_1, T_2, T_3, T_4)$ has been produced by signer i_0 or signer i_1, since the corresponding SoK has statistical ZK property, i.e., it does not reveal any information about $\boldsymbol{gsk}[i_b]$ and even if $\boldsymbol{gsk}[i_0]$ and $\boldsymbol{gsk}[i_1]$ are known to $\mathcal{A}$ (which full anonymity implicitly assumes) then distinguishing whether these values have been used to compute T_1, T_2, T_3 and T_4 is hard under the DL assumption in $QR(N)$ groups. From this it follows that the adversary is not able to distinguish whether the signature has been produced by member 0 or member 1.

Traceability. The basic CG scheme seems to offer full traceability guarantees from Definition 2.6 in the Random Oracle Model under the DL assumption in $QR(N)$ groups and under the strong RSA assumption in $\mathbb{Z}_N^*$ groups. In particular, an adversary on input $(gpk, gmsk, \boldsymbol{gsk})$ would not be able to generate a valid message-signature pair (m^*, σ^*) which does not open to some member of the group. At first we notice that due to the zero knowledge property of the proof of knowledge of x_i, the signing oracle reveals nothing about the corresponding witness x_i. The knowledge of the group manager's secret key $Y_i = G^{x_i} \bmod P$ allows furthermore perfect simulation of group signatures without possession of x_i. Due to the DL assumption, it seems infeasible for the adversary $\mathcal{A}$ that does not know x_i to come up with a valid group signature σ^* on some new message m^* where Y_i would be encrypted in σ^*. Finally, we observe that the existential unforgeability of the used Camenisch-Lysyanskaya signature scheme implies that a valid group signature contains G^{x_i} within its components (T_2, T_3).

Non-Frameability. The basic CG scheme seems to satisfy the requirement of full non-frameability from Definition 2.8 in the Random Oracle Model under the DL assumption in

$QR(N)$ groups from Definition 3.5 and under the strong RSA assumption in $\mathbb{Z}_N^*$ groups from Definition 3.3. Using the same arguments as for the traceability it would be possible to show that any adversary with input $(gpk, gmsk)$ and with limited access to the oracles $\mathsf{Corrupt}(\cdot)$ and $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot)$ will not succeed in outputting a valid message-signature pair (m^*, σ^*) that would point to some member i of the group.

5.4.3. Dynamic Extensions of the CG Scheme

Camenisch and Groth [49] proposed several extensions to their basic scheme aiming to introduce support for dynamic behavior and revocation. These extensions offer a protocol for the admission of new group members and algorithms for their revocation. The first extension, detailed in the following, prevents revoked members from issuing valid group signatures under the current group public key. This is realized using an approach, which resembles the accumulator for prime numbers from [56] with some modifications making the update of the accumulator upon the admission of new group members obsolete. The second extension, which we briefly discuss in the next section, can be further used to prevent signers from claiming validity of their group signatures under older public keys. It is realized using an approach that is similar to revocation lists.

Key generation. The key generation algorithm GKg on input 1^κ updates the original key generation algorithm such that:

1. The number of group members is set to zero, i.e., $\boldsymbol{gsk}$ is empty.
2. An additional random element $w \in_R QR(N)$ is chosen and added to the group public key.
3. The element g is computed as $g = h^\xi \bmod N$ with $\xi \in_R \mathbb{Z}_Q$.
4. Output $(gpk, gmsk, \boldsymbol{reg})$ such that:
   - group public key $gpk = (N, a, g, h, F, G, H, P, Q, w)$
   - group manager's secret key $gmsk = (gpk, p, q, X_G)$ (where p, q are the factors of N returned by $\mathsf{RSAGen}(1^{\ell_N})$)
   - registration list $\boldsymbol{reg}$ is initially empty.

It is assumed that key generation is performed in a trusted way. Otherwise a zero knowledge proof of knowledge of the value ξ can assure the correctness of the group public key. The additional value of w allows the user to chose his secret x_i on his own and allows him to prove knowledge of a root of w in the signing process.

Join protocol. The join protocol Join is executed between the group manager with input $gmsk = (gpk, p, q, X_G)$ and a prospective member i with input $gpk = (N, a, g, h, F, G, H, P, Q, w)$. It proceeds as follows:

1. Member i picks random $x_i \in_R \mathbb{Z}_Q$ and $r' \in_R \mathbb{Z}_N$, computes $Y_i = G^{x_i} \bmod P$ and $C_i = g^{x_i} h^{r'_i} \bmod N$ and sends Y_i, C_i to the group manager together with the proof
$$\mathsf{NIZKPoK} \quad \alpha, \beta \quad : \quad Y_i = g^{\alpha} \bmod P \text{ and } C_i = g^{\alpha} h^{\beta} \bmod N \quad .$$
2. The group manager proceeds if the proof above was correct.
   - The group manager picks $e_i \in_R \{0,1\}^{\ell_e}$ such that $E_i = 2^{\ell_E} + e_i$ is prime.
   - The group manager computes $w_i = w^{E_i^{-1}} \bmod N$.
   - The group manager selects $r''_i \in_R \mathbb{Z}_{\ell_e}$ and computes $y_i = (aC_i h^{r''_i})^{E_i^{-1}} \bmod N$ which requires the knowledge of *gmsk*, i.e., the factorization p, q of N.
   - The group manager sets $\boldsymbol{reg}[i] = (w_i, Y_i, E_i)$ and sends (w_i, y_i, E_i, r''_i) back to member i.
3. Member i stores $\boldsymbol{gsk}[i] = (gpk, w_i, x_i, r_i, y_i, e_i)$ with $r_i = r'_i + r''_i$ and $e_i = E_i - 2^{\ell_E}$ as its secret signing key.

This protocol is very similar to the ACJT joining protocol, but takes only two rounds. As in the static CG scheme the member gets a Camenisch-Lysyanskaya signature of the group manager that proves his membership of the group. The additional w_i allows members to dynamically join the group and the group manager to revoke their memberships.

Revocation procedure. The *revocation algorithm* Revoke takes as input the group manager's secret key $gmsk = (gpk, p, q, X_G)$, the identity $i \in [1, n]$ of a member to be revoked, the registration entry $\boldsymbol{reg}[i] = (w_i, Y_i, E_i)$ and the current update information $\boldsymbol{upd}$ and proceeds as follows:

1. Publish $(i, E_i, \texttt{del})$ in $\boldsymbol{upd}$.
2. Replace w in *gpk* with $w^{E_i^{-1}} \bmod N$, whereby the inverse E_i^{-1} is computed using the factors p and q from *gmsk*.

Observe that each revocation event results in the growth of the published update information $\boldsymbol{upd}$.

Update procedure. The randomized *update algorithm* UpdM takes as input the current secret signing key $\boldsymbol{gsk}[i]$ and at least the update information $\boldsymbol{upd}$, and results in a modification of $\boldsymbol{gsk}[i] = (gpk, w_i, x_i, r_i, y_i, e_i)$. For each new $(j, E_j, \texttt{del})$ in $\boldsymbol{upd}$ proceed as follows:

1. Find α, β such that $\alpha E_i + \beta E_j = 1$.
2. Replace w_i in $\boldsymbol{gsk}[i]$ with $w_i^{\beta} w^{\alpha} \bmod N$.

Notice that the update procedure provides each unrevoked group member i with a new value w_i, which corresponds to the E_i-th root of the current value w published in *gpk*.

Signature generation. The signing algorithm updates the original signing algorithm such that:

1. T_1 is computed as $T_1 = h^r y_i w_i \bmod N$.

2. In the signature of knowledge S value $aw = T_1^{2^{\ell_E}+\varepsilon} g^{-\xi} h^{\rho} \bmod N$ is used instead of a.

3. The output group signature is $\sigma = (S, T_1, T_2, T_3, T_4)$.

This change in the signature generation ensures that only members knowing a root of the current value w from the group public key are able to sign. In particular, revoked member do no longer have knowledge of the appropriate root and thus cannot sign.

Signature verification. The procedure is the same as in the static CG scheme.

Opening procedure. The procedure is the same as in the static CG scheme.

In the following we briefly revisit the security of the dynamic CG scheme in the light of our definitions for anonymity, traceability and non-frameability from Section 2.2. The dynamic extension of the CG scheme satisfies the full anonymity notion from Definition 2.12 in the Random Oracle Model as the ability to adaptively joining new members to the group is of no advantage for the adversary. The member's secret x_i is hidden through the zero knowledge proof of knowledge during the join protocol. The dynamic CG scheme cannot satisfy full traceability as mentioned in Section 2.2.5. But since Y_i stays the same in the dynamic extension it still offers insider traceability from Definition 2.5. The full non-frameability from Definition 2.15 follows directly from the full non-frameability of the static CG scheme because member i is still the only one who knows the discrete logarithm of Y_i.

Extension with Full Revocation

The second dynamic extension of the CG signature scheme offers support for full revocation. This mechanism aims to eliminate the problem with the previous revocation process where revoked members could still claim validity of the group signatures under older public keys. This is prevented by extending the accumulator-based approach from the previous section with an additional token s_i that is generated by the candidate member during the joining procedure. In order to revoke the signer the *full revocation* algorithm, executed by the group manager, publishes s_i in the public update information ***upd***. This revocation method thus combines the use of the accumulator with a revocation list. The main technical idea behind this new approach is to let the group manager sign s_i as part of the issued membership credential such that the group member by committing to s_i during the signature generation can still prove the possession of a valid individual secret signing key and preserve unlinkability of his group signatures. Once the token s_i is published all signatures issued by the member i become linkable and are no longer treated as valid. The verification algorithm performs revocation checks with all so far published revocation tokens to detect whether the signer is revoked. This introduces linear costs to the verification procedure.

5.5. The Kiayias-Yung Scheme

In this section we present the dynamic group signature scheme proposed by Kiayias and Yung [119, 121]. This scheme, which we refer to as KY, belongs to the class of schemes that were defined in Section 2.3 since it admits an explicit opening proof as outlined by the authors. Furthermore, the KY scheme can be modified to a scheme with distributed authorities defined in Section 2.4. The KY scheme can be seen as a modification of the ACJT scheme (cf. Section 5.1).

5.5.1. The KY Scheme

The KY group signature scheme is dynamic and works in the RSA setting. Let $\ell, \mu, k, \varepsilon$ denote integers satisfying the following conditions:

- $S(2^\ell, 2^\mu) = \{2^\ell - 2^\mu + 1, \ldots, 2^\ell + 2^\mu - 1\} \subseteq \{1, \ldots, p'q'\}$ where p', q' are the same as used in the RSAGen algorithm, defined in Section 3.2.1.
- $\varepsilon > 1$.
- $S(2^\ell, 2^{\varepsilon(\mu+k)+2}) \subseteq \{5, \ldots, \min\{p', q'\} - 1\}$.

In the following we specify the core algorithms and protocols of the KY scheme. Our description follows the specification from [119, 121].

Key generation. The key generation algorithm GKg on input 1^κ performs the following steps:

1. Use RSAGen to generate a tuple (N, p, q, p', q') where (N, p, q) is the output of the RSAGen algorithm and p', q' are the two primes used during the RSAGen algorithm.
2. Chose an element g as quadratic residue modulo N, that is generator of $QR(N)$.
3. Compute a generator $\mathfrak{a} = \rho(g_1^{(q')^{-1}}, g_2^{(p')^{-1}})$ of $QR(N)$ using the Chinese remaindering mapping ρ from $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$ to $\mathbb{Z}_N^*$ and g_1, g_2 as non-trivial quadratic residues modulo p and modulo q, respectively. That is $g_1, g_2 \notin \{0, 1\}$. Note that ρ preserves the quadratic residuosity, i.e., $\forall x \in QR(p)\ \forall y \in QR(q)\ \exists z \in QR(N)\ \rho(x, y) = z$.
4. Select $x, \hat{x} \in_R \mathbb{Z}_{p'q'}^*$ and $a_0, a, h \in_R QR(N)$.
5. Compute $y = g^x$, $\hat{y} = g^{\hat{x}}$.
6. Output $(gpk, gmsk, \boldsymbol{reg})$ such that:
    - group public key $gpk = (N, a_0, a, g, h, y, \hat{y})$
    - group manager's secret key $gmsk = (gpk, p, q, x, \hat{x})$
    - registration list $\boldsymbol{reg}$ is initially empty.

Join protocol. The join protocol Join is executed between the group manager with input $gmsk = (gpk, p, q, x, \hat{x})$, a prospective member i with input $gpk = (N, a_0, a, g, h, y, \hat{y})$ and a trusted third party T with input $gpk = (N, a_0, a, g, h, y, \hat{y})$. It proceeds as follows:

1. Member i asks T to start the joining process.
2. T selects a random $\mathfrak{x}_i \in_R \lfloor N/4 \rfloor$ and computes $C_i = a^{\mathfrak{x}_i} \bmod N$.
3. T sends C_i to the group manager and $\mathfrak{x}_i$ to the member.
4. The group manager selects a random prime $e_i \in_R S(2^\ell, 2^\mu) - \{p', q'\} = \{2^\ell - 2^\mu + 1, \ldots, 2^\ell + 2^\mu - 1\} - \{p', q'\}$ and computes $A_i = (a_0 C_i)^{1/e_i} \bmod N$.
5. The group manager stores $(A_i, e_i, C_i, \mathtt{trans}_i)$ to $\boldsymbol{reg}[i]$ where $trans_i$ is the communication transcript and sends (A_i, e_i) back to member i.
6. Member i sets $\boldsymbol{gsk}[i] = (gpk, A_i, e_i, \mathfrak{x}_i)$

Note that the join protocol of the KY scheme is executed with the assistance of some third trusted party T. Furthermore, Kiayias and Yung [119, 121] distinguish between the member's certificate and the member's secret. The member's secret $\mathfrak{x}$ is the private key of an ElGamal key pair. The group manager generates the certificate (A_i, e_i) over the public value a_0, a that attests member's i membership in the group. The communication transcripts $\mathtt{trans}_i$ that the issuer stores in $\boldsymbol{reg}$ are assumed to be authenticated by member i in a way that any publication of $\mathtt{trans}_i$ uniquely identifies i as participant of that join protocol.

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, A_i, e_i, \mathfrak{x}_i)$ of member i, with $gpk = (N, a_0, a, g, h, y, \hat{y})$ and a message $m \in \{0,1\}^*$ and proceeds as follows:

1. Pick random $r, \hat{r} \in_R \lfloor N/4 \rfloor$ and compute

   $$T_1 = A_i y^r \bmod N,\ T_2 = g^r \bmod N,\ T_3 = A_i \hat{y}^{\hat{r}} \bmod N,\ T_4 = g^{\hat{r}} \bmod N,\ T_5 = g^{e_i} h^r \bmod N$$

2. Compute S as a signature of knowledge of $\alpha, \beta, \gamma, \delta, \epsilon, \zeta$

   $$\mathsf{SoK}\left[\ \alpha, \beta, \gamma, \delta, \epsilon, \zeta \quad : \quad \begin{array}{c} T_1/T_3 = y^\alpha/\hat{y}^\beta \bmod N \text{ and } T_2 = g^\alpha \bmod N \\ T_3 = g^\beta \bmod N \text{ and } T_5 = g^\gamma h^\alpha \bmod N \\ T_2^\gamma = g^\epsilon \bmod N \text{ and } T_1^\gamma = a_0 a^\delta y^\epsilon \bmod N \\ T_5 = g(g^2)^\zeta h^\alpha \bmod N \end{array}\ \right] m \quad .$$

3. Output group signature $\sigma = (S, T_1, T_2, T_3, T_4, T_5)$.

The pairs (T_1, T_2) and (T_3, T_4) form ElGamal ciphertexts of A_i in the $QR(N)$ group. The SoK signature S ensures that those pairs do, actually, encrypt the same value A_i. Furthermore, S proves that the signer is a member of the group by proving knowledge of a valid certificate (A_i, e_i) issued by the group manager.

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (N, a_0, a, g, h, y, \hat{y})$, a message m and a candidate group signature σ, and proceeds as follows:

1. Parse σ as $(S, T_1, T_2, T_3, T_4, T_5)$.
2. If S is a valid SoK signature on message m then output 1; otherwise output 0.

Opening procedure. The opening algorithm Open takes as input the group manager's secret key $gmsk = (gpk, p, q, x, \hat{x})$, a message m and a group signature σ, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.
2. Parse σ as $(S, T_1, T_2, T_3, T_4, T_5)$.
3. Get $A_i = (T_1 T_2^{-x})^2 \bmod N$ using ElGamal.
4. Find i such that $\boldsymbol{reg}[i] = (A_i, e_i)$ if no such entry exists output 0.
5. Compute J as
$$\mathsf{NIZKPoK}\quad \omega \quad : \quad T_1^2 = T_2^{2\omega} A_i \bmod N \;\text{ and }\; y = g^{\omega} \bmod N \quad .$$
6. Output (i, τ) with $\tau = (\langle A_i, e_i, C_i, \mathtt{trans}_i \rangle, J)$.

To prevent the group manager from outputting arbitrary identities i the opening procedure, additionally, proves that the ciphertext (T_1, T_2) was decrypted correctly through the corresponding NIZPoK proof τ.

Judgement procedure. The judgement algorithm Judge takes as input the group public key $gpk = (N, a_0, a, g, h, y, \hat{y})$, a message m, a signature σ, an identity i, and a proof τ, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.
2. Parse τ as $(\langle A_i, e_i, C_i, \mathtt{trans}_i \rangle, J)$.
3. If all of the following holds then output 1; otherwise output 0:
   - J is a valid NIZKPoK proof.
   - A_i is included in $\mathtt{trans}_i$.
   - $\mathtt{trans}_i$ is authenticated by i
   - (A_i, e_i) satisfies $A_i^{e_i} = a_0 C_i \bmod N$.

The validity of the NIZKPoK proof J ensures, that the opener correctly decrypted the claimed value A_i from the ElGamal ciphertext (T_1, T_2). The proof that A_i identifies i stems from the authenticity of the transcript $\mathtt{trans}_i$ in combination with the validity check of the disclosed certificate (A_i, e_i).

5.5.2. Security of the KY Scheme

In the following we discuss security of the KY scheme considering our notions for anonymity, traceability and non-frameability from Section 2.3.

Anonymity. The KY scheme seems to offer full anonymity from Definition 2.12. This seems to hold in the Random Oracle Model under the DDH assumption from Definition 3.6. Note that this assumption holds even if the factorization of the RSA modulus N becomes public. It seems possible to transform any adversary against the full anonymity of KY signatures into an IND-CPA adversary against a variant of the ElGamal encryption scheme in the $QR(N)$ group, where factorization is not used for the purpose of decryption and can be made public.

Traceability. The KY scheme cannot satisfy full traceability as mentioned in Section 2.3.5. The KY scheme, however, seems to guarantee insider traceability from Definition 2.18 under the strong RSA assumption from Defintion 3.3 in the Random Oracle Model. From the strong RSA assumption it follows directly that an adversary on input gpk would not be able to generate a valid message-signature pair (m^*, σ^*), which cannot be open then to some member of the group. Due to the correctness of the KY scheme the judgement procedure will never output 0 if the signature is verifiable and the opening procedure returns $i = 0$.

Non-Frameability. The KY scheme seems to satisfy the notion of full non-frameability from Definition 2.21 under the DL assumption over $QR(N)$ groups with known factorization from Definition 3.5 in the Random Oracle Model, where the factorization of N is known to the adversary. An adversary with input $(gpk, gmsk, \boldsymbol{reg})$, which is able to output a valid message-signature pair (m^*, σ^*) on behalf of some group member i can be used to compute the discrete logarithm of A to the base a. Furthermore, the soundness property of the NIZKPoK J and the authentication of the transcript $\texttt{trans}_i$ by an honest member i prevent the adversary being in control of the group manager from outputting a message-signature pair (m^*, σ^*) that would open to i and for which the algorithm Judge would output 1.

5.5.3. The KY Scheme with Distributed Authorities

As mentioned before the KY scheme offers an easy way to distribute the role of the group manager across an issuer and an opener, resulting in a DA-scheme as defined in Section 2.4. The only modification concerns the key generation process. Namely, the modified key generation algorithm GKg on input 1^κ modifies the output $(gpk, ik, \boldsymbol{reg}, ok)$ of the original key generation algorithm such that:

- group public key $gpk = (N, a_0, a, g, h, y, \hat{y})$
- secret issuing key $ik = (gpk, p, q)$
- secret opening key $ok = (gpk, x, \hat{x})$
- registration list $\boldsymbol{reg}$ is initially empty.

This is the only modification needed to obtain the DA-scheme out of the basic KY scheme. Note that the group manager's secret key $gmsk$ is divided into two parts: the secret issuing key ik and the secret opening key ok. The issuer obtains the prime factors p and q of N needed to admit prospective group members by executing the original join protocol. More precisely, a prospective group member i interacts with the trusted third party T and the issuer in the join protocol to obtain his secret signing key $\boldsymbol{gsk}[i]$. On the other hand, the opener obtains the discrete logarithms x and $\hat{x}$ of the corresponding public values y and $\hat{y}$ needed to identify signers. Thus, the KY group signature scheme with distributed authorities has almost the same specification as the scheme with a single group manager. In particular, it also offers the same level of security as the basic scheme.

6. Group Signatures in the Discrete Logarithm Setting

So far only few schemes have been proposed in the DL setting. These schemes are typically less efficient than schemes in the RSA setting and the lack of trapdoor permutations (that seem to be necessary for constructing group signatures satisfying modern security definitions) in the pure DL setting makes their design challenging. Therefore, many schemes use a rather mixed setting, by employing the additional RSA parameters, typically generated in some trusted way. An early dynamic but inefficient construction in the DL setting was proposed by Chen and Pedersen [73]. In particular, the length of signatures produced by this scheme was not constant and its security with respect to modern definitions is questionable, e.g. the scheme cannot prevent framing attacks, mounted by coalitions of malicious group members. Moreover, identification of the signer had to be performed through a cooperation of group members. Similar drawbacks were also inherent to another scheme in the DL setting, proposed by Petersen [160]. Ateniese and de Medeiro [12] came up with a construction in a mixed setting, where in addition to the DL setting they required RSA parameters generated by a trusted party; however, without requiring knowledge of factorization (the trapdoor property) for the functionality of the scheme. Later, Furukawa and Yonezawa [94] came up with a slightly different version of the scheme, where no additional RSA parameters were used and management authorities were separated. In the following we describe these two schemes in more detail.

6.1. The Ateniese-de Medeiros Scheme

In this section we present the group signature scheme proposed by Ateniese and Medeiros [12]. This scheme, which we refer to as AM, is dynamic and offers verifiable opening. It thus belongs to the class of schemes that were defined in Section 2.3. In the following we provide the description of the AM scheme and discuss its security.

6.1.1. The AM Scheme

The AM group signature scheme is dynamic and offers verifiable opening. It uses the DL setting and has three security parameters κ, λ_1, and λ_2 where $\lambda_1 < \lambda_2$. Furthermore the scheme assumes the existence of a trusted party and a public key infrastructure. In the following we specify the core algorithms and protocols of the AM scheme. Our description follows the specification from [12], except that we also provide the specification of the judgement procedure that has been omitted in the original work.

Key generation. The key generation algorithm GKg on input κ performs the following steps:

1. Pick primes P, Q, $\bar{P}$ such that $|Q| = \kappa$, $P = 2Q + 1$ and $\bar{P} = 2P + 1$. Let $\mathbb{G}$ be $QR(P)$ (of order Q) and let $\mathbb{F}$ be $QR(\bar{P})$ (of order P).
2. Pick three random generators g, g_1, g_2 of $\mathbb{G}$.
3. Trusted party computes safe RSA modulus N using the $\mathsf{RSAGen}(1^{2\kappa})$ algorithm. Let $\mathbb{E}$ denote $QR(N)$.
4. Pick random $x \in_R \mathbb{Z}_Q$ and compute $y = g^x \bmod P$.
5. Pick random $z \in_R \mathbb{Z}_Q$ and compute $y_2 = g_2^x \bmod P$.
6. Output $(gpk, gmsk, \boldsymbol{reg})$ such that:
   - group public key $gpk = (P, Q, \bar{P}, N, g, g_1, g_2, y, y_2)$
   - group manager's secret key $gmsk = (gpk, x, z)$
   - $\boldsymbol{reg}$ is the initially empty registration list.

Note that the order of $\mathbb{E}$ remains unknown and several groups could safely share the parameters $P, Q, \bar{P}, N, g, g_1, g_2$ as long as N was computed by a trusted party and the factorization is never revealed. The RSA modulus N and the auxiliary group $\mathbb{E}$ are required for committing to several values during the signature generation.

Join protocol. The join protocol Join is executed between the group manager with input $gmsk = (gpk, x, z)$ and a prospective member i with input $gpk = (P, Q, \bar{P}, N, g, g_1, g_2, y, y_2)$. It proceeds as follows:

1. Member i picks random $m \in_R \mathbb{Z}_Q$, computes $J_i = g_1^m \bmod P$ and sends J_i to the group manager.
2. The group manager picks random $a, b \in_R \mathbb{Z}_Q$ and sends (a, b) to i.
3. Member i computes $I_i = J_i^a g_1^b \bmod P$ and $u_i = am+b \bmod Q$ and sends to group manager I_i together with a proof π_i computed as
$$\mathsf{NIZKPoK} \quad u_i \quad : \quad I_i = g_1^{u_i} \bmod Q \quad .$$
4. The group manager proceeds if $I_i = J_i^a g_1^b \bmod P$ and π_i is correct. The group manager GM picks $k \in_R \mathbb{Z}_Q$ and computes $r_i = I_i g^{-k} \bmod P$ and checks whether $r_i < P - 2^{\lambda_2+1}\sqrt{P}$. If r_i does not meet this requirement, the group manager repeats this step until a suitable r_i is found. Finally, the group manager computes $s_i = k - x r_i \bmod Q$ and sends (r_i, s_i) to i as a membership certificate.
5. Member i verifies that $I_i = r_i y^{r_i} g^{s_i} \bmod P$, and if so stores $\boldsymbol{gsk}[i] = (gpk, r_i, s_i, I_i, u_i)$ as its secret signing key.

6. The group manager sets $\boldsymbol{reg}[i] = (I_i, \pi_i, r_i, s_i, \texttt{trans}_i)$ where $\texttt{trans}_i$ is the communication transcript.

Note that (r_i, s_i) in the above Join protocol is a modified Nyberg-Rueppel signature [154] on I_i under the group manager's public key. The above join protocol should be performed over a secure channel in order to prevent leakage of the pair (r_i, s_i) which is sent in clear to member i. Furthermore, the communication transcripts $\texttt{trans}_i$ that the group manager stores in $\boldsymbol{reg}$ are assumed to be authenticated by i in a way that any publication of $\texttt{trans}_i$ uniquely identifies i who participated in the join protocol. These assumptions can be realized using PKI in which candidate members have certified private/public key pairs. That is i can sign the transcript using those keys.

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, r_i, s_i, u_i)$ of member i, where $gpk = (P, Q, \bar{P}, N, g, g_1, g_2, y, y_2)$, and a message $m \in \{0,1\}^*$, and proceeds as follows:

1. Pick random $l, l' \in_R \mathbb{Z}_Q$ and compute
$$T_1 = I_i y_2^{l'} \bmod P, \qquad T_2 = g_2^{l'} \bmod P, \qquad T_3 = r_i^{-1} y_2^{l} \bmod P, \qquad T_4 = g_2^{l} \bmod P.$$
2. Pick a random generator χ of $\mathbb{F}$.
3. Pick two generators γ, β of $\mathbb{E}$ provably at random.
4. Pick random $s_2 \in_R [-2^{\kappa/2+1+\lambda_1}, 2^{\kappa/2+1+\lambda_1}]$.
5. Compute $T_5 = \chi^{r_i} \mod \hat{P}$ and $T_6 = \gamma^{r_i}\beta^{s_2} \bmod N$.
6. Compute signature of knowledge S as
$$\mathsf{SoK}\left[\begin{array}{ccc} & & T_1 = g_1^{u_i} y_2^{l'} \bmod P \text{ and } T_2 = g_2^{l'} \bmod P \\ & & T_3 = r_i^{-1} y_2^{l} \bmod P \text{ and } T_4 = g_2^{l} \bmod P \\ u_i, l', l, r_i, s_i, s_2, t & : & T_5 = \chi^{r_i} \mod \hat{P} \text{ and } T_6 = \gamma^{r_i}\beta^{s_2} \bmod N \\ & & T_1T_3 = y^{r_i} g^{s_i} y_2^{t} \bmod P \text{ and } T_2T_4 = g_2^{t} \bmod P \\ & & r_i \in [-2^{\lambda_2}\sqrt{c}, c + 2^{\lambda_2}\sqrt{c}] \end{array}\right] m$$
for $c = P - 2^{\lambda_2+1}\sqrt{P}$.
7. Output group signature $\sigma = (S, T_1, T_2, T_3, T_4, T_5, T_6, \chi, \gamma, \beta)$.

Note that (T_3, T_4) and (T_1, T_2) are ElGamal ciphertexts of r_i and I_i, respectively, under the secret opening key of the group manager and thus can be used by the group manager to identify the signer. T_5 and T_6 can be seen as commitments to r_i. Together with the SoK signature S, these commitments prove that the signer possesses a valid certificate issued by the group manager. Note that picking γ and β provably at random can be achieved by using a secure pseudo random number generator and disclosing the seed resulting in the two values. Although we do not give details about the construction of S, we notice that as described in [12] the actual length of S is proportional to the number of bits in the output of the hash function used in its computation.

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (P, Q, \bar{P}, N, g, g_1, g_2, y, y_2)$, a message m, and a candidate group signature σ, and proceeds as follows:

1. Parse σ as $(S, T_1, T_2, T_3, T_4, T_5, T_6, \chi, \gamma, \beta)$.
2. If S is a valid SoK signature on message m then output 1; otherwise output 0.

Opening procedure. The opening algorithm Open takes as input the group manager's secret key $gmsk = (gpk, x, z)$, where $gpk = (P, Q, \bar{P}, N, g, g_1, g_2, y, y_2)$, message m, group signature $\sigma = (S, T_1, T_2, T_3, T_4, T_5, T_6, \chi, \gamma, \beta)$, and registration list ***reg*** and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.
2. Decrypt (T_1, T_2) by $I_i = T_1/T_2^x \bmod P$.
3. Find i for which $\boldsymbol{reg}[i] = (I_i, \pi_i, r_i, s_i, \mathtt{trans}_i)$.
4. Compute J as $\mathsf{NIZKPoK} \quad x \quad : \quad T_1 I_i^{-1} = T_2^x \bmod P$ and $y_2 = g_2^x \bmod P \quad .$
5. Output (i, τ) where $\tau = (I_i, r_i, s_i, \mathtt{trans}_i, J)$.

Judgement procedure. The judgement algorithm Judge takes as input the group public key $gpk = (P, Q, \bar{P}, N, g, g_1, g_2, y, y_2)$, message m, signature σ, identity i, and proof τ, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.
2. Parse τ as $(I_i, r_i, s_i, \mathtt{trans}_i, J)$.
3. If all of the following holds then output 1; otherwise output 0:
   - J is a valid NIZKPoK proof
   - I_i is included in $\mathtt{trans}_i$.
   - $\mathtt{trans}_i$ is authenticated by i
   - (r_i, s_i) satisfies $I_i = r_i y^{r_i} g^{s_i} \bmod P$

The validity of the NIZKPoK proof J ensures, that the group manager correctly decrypted the claimed value I_i from the ElGamal ciphertext (T_1, T_2). The proof that I_i identifies i stems from the authenticity of $\mathtt{trans}_i$ in combination with the validity of the membership certificate (r_i, s_i).

6.1.2. Security of the AM Scheme

Ateniese and Medeiros [12] did not provide a security proof of their scheme. They do, however, provide arguments, why their scheme remains secure in the Random Oracle Model. We will discuss the security of the AM scheme in the light of our definitions.

Anonymity. The AM scheme seems to satisfy the insider anonymity notion from Definition 2.3. The knowledge of secret signing keys $\boldsymbol{gsk}[i]$ of all group members would not help the adversary $\mathcal{A}$ in distinguishing, whether the challenge signature $\sigma^* = (S, T_1, T_2, T_3, T_4, T_5, T_6, \chi, \gamma, \beta)$ has been produced by signer i_0 or signer i_1. The corresponding SoK signature S has computational ZK property, i.e., it does not reveal any information about $\boldsymbol{gsk}[i_b]$. (T_3, T_4) and (T_1, T_2) are ElGamal encryptions of r_i and I_i, respectively, under the group manager's secret opening key. As the adversary does not know $gmsk$, it cannot decide if the signature is produced by i_0 or i_1 as the ElGamal-Encryption is IND-CPA secure under the DDH assumption. T_5, T_6 and T_1T_3 are commitments to r_i in different groups. As commitments have at least computational hiding property it is infeasible for the adversary to extract the value of r_i from any of the commitments. Values χ, γ, and β are chosen at random and contain no information about the signer. It follows that the adversary is not able to distinguish who signed the challenge signature.

Note that the AM scheme cannot satisfy the (stronger) full anonymity notion from Definition 2.4 because the group signature contains values χ and $T_5 = \chi^{r_i}$. Hence, a full anonymity adversary in possession of secret signing keys can simply check whether $T_5 = \chi^{r_0}$ or $T_5 = \chi^{r_1}$, and thus identify the signer.

Traceability. The AM scheme is dynamic and cannot satisfy full traceability as mentioned in Section 2.2.4. The AM scheme, however, seems to offer insider traceability from Definition 2.18. Since the group manager remains honest the adversary $\mathcal{A}$ has to come up with the SoK signature S^* which is valid for $(T_1, T_2, T_3, T_4, T_5, T_6, \chi, \gamma, \beta)$ and message m, while the decryption of the ElGamal ciphertext (T_1, T_2) by the group manager results in some I_i for which no member exists in $\boldsymbol{reg}$. The SoK soundness property, which holds in the Random Oracle Model, ensures that any successful verification of the group signature implies that $(T_1, T_2, T_3, T_4, T_5, T_6, \chi, \gamma, \beta)$ satisfy the required relations. Moreover, the PoK property of S^* guarantees that in this case $\mathcal{A}$ knows (r_i, s_i), which is a valid modified Nyberg-Rueppel signature under the group manager's secret issuing key. This is infeasible because $\mathcal{A}$ does not learn $gmsk$ and the modified Nyberg-Rueppel signature scheme is unforgeable under the DL assumption in the Random Oracle Model and the Generic Group Model as proven in [13]. Hence, the opening algorithm will likely output (i, τ) that will be accepted by the judgement algorithm that checks the validity of the modified Nyberg-Rueppel signature and verifies the authenticity of $\texttt{trans}_i$, signed by the i.

Non-Frameability. The AM scheme seems to satisfy the notion of full non-frameability from Definition 2.21. The reason is that the algorithm Judge first ensures that (m, σ^*) can be successfully verified. As discussed above this means that σ has been produced with knowledge of (r_i, s_i, u_i). Although the adversary $\mathcal{A}$ may learn (r_i, s_i) by acting as the group manager in the join protocol, it is hard for $\mathcal{A}$ to actually obtain u_i due to the hardness of the DL problem in $\mathbb{Z}_Q^*$

and the ZK property of the NIZKPoK proof in the join protocol and the SoK signature in the signing algorithm. $\mathcal{A}$ could still generate (m^*, σ^*) on behalf of some corrupted group member and attempt to provide a proof $\tau = (I_i, r_i, s_i, \mathtt{trans}_i, J)$ that this pair opens to some honest member i. This is, however, prevented by the soundness property of the NIZKPoK J, which holds in the Random Oracle Model, and the assumption that each $\mathtt{trans}_i$ is authenticated by the corresponding group member i.

6.2. The Furukawa-Yonezawa Scheme

In this section we present the group signature scheme proposed by Furukawa and Yonezawa [94]. This scheme, which we refer to as FY, is dynamic with distributed authorities and offers verifiable opening. It thus belongs to the class of schemes that were defined in Section 2.4. In the following we provide the description of the FY scheme and discuss its security.

6.2.1. The FY Scheme

The FY group signature scheme is dynamic with distributed authorities and offers verifiable opening. It uses the DL setting and has one security parameters κ. The FY scheme assumes the existence of a public key infrastructure.

In the following we specify the core algorithms and protocols of the FY scheme. Our description follows the specification from [94], except that we also provide specification of the judgement procedure that has been omitted in the original work.

Key generation. The key generation algorithm GKg on input κ performs the following steps:

1. Pick sufficiently large primes P, Q, $\bar{P}$ such that $P = k_1 Q + 1$ and $\bar{P} = k_2 P + 1$ for some $k_1, k_2 \in \mathbb{N}$. Let $\mathbb{G}_Q$ be a subgroup of $\mathbb{Z}_P^*$, of order Q and let $\mathbb{G}_P$ be a subgroup of $\mathbb{Z}_{\bar{P}}^*$, of order P.
2. Choose $g, h, f \in_R \mathbb{G}_Q$ and $G, H \in_R \mathbb{G}_p$.
3. Pick random $v \in_R \mathbb{Z}_Q$, compute $y = h^v \bmod P$.
4. Pick random $\omega \in_R \mathbb{Z}_Q$, compute $e = g^\omega \bmod P$.
5. Output $(gpk, ik, \boldsymbol{reg}, ok)$ such that:
    - group public key $gpk = (e, y, P, Q, \bar{P}, g, h, f, G, H)$
    - secret issuing key $ik = (gpk, v)$
    - $\boldsymbol{reg}$ is the initially empty registration list
    - secret opening key $ok = (gpk, \omega)$.

Join protocol. The join protocol Join is executed between the group issuer with input $ik = (gpk, v)$ and a prospective member i with input $gpk = (e, y, P, Q, \bar{P}, g, h, f, G, H)$. It proceeds as follows:

1. Member i picks random $x_i \in_R \mathbb{Z}_Q$, computes $I_i = g^{x_i} \bmod P$, generates the non-interactive zero knowledge proof of knowledge π_i as
$$\mathsf{NIZKPoK}\quad x_i \quad : \quad I_i = g^{x_i} \bmod P$$
and sends (I_i, π_i) to the issuer.
2. The issuer checks if I_i and π_i are valid. He picks random $\rho \in_R \mathbb{Z}_Q$ and computes $r_i = I_i h^\rho \bmod P$ and $\xi_i = \rho - r_i v \bmod Q$. The issuer sends (r_i, ξ_i) to i as a membership certificate.
3. Member i checks whether (r_i, ξ_i) satisfies $r_i = y^{r_i} g^{x_i} h^{\xi_i} \bmod P$ and $r_i \in [0, P-1]$ and if so stores $\boldsymbol{gsk}[i] = (gpk, r_i, \xi_i, x_i,)$ as its secret signing key.
4. The issuer adds $(I_i, \pi_i, r_i, \xi_i, trans_i)$ to the registration list $\boldsymbol{reg}$ where $\mathtt{trans}_i$ is the communication transcript.

Note that (r_i, ξ_i) in the above Join protocol can be seen as a modified Nyberg-Rueppel signature [154] on I_i under the issuer's public key. The communication transcripts $\mathtt{trans}_i$ that the issuer stores in $\boldsymbol{reg}$ are assumed to be authenticated by i in a way that any publication of $\mathtt{trans}_i$ uniquely identifies i who participated in the join protocol. These assumptions can be realized using PKI in which candidate members have certified private/public key pairs. That is i can sign the transcript using those keys.

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, r_i, \xi_i, x_i)$ of member i, where $gpk = (e, y, P, Q, \bar{P}, g, h, f, G, H)$, and a message $m \in \{0,1\}^*$, and proceeds as follows:

1. Pick random $\tau \in_R \mathbb{Z}_Q$ and compute $T_1 = g^\tau \bmod P$ and $T_2 = r_i^{-1} e^\tau \bmod P$.
2. Pick random $\alpha \in_R \mathbb{Z}_Q$ and compute $T_3 = y^{r_i} f^\alpha \bmod P$.
3. Pick random $\beta \in_R \mathbb{Z}_P$ and compute $T_4 = G^{r_i} H^\beta \bmod \bar{P}$.
4. Compute signature of knowledge S as
$$\mathsf{SoK}\left[\; r_i, \xi_i, x_i, \tau, \alpha, \beta \quad : \quad \begin{array}{c} T_1 = g^\tau \bmod P \;\text{ and }\; T_2 = r_i^{-1} e^\tau \bmod P \\ T_3 = y^{r_i} f^\alpha \bmod P \;\text{ and }\; T_4 = G^{r_i} H^\beta \bmod \bar{P} \\ T_2 T_3 = f^\alpha g^{-x_i} h^{-\xi_i} e^\tau \bmod P \;\text{ and }\; r_i \in [0, P-1] \end{array} \right] \quad m \quad .$$
5. Output group signature $\sigma = (S, T_1, T_2, T_3, T_4)$.

Note that (T_1, T_2) is an ElGamal encryption of r_i and T_3 and T_4 can be seen as Pedersen commitments [159] of r_i. Note that the actual length of S is proportional to the number of bits in the output of the hash function used in its computation (akin to [12]).

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (e, y, P, Q, \bar{P}, g, h, f, G, H)$, a message m, and a candidate group signature σ, and proceeds as follow:

1. Parse σ as (S, T_1, T_2, T_3, T_4).
2. If S is a valid SoK signature on message m then output 1; otherwise output 0.

Opening procedure. The opening algorithm Open takes as input the secret opening key ok, message m, group signature σ, and registration list $\boldsymbol{reg}$ and proceeds as follows:

1. Parse σ as (S, T_1, T_2, T_3, T_4).
2. Decrypt $r_i = T_1^\omega / T_2 \bmod P$.
3. Find i for which $\boldsymbol{reg}[i] = (I_i, \pi_i, r_i, \xi_i, \mathtt{trans}_i)$.
4. Compute J as $\mathsf{NIZKPoK} \quad \omega \quad : \quad T_1^\omega = r_i^{-1} T_2 \bmod P \quad$.
5. Output (i, τ) where $\tau = ((I_i, \pi_i, r_i, \xi_i, \mathtt{trans}_i), J)$.

Judgement procedure. The judgement algorithm Judge takes as input the group public key $gpk = (e, y, P, Q, \bar{P}, g, h, f, G, H)$, message m, signature σ, identity i, and proof τ, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.
2. Parse τ as $((I_i, \pi_i, r_i, \xi_i, \mathtt{trans}_i), J)$.
3. If all of the following holds then output 1; otherwise output 0:
   - J is a valid NIZKPoK proof
   - r_i is included in $\mathtt{trans}_i$
   - $\mathtt{trans}_i$ is authenticated by i
   - (r_i, ξ_i) satisfies $r_i = y^{r_i} I_i h^{\xi_i}$ and $r_i \in [0, P-1]$.

The validity of the NIZKPoK proof J ensures, that the opener correctly decrypted the claimed value r_i from the ElGamal ciphertext (T_1, T_2). The proof that r_i identifies i stems from the the authenticity of $\mathtt{trans}_i$ in combination with properties of (r_i, ξ_i).

6.2.2. Security of the FY Scheme

Furukawa and Yonezawa [94] proved their scheme to possess several security properties, namely *membership manager invulnerability*, *tracing manager invulnerability* and *member invulnerability*. These notions roughly correspond to our definitions of insider traceability, full anonymity and full non-frameability for schemes with distributed authorities from Section 2.4. In the following we revisit the security of the FY scheme in the light of our definitions.

Anonymity. The FY scheme seems to offer full anonymity from Definition 2.25 in the Random Oracle Model and the Generic Group Model [170]. The latter is a restricted model of computation that abstracts away any concrete representation of group elements, meaning that group operations are performed using respective calls to some black-box oracle, which represents the group operation procedure,. The knowledge of secret signing keys $\boldsymbol{gsk}[i]$ of all group members would not help the adversary $\mathcal{A}$ in distinguishing, whether the challenge signature $\sigma^* = (S, T_1, T_2, T_3, T_4)$ has been produced by signer i_0 or signer i_1, since the corresponding SoK signature has statistical ZK property, i.e., it does not reveal any information about $\boldsymbol{gsk}[i_b]$. Furthermore, if $\boldsymbol{gsk}[i_0]$ and $\boldsymbol{gsk}[i_1]$ are known to $\mathcal{A}$, then distinguishing which of them has been used to compute T_1, T_2, T_3, T_4 and S remains hard. T_3 and T_4 are both Pedersen commitments [159] having perfect hiding property. Thus, it is impossible for $\mathcal{A}$ to decide which r_i has been committed. By construction, the challenge signature contains an ElGamal encryption of r_i with a random value τ and a signature of knowledge on τ. Such a pair, consisting of the ciphertext and signature, can be regarded as a signed ElGamal encryption scheme, which was proven to be IND-CCA secure by Schnorr and Jakobsson [167]. Since an adversary against the full anonymity of the FY scheme can be used to break the IND-CCA security of the signed ElGamal encryption scheme, distinguishing whether the challenge signature has been produced by member i_0 or member i_1 is hard.

Traceability. The FY scheme is dynamic and cannot satisfy full traceability as mentioned in Section 2.2.4. The FY scheme, however, seems to guarantee insider traceability from Definition 2.26 under the DL assumption in the Random Oracle Model and the Generic Group Model [170]. Since the opener remains honest the adversary $\mathcal{A}$ has to come up with the SoK signature S^* which is valid for (T_1, T_2, T_3, T_4) and message m, while the decryption of the ElGamal ciphertext (T_1, T_2) by the opener results in some r_i for which no member exists in $\boldsymbol{reg}$. The SoK soundness property ensures that any successful verification of the group signature implies that (T_1, T_2, T_3, T_4) satisfy the required relations. Moreover, the PoK property of S^* guarantees that in this case $\mathcal{A}$ knows (r_i, ξ_i, x_i) such that (r_i, ξ_i) is a valid modified Nyberg-Rueppel signature under the issuing key ik. This is infeasible to forge because $\mathcal{A}$ never gets ik and the modified Nyberg-Rueppel signature scheme is proven to be UNF secure under the above mentioned assumptions in [13]. Hence, the opening algorithm will likely output (i, τ) that will be accepted by the judgement algorithm that checks the validity of the modified Nyberg-Rueppel signature and verifies the authenticity of $\texttt{trans}_i$, signed by i.

Non-Frameability. The FY scheme is likely to provide the notion of full non-frameability from Definition 2.28 in the Random Oracle Model under the DL assumption. The reason is that the algorithm Judge first ensures that (m, σ) can be successfully verified. As discussed above this means that σ has been produced with knowledge of (r_i, ξ_i, x_i). Although the adversary $\mathcal{A}$ may learn (r_i, ξ_i) by acting as the group manager in the join protocol, it is hard for $\mathcal{A}$ to actually obtain x_i due to the hardness of the DL problem in $\mathbb{Z}_Q$ and the ZK property of the NIZKPoK proofs in the join protocol and the signature generation. $\mathcal{A}$ could still generate (m^*, σ^*) on behalf of some corrupted group member and attempt to provide a proof $\tau = ((I_i, \pi_i, r_i, \xi_i, \texttt{trans}_i), J)$ that this pair opens to some honest member i. This is, however, prevented by the soundness property of the NIZKPoK J and the assumption that each $\texttt{trans}_i$ is authenticated by the identified user.

6.2.3. Approach to Distribute Join and Open Procedures

Furukawa and Yonezawa [94] further mention that computations performed by the issuer and the opener can be carried out in a distributed fashion. That is, multiple issuers and openers can be involved such that neither of them needs to be fully trusted to perform its respective tasks. In the following we highlight the intuition of this approach, which can be realized using *threshold secret sharing* techniques introduced by Pedersen [158].

Let ik be the secret issuer key and ok the secret opener key generated by the GKg algorithm. The issuer key ik is distributed into n elements $ik_1, \dots, ik_n$ satisfying $v = ik_1 + \cdots + ik_n$ and each key (gpk, ik_i) is given to one of the n issuers. Similarly the opener key ok is distributed into n elements $ok_1, \dots, ok_n$ satisfying $\omega = ok_1 + \cdots + ok_n$ and each key (gpk, ok_j) is given to one of the n openers. In the Join protocol, each issuer $j \in [1, n]$ computes $r_i = I_i h^t$ and $\xi'_j = k_j - r_i \cdot ik_j$ where t and k_j are computed in a distributed way such that they satisfy the equation $t = h^{k_1 + \cdots + k_n}$. The joining member i obtains its membership certificate (r_i, ξ_i) by computing $\xi_i = \xi'_1 + \cdots + \xi'_n$. In the Open algorithm, each opener $j \in [1, n]$ computes $T_{1,j} = T_1^{ok_j}$ and the entire decryption of r_i can be completed by computing $r_i = (T_{1,1} \cdots T_{1,n})/T_2 \bmod P$.

7. Group Signatures in the Setting of Bilinear Maps

Group signature schemes in the setting of bilinear maps (pairings) are usually more efficient than schemes in the RSA and DL settings. First constructions based on pairings were discovered independently by Boneh, Boyen, and Shacham [36] and by Camenisch and Lysyanskaya [57] under different hardness assumptions. The scheme from [36] relies on Linear encryption techniques. Its basic version is static but can be extended towards the dynamic setting and the additional revocation mechanism. However, this scheme offers CPA-full anonymity only. The scheme from [57] is dynamic and relies on Cramer-Shoup encryption [77] in the target group to suit the setting of bilinear maps. These two schemes influenced many other pairing-based constructions. Nguyen and Safavi-Naini [151] introduced a dynamic scheme based on ElGamal encryption in the target group. They explicitly deployed user PKI to obtain verifiable opening (akin to the model from [25]). Another pairing-based scheme with distributed authorities was introduced by Furukawa and Imai [93] by applying ElGamal encryption in the input group, which can be used with special types of bilinear groups. Kiayias and Yung [120] came up with a construction of a dynamic scheme in a mixed setting that used bilinear groups and RSA parameters, focusing explicitly on the security of concurrently executed join procedures. Delerablée and Pointcheval [81] came up with a dynamic scheme using Double ElGamal encryption techniques, which has comparable security to the scheme from [120] but without the additional RSA parameters. Boyen and Waters [39, 40] proposed dynamic group signature schemes using (composite) bilinear groups of unknown order, offering CPA-full anonymity in the standard model. Another dynamic group signature scheme in the standard model with distributed authorities and verifiable opening is due to Groth [105]. More recently, Bichsel et al. [31] introduced a dynamic group signature scheme, which unlike previous schemes does not use the "sign-and-encrypt-and-prove" paradigm, leading to better efficiency. Their scheme requires user PKI and achieves verifiable opening with a single group manager.

In the following we focus on three group signature schemes in the setting of bilinear maps. We first detail the design of the two initial constructions by Boneh, Boyen, and Shacham [36] and by Camenisch and Lysyanskaya [57], and then describe the more recent scheme by Bichsel et al. [31].

7.1. The Boneh-Boyen-Shacham Scheme

In this section we present the group signature scheme proposed by Boneh, Boyen, and Shacham [36]. This scheme, which we refer to as BBS, is static and comes with distributed authorities for issuing the secret signing keys to prospective group members and for opening their group signatures. The BBS scheme can thus be seen as a representative of group signature schemes that we defined in Section 2.4. Furthermore, the BBS scheme can be extended in two ways: (1) to support revocation of group members, leading to a partially dynamic scheme, where existing members can be revoked but no new members added, and (2) to support a fully dynamic behavior according to our definitions in Section 2.2. From an historical point of view, BBS was amongst the first group signature schemes based on bilinear maps. In the following we describe algorithms and security of the BBS scheme and its extensions.

7.1.1. The BBS Scheme

The BBS group signature scheme has a single security parameter $\kappa \in \mathbb{N}$ and uses bilinear groups $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, and $\mathbb{G}_T$ of prime order Q with $|Q| = \kappa$, a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, and an efficiently computable homomorphism ψ from $\mathbb{G}_2$ to $\mathbb{G}_1$ with $\psi(g_2) = g_1$.

In the following we specify the core algorithms and protocols of the BBS scheme. Our description follows the specification from [36], except that we apply the notation used for group signature schemes with distributed authorities from Section 2.4 (adopted for static schemes) while the original description omitted an explicit denotation of the issuing authority.

Key generation. The key generation algorithm GKg on input 1^κ and the number of groups members n performs the following steps:

1. Select $h \in_R \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}$ and $\xi_1, \xi_2 \in_R \mathbb{Z}_Q^*$.
2. Set $u, v \in \mathbb{G}_1$ such that $u^{\xi_1} = v^{\xi_2} = h$.
3. Select $\gamma \in_R \mathbb{Z}_Q^*$ and set $w = g_2^\gamma$.
4. Using γ, generate for each user i, $1 \leq i \leq n$, an SDH tuple (A_i, x_i) with $x_i \in_R \mathbb{Z}_Q^*$ and $A_i = g_1^{1/(\gamma + x_i)}$.
5. Output $(gpk, ik, \boldsymbol{gsk}, \boldsymbol{reg}, ok)$ such that:
   - group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, h, u, v, w)$
   - secret issuing key $ik = (gpk, \gamma)$
   - secret opening key $ok = (gpk, \xi_1, \xi_2)$
   - n-vector of secret signing keys of member i: $\boldsymbol{gsk}[i] = (gpk, A_i, x_i)$
   - n-vector registration list with $\boldsymbol{reg}[i] = (A_i)$ for each member i.

It is assumed that key generation is performed in a trusted way. In particular, this means that the elements h, ξ_1, ξ_2, γ are chosen independently at random from $\mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}$ respectively $\mathbb{Z}_Q^*$ and, more importantly, γ is not known to the opener. This assumption is necessary to ensure trust into the group public key gpk.

As mentioned in Section 2.4, the secret issuing key ik can safely be erased in this fully static version of the BBS scheme; it is not needed until it comes to user revocation as we will see later.

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, A_i, x_i)$ of member i, where $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, h, u, v, w)$, and a message $m \in \{0,1\}^*$, and proceeds as follows:

1. Select $\alpha, \beta \in_R \mathbb{Z}_Q$ and compute:

$$T_1 = u^\alpha, \quad T_2 = v^\beta, \quad T_3 = A_i h^{\alpha+\beta}.$$

2. Compute S as a signature of knowledge

$$\mathsf{SoK}\left[\begin{array}{ccc} & & T_1 = u^\alpha \text{ and } T_2 = v^\beta \\ \alpha, \beta, x_i & : & e(T_3, g_2)^{x_i} e(h, w)^{-\alpha-\beta} e(h, g_2)^{-x_i\alpha - x_i\beta} = e(g_1, g_2)/e(T_3, w) \\ & & T_1^{x_i} u^{-x_i\alpha} = 1 \text{ and } T_2^{x_i} v^{-x_i\beta} = 1 \end{array}\right] m \quad .$$

3. Output group signature $\sigma = (S, T_1, T_2, T_3)$.

In the above signature generation algorithm (T_1, T_2, T_3) is a Linear encryption of A_i under the public key (u, v, h), decryptable using the according private key (ξ_1, ξ_2) whose IND-CPA security is based on the DLIN assumption. The SoK signature thus proves that the signer is in possession of a pair (A_i, x_i) such that $A_i = g_1^{1/(\gamma+x_i)}$; thus, proving that the signer has a valid signing key $\boldsymbol{gsk}[i]$ and that σ can be opened by the group manager, who can decrypt A_i from the ciphertext.

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, h, u, v, w)$, a message m, and a candidate group signature σ and proceeds as follows:

1. Parse σ as (S, T_1, T_2, T_3).

2. If S is a valid SoK signature on message m then output 1; otherwise output 0.

Opening procedure. The opening algorithm Open takes as input the secret opening key $ok = (gpk, \xi_1, \xi_2)$ where $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, h, u, v, w)$, a message m, a group signature σ, and the registration list $\boldsymbol{reg}$ and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.

2. Parse σ as (S, T_1, T_2, T_3).

3. Compute $A_i = T_3/(T_1^{\xi_1} T_2^{\xi_2})$.

4. Find i for which $\boldsymbol{reg}[i] = (A_i)$, if no such i exists, output 0.

5. Output i.

7.1.2. Security of the BBS Scheme

Boneh, Boyen, and Shacham [36] prove that their scheme satisfies the notions of CPA-full anonymity, insider traceability, and an intermediate flavor between our insider and full non-frameability. In the following we discuss security of the BBS scheme in the light of our security notions from Section 2.4.

Anonymity. The BBS scheme does not satisfy the full anonymity notion from Definition 2.4, but the relaxed notion of CPA-full anonymity from Remark 2.1.3. The latter is similar to full anonymity except that the adversary is not allowed to query the opening oracle (neither in the first nor in the second attack stage). Boneh, Boyen, and Shacham [36] prove this property for their scheme in the Random Oracle Model under the DLIN assumption (cf. Definition 3.10). Their proof shows, how to use a successful CPA-full anonymity adversary to break the IND-CPA security of the Linear Encryption scheme, whose security in turn requires the DLIN assumption.

Traceability. Boneh, Boyen, and Shacham [36] also proved that their scheme satisfies the full traceability requirement of Bellare, Micciancio, and Warinschi [22], which slightly differs from our notion (cf. Remark 2.1.4), where the adversary receives the secret issuing key ik. In the static BBS scheme where revocation is not handled it is however possible to safely erase ik at the end of the key generation procedure. Therefore, this version of BBS would also satisfy our notion of full traceability under the q-SDH assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ from Definition 3.9.

Non-Frameability. The BBS scheme seems to offer full non-frameability from Definition 2.28 under the q-SDH assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ (cf. Definition 3.9). The reason is similar to the case of traceability of the BBS scheme. Indeed, the erasure of the issuing key ik at the end of the key generation procedure is sufficient for full non-frameability of the static BBS scheme. The impact on non-frameability of the BBS scheme with revocation support, where ik cannot be erased as it is needed for revocation purposes, is discussed in the next section.

7.1.3. Extensions of the BBS Scheme

In addition to their basic scheme, Boneh, Boyen, and Shacham [36] introduce an extension to address revocation of users, resulting in a partially dynamic scheme where members can only be removed from the group (but not added). We highlight these modifications using the terminology from Section 2.2.2.

Key generation. The key generation algorithm GKg publishes the initially empty update information $\boldsymbol{upd}$. Additionally, the secret issuing key has to be stored for later revocation operations and cannot be erased anymore as in the fully static scheme.

Revocation procedure. The new revocation algorithm Revoke of the BBS scheme proceeds as follows. In order to revoke a member i, the group manager computes $A_i^* = g_2^{1/(\gamma+x_i)} \in \mathbb{G}_2$ (note that $A_i = \psi(A_i^*)$) and adds the tuple (A_i^*, x_i) to $\boldsymbol{upd}$. He also computes the new public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, \hat{g}_1, \hat{g}_2, e, h, u, v, \hat{w})$ where $\hat{g}_1 = g_1^{1/(\gamma+x_i)} = \psi(A_i^*)$, $\hat{g}_2 = g_2^{1/(\gamma+x_i)} = A_i^*$, and

$\hat{w} = (\hat{g}_2)^\gamma = g_2(A_i^*)^{-x_i}$. Note that the new public key can also be computed by anyone knowing the old one and ***upd***, i.e., it suffices to distribute ***upd***.

Update procedure. The new update algorithm UpdM of the BBS scheme proceeds as follows. An unrevoked group member i in possession of (A_i, x_i) obtains the recent changes from ***upd*** (consisting of entries of the form (A_r, x_r) for a revoked user r). For each entry (A_r, x_r) member i computes $\hat{A} = \psi(A_r^*)^{1/(x_i - x_r)}/A_r^{1/(x_i-x_r)}$ and sets her new private key to be $(\hat{A}_i, x_i)$ which satisfies, as required,

$$(\hat{A}_i)^{\gamma+x_i} = \psi(A_r^*)^{\frac{\gamma+x_i}{x_i-x_r}}/A_r^{\frac{\gamma+x_i}{x_i-x_r}} = \psi(A_r^*)^{\frac{(\gamma+x_i)+(x_i-x_r)}{x_i-x_r}}/g_1^{1/(x_i-x_r)} = \psi(A_r^*) = \hat{g}_1.$$

A revoked user cannot construct a private key for the new public key unless the user can break the SDH assumption. In brief, given an SDH challenge one can easily generate a public key tuple $(Q, \mathbb{G}_1, \mathbb{G}_2, \hat{g}_1, \hat{g}_2, e, h, u, v, \hat{w})$ along with the private key $(g_1^{1/(\gamma_r+x_r)}, x_r)$ for a revoked user r. An algorithm able to forge signatures given these two tuples can be used to solve the SDH challenge.

Revocation extension for BBS, however, seems to reduce its security to only insider traceability (cf. Definition 2.26) and insider non-frameability (cf. Definition 2.27), since the issuing key *ik* cannot be erased anymore after the key generation. The adversary in the full traceability game is thus able to compute a new signing key (A_i^*, x_i^*) and generate signatures that verify but open to some user i for which no registration entry ***reg***$[i]$ exists. In the full non-frameability game the adversary has additionally *write* access to ***reg*** and hence can easily generate a new signing key (A_i^*, x_i^*) for an existing member i^* and overwrite ***reg***$[i^*]$ with (A_i^*). That way, the adversary is able to output a valid signature that opens to an existing member i^*. To prevent the latter attack, one could additionally store as part of the registration list ***reg***$[i]$ the authenticated communication transcript $\texttt{trans}_i$ of the protocol Join, in which member i receives its secret signing key ***gsk***$[i] = (A_i, x_i)$. Such authentication of transcripts could be realized using a PKI. The full non-frameability adversary would be unable to produce an authenticated transcript without knowledge of x_i.

The second extension to the BBS scheme described in [36] introduces protocol Join to handle the admission of new group members, which makes the scheme (fully) dynamic. Roughly, the secret signing key ***gsk***$[i]$ that member i obtains in the joining protocol with the issuer consists of a triple (A_i, x_i, y_i) satisfying the equation $A_i^{\gamma+x_i} h_1^{y_i} = g_1$ for some $h_1 \in \mathbb{G}_1$, which is part of the group public key *gpk*. Of crucial importance for the security of the dynamic scheme is, that y_i is chosen by the group member i and remains unknown to the issuer due to the use of appropriate ZKPoK proofs on the member's side. In fact, security properties of the basic BBS scheme are preserved by this dynamic variant. In particular, its full non-frameability holds only in combination with authenticated transcripts and a user PKI.

7.2. The Camenisch-Lysyanskaya Scheme

In this section we present the group signature scheme proposed by Camenisch and Lysyanskaya [57]. This scheme, which we refer to as CL, is dynamic and comes with distributed authorities for issuing the secret signing keys to prospective group members and for opening their group signatures. The CL scheme can thus be seen as a representative of group signature schemes, defined in Section 2.4. Historically, the CL scheme, along with the BBS scheme, was one of the first group signature schemes in the setting of bilinear maps.

7.2.1. The CL Scheme

The CL group signature scheme is a dynamic scheme with distributed authorities. It has a single security parameter $\kappa \in \mathbb{N}$ and uses cyclic groups $\mathbb{G} = \langle g \rangle$ and $\mathbb{G}_T = \langle g_T \rangle$ of prime order Q with $|Q| = \kappa$, a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, and the generator $g_T = e(g, g)$. Additionally, a collision resistant hash function $\mathsf{Hash} : \{0,1\}^* \to \mathbb{Z}_Q$ is used and modeled as a random oracle in the proof of security.

In the following we specify the core algorithms and protocols of the CL scheme. Our description follows the specification from [57].

Key generation. The key generation algorithm GKg on input 1^κ performs the following steps:

1. Select $x \in_R \mathbb{Z}_Q$, $y \in_R \mathbb{Z}_Q$ and set $X = g^x$, $Y = g_T^y$.
2. Select $h \in_R \mathbb{G}_T \setminus \{1_{\mathbb{G}_T}\}$, $x_1, \ldots, x_5 \in_R \mathbb{Z}_Q$ and set $y_1 = g_T^{x_1} h^{x_2}$, $y_2 = g_T^{x_3} h^{x_4}$, and $y_3 = g_T^{x_5}$.
3. Output $(gpk, ik, ok, \boldsymbol{reg})$ such that:
   - group public key $gpk = (Q, \mathbb{G}, \mathbb{G}_T, g, g_T, e, X, Y, h, y_1, y_2, y_3)$
   - secret issuing key $ik = (gpk, x, y)$
   - secret opening key $ok = (gpk, x_1, \ldots, x_5)$
   - registration list $\boldsymbol{reg}$ is initially empty.

It is assumed that key generation is performed in a trusted way. In particular, this means that the elements $x, y, h, x_1, \ldots, x_5$ are chosen independently at random from $\mathbb{Z}_Q$ respectively $\mathbb{G}_T \setminus \{1_{\mathbb{G}_T}\}$. This assumption is necessary to ensure trust into the group public key gpk.

Note that the tuple (h, y_1, y_2, y_3) is a public key of the Cramer-Shoup encryption scheme [77] over the group $\mathbb{G}_T$ and $(x_1, \ldots, x_5)$ is the corresponding private key. As the Cramer-Shoup scheme is secure under the DDH assumption it cannot be used over the group $\mathbb{G}$ (cf. Section 3.2.3) and thus has to be used over $\mathbb{G}_T$ instead.

Join protocol. The join protocol Join is executed between the issuer with input $ik = (gpk, x, y)$ and a prospective member i with input $gpk = (Q, \mathbb{G}, \mathbb{G}_T, g, g_T, e, X, Y, h, y_1, y_2, y_3)$. It proceeds as follows:

1. Member i picks random $k_i \in_R \mathbb{Z}_Q$, sets $P_{i,1} = g^{k_i}$ and sends $P_{i,1}$ authentically to the issuer together with a proof
$$\mathsf{ZKPoK}\quad k_i \quad : \quad P_{i,1} = g^{k_i} \quad .$$

2. The issuer proceeds if $P_{i,1} \in \mathbb{G}$ and the above proof was correct. It picks random $r \in_R \mathbb{Z}_Q$, computes $a_i = g^r$, $b_i = a_i^y$, and $c_i = a_i^x P_{i,1}^{rxy}$ and sends the membership certificate (a_i, b_i, c_i) back to i.

3. Member i stores $\boldsymbol{gsk}[i] = (gpk, k_i, a_i, b_i, c_i)$ as its secret signing key.

4. Finally, the issuer computes $P_{i,2} = e(P_{i,1}, g)$ and sets $\boldsymbol{reg}[i] = (P_{i,1}, P_{i,2})$.

It is assumed, that the join protocol is executed over a secure channel to protect the distribution of the membership certificate. The (a_i, b_i, c_i) part of the secret signing key $\boldsymbol{gsk}[i]$ represents an ordinary Camenisch-Lysyanskaya signature on the message k_i. This signature scheme was introduced in [57] and its unforgeability property holds under the LRSW assumption (cf. Definition 3.12).

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, k_i, a_i, b_i, c_i)$ of member i, where $gpk = (Q, \mathbb{G}, \mathbb{G}_T, g, g_T, e, X, Y, h, y_1, y_2, y_3)$, and a message $m \in \{0,1\}^*$, and proceeds as follows:

1. Select $r, r' \in_R \mathbb{Z}_Q$ and compute a blinded version of the certificate as
$$\tilde{\sigma} := (a_i^{r'}, b_i^{r'}, c_i^{r'r}) = (T_5, T_6, \tilde{c}^r) = (T_5, T_6, T_7).$$

2. Compute $P_{i,2} = g_T^{k_i} = e(P_{i,1}, g)$ and encrypt $P_{i,2}$ under the group opener's public key, i.e., choose $u \in_R \mathbb{Z}_Q$ and compute
$$T_1 = g_T^u, \quad T_2 = h^u, \quad , T_3 = y_1^u P_{i,2}, \quad T_4 = y_2^u y_3^{u\mathsf{Hash}(T_1 \| T_2 \| T_3)}.$$

3. Compute S as a signature of knowledge
$$\mathsf{SoK} \quad \mu, \rho, \upsilon \quad : \quad \begin{array}{c} e(g, T_7)^\rho = e(X, T_5) e(X, T_6)^\mu \;\text{ and }\; T_1 = g_T^\upsilon \;\text{ and }\; T_2 = h^\upsilon \\ T_3 = y_1^\upsilon g_T^\mu \;\text{ and }\; T_4 = \quad y_2 y_3^{\mathsf{Hash}(T_1 \| T_2 \| T_3)} \quad {}^{\upsilon} \end{array} \qquad m \quad .$$

4. Output group signature $\sigma = (S, T_1, T_2, T_3, T_4, T_5, T_6, T_7)$.

In the above signature generation algorithm (T_1, T_2, T_3, T_4) is a Cramer-Shoup encryption [77] of $P_{i,2}$ under the group opener's public key (h, y_1, y_2, y_3). This ciphertext is decryptable using the private key $(x_1, \ldots, x_5)$. The SoK signature S thus proves that the signer is in possession of a k_i, satisfying the equation $P_{i,1} = g^{k_i}$; thus, proving that the signer has a valid signing key $\boldsymbol{gsk}[i]$ and that σ can be opened by decrypting $P_{i,2}$ from the ciphertext.

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (Q, \mathbb{G}, \mathbb{G}_T, g, g_T, e, X, Y, h, y_1, y_2, y_3)$, a message m, and a candidate group signature σ and proceeds as follows:

1. Parse σ as $(S, T_1, T_2, T_3, T_4, T_5, T_6, T_7)$.

2. If S is a valid SoK signature on message m and $e(T_5, Y) = e(g, T_6)$ then output 1; otherwise output 0.

Opening procedure. The opening algorithm Open takes as input the secret opening key $ok = (gpk, x_1, \dots, x_5)$ where $gpk = (Q, \mathbb{G}, \mathbb{G}_T, g, g_T, e, X, Y, h, y_1, y_2, y_3)$, a message m, a group signature σ, and the registration list $\boldsymbol{reg}$ and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.
2. Parse σ as $(S, T_1, T_2, T_3, T_4, T_5, T_6, T_7)$.
3. Compute $H = \mathsf{Hash}(T_1 \parallel T_2 \parallel T_3)$ and verify that $T_4 = T_1^{x_3 + x_5 H} T_2^{x_4}$, otherwise output 0.
4. Compute $P_{i,2} = T_3/(T_1^{x_1} T_2^{x_2})$.
5. Find i for which $\boldsymbol{reg}[i] = (P_{i,1}, P_{i,2})$, if no such i exists, output 0.
6. Output i.

7.2.2. Security of the CL Scheme

Camenisch and Lysyanskaya [57] prove security of their scheme using definitions from Bellare, Micciancio, and Warinschi [22]. Additionally, they show that the interactive protocol underlying the signature of knowledge is secure. In the following we discuss security of the CL scheme in the light of our definitions of anonymity, traceability, and non-frameability from Section 2.4.

Anonymity. The CL scheme seems to provide full anonymity guarantees from Definition 2.25 in the Random Oracle Model under the DDH assumption in $\mathbb{G}_T$. The knowledge of secret signing keys $\boldsymbol{gsk}[i]$ of all group members would not help the adversary $\mathcal{A}$ in distinguishing, whether the challenge signature $\sigma^* = (S^*, T_1^*, T_2^*, T_3^*, T_4^*, T_5^*, T_6^*, T_7^*)$ has been produced by signer i_0 or signer i_1, since the corresponding SoK signature has statistical ZK property, i.e. it does not reveal any information about $\boldsymbol{gsk}[i_b]$. Furthermore, even if $\boldsymbol{gsk}[i_0]$ and $\boldsymbol{gsk}[i_1]$ are known to $\mathcal{A}$ (which full anonymity implicitly assumes), distinguishing whether these values have been used to compute $T_1^*, T_2^*, T_3^*, T_4^*$ is hard under the DDH assumption in $\mathbb{G}_T$. As values T_5^*, T_6^*, T_7^* are blinded, they also do not reveal any information about the membership certificate of i_b. Thus, any adversary breaking full anonymity, can be used to break the IND-CCA security of the Cramer-Shoup scheme [77], which is known to have this property under the DDH assumption.

Traceability. The CL scheme is dynamic and cannot satisfy full traceability as mentioned in Section 2.2.5. It, however, seems to offer insider traceability from Definition 2.26 with the modifications mentioned in Remark 2.4.2. Since the issuer remains honest the adversary $\mathcal{A}$ has to come up with the group signature $\sigma^* = (S^*, T_1^*, T_2^*, T_3^*, T_4^*, T_5^*, T_6^*, T_7^*)$ on some message m^* such that the signature is valid but the algorithm Open outputs 0. The ZK property of S^* ensures that no information about the secret key $\boldsymbol{gsk}[i]$ is leaked and the soundness property of S^* guarantees that the signer is in possession of a valid Camenisch-Lysyanskaya signature

(a_i, b_i, c_i), whose unforgeability holds under the LRSW assumption (cf. Definition 3.12). The soundness property of the SoK signature S^* implies that, if σ^* is valid then its components $T_1^*, T_2^*, T_3^*, T_4^*$ encrypt $P_{i^*,2}$.

Non-Frameability. The CL scheme seems to satisfy the notion of full non-frameability from Definition 2.28. The reason is that the algorithm Open first checks that (m^*, σ^*) can be successfully verified. The PoK property of the SoK signature S^* (which is part of σ^*) implies that σ^* has been produced with the knowledge of k_i. Although the adversary $\mathcal{A}$ may learn the membership certificate (a_i, b_i, c_i) of a user i by acting as the group issuer in the join protocol, it is hard for $\mathcal{A}$ to actually obtain k_i due to the hardness of the DL problem in $\mathbb{G}$ and the ZK property of the ZKPoK protocol, executed in Step 1 of Join, and of the SoK signatures.

7.3. The Bichsel-Camenisch-Neven-Smart-Warinschi Scheme

In this section we present the group signature scheme proposed by Bichsel, Camenisch, Neven, Smart, and Warinschi [31]. This scheme, which we refer to as BCNSW, is dynamic with verifiable opening and involves a user PKI for potential group members. The BCNSW scheme can thus be seen as a representative of group signature schemes defined in Section 2.3 employing the user PKI procedures mentioned in Section 2.3.2. In addition, the BCNSW scheme can be extended to add verifier-local revocation. Departing from the "sign-and-encrypt-and-prove" paradigm, the BCNSW scheme allows for the shortest known signature size as well as comparably low computation time while keeping a strong security level. In the following we will describe algorithms and security of the BCNSW scheme. Its extension toward verifier-local revocation is described separately in Section 8.4.

7.3.1. The BCNSW Scheme

The BCNSW group signature scheme is a dynamic scheme with verifiable opening employing a user PKI (connected with an unforgeable digital signature scheme $\Sigma = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vrfy})$ specified in Section 3.4). It has a single security parameter $\kappa \in \mathbb{N}$ and uses bilinear groups $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, and $\mathbb{G}_T$ of prime order Q with $|Q| = \kappa$ and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Additionally, two hash functions $\mathsf{Hash}_1, \mathsf{Hash}_2 : \{0,1\}^* \to \mathbb{Z}_Q$ are used and modeled as random oracles in the proof of security. The BCNSW scheme is based on the ordinary Camenisch-Lysyanskaya signature scheme [57, Scheme A] and especially makes use of the fact that these signatures are re-randomizable, i.e., given a valid signature $(a, b, c) \in \mathbb{G}_1^3$ on a message m, the signature (a^r, b^r, c^r) will also be valid for any $r \in \mathbb{Z}_Q^*$.

In the following we specify the core algorithms and protocols of the BCNSW scheme. Our description follows the specification from [31].

Key generation. The key generation algorithm GKg on input 1^κ performs the following steps:

1. Select $x \in_R \mathbb{Z}_Q$, $y \in_R \mathbb{Z}_Q$ and set $X = g_2^x$, $Y = g_2^y$.

2. Output $(gpk, gmsk, \boldsymbol{reg})$ such that:
   - group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, X, Y)$
   - group manager's secret key $gmsk = (gpk, x, y)$
   - registration list $\boldsymbol{reg}$ is initially empty.

It is assumed that key generation is performed in a trusted way. In particular, this means that the elements x, y are chosen independently at random from $\mathbb{Z}_Q$. This assumption is necessary to ensure trust into the group public key gpk.

User key generation. The user key generation algorithm UKg on input 1^κ computes and returns the private/public key pair $(\boldsymbol{usk}[i], \boldsymbol{upk}[i]) \leftarrow_R \mathsf{Kg}(1^\kappa)$ for the digital signature scheme Σ where $\boldsymbol{upk}[i]$ is assumed to be certified. As noticed in Section 2.3.2 a user PKI is modeled here through public (read) access to the list of registered public keys $\boldsymbol{upk}$.

Join protocol. The join protocol Join is executed between the group manager with input $gmsk = (gpk, x, y)$ and a prospective member i with input $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, X, Y)$ and an own PKI-certified key pair $(\boldsymbol{usk}[i], \boldsymbol{upk}[i])$. It proceeds as follows:

1. The group manager chooses a random $K_i \in_R \mathbb{Z}_Q$, computes $t_i = \mathsf{Hash}_2(K_i)$, and sends t_i to the member i.

2. Member i chooses $\tau_i \in_R \mathbb{Z}_Q$, computes $s_i = g_1^{\tau_i}$, $r_i = X^{\tau_i}$, $k_i = e(g_1, r_i)$, as well as $\bar{\sigma}_i \leftarrow_R \mathsf{Sign}(\boldsymbol{usk}[i], k_i)$, sends $(s_i, r_i, \bar{\sigma}_i)$ to the group manager together with a proof
$$\mathsf{NIZKPoK}\left[\tau_i \;:\; s_i = g_1^{\tau_i} \text{ and } r_i = X^{\tau_i}\right].$$

3. The group manager verifies the signature using $\mathsf{Vrfy}(\boldsymbol{upk}[i], e(g_1, r_i), \bar{\sigma}_i)$ and computes, if the signature is valid, $z_i = s_i g_1^{K_i}$ and $w_i = r_i X^{K_i}$, stores $(w_i, r_i, K_i, \bar{\sigma}_i)$ in $\boldsymbol{reg}[i]$, chooses $\rho_i \in_R \mathbb{Z}_Q$, computes $a_i = g_1^{\rho_i}$, $b_i = a_i^y$, and $c_i = a_i^x z_i^{\rho_i xy}$, and sends (a_i, b_i, c_i, K_i) to the user together with a proof
$$\mathsf{NIZKPoK}\left[x, y, \rho_i \;:\; \begin{array}{c} c_i = a_i^x z_i^{\rho_i xy} \text{ and } a_i = g_1^{\rho_i} \text{ and } X = g_2^x \\ Y = g_2^y \text{ and } 1 = b_i^x / g_1^{\rho_i xy} \end{array}\right].$$

4. Member i computes $\xi_i = \tau_i + K_i \bmod Q$ and checks whether $t_i = \mathsf{Hash}_2(K_i)$. She also verifies $e(a_i, Y) = e(b_i, g_2)$ and, if the verification is successful, stores the entry $\boldsymbol{gsk}[i] = (gpk, \xi_i, a_i, b_i, c_i)$.

The (a_i, b_i, c_i) part of the secret signing key represents an ordinary Camenisch-Lysyanskaya signature on message ξ_i.

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, \xi_i, a_i, b_i, c_i)$ of member i, where $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, X, Y)$, and a message $m \in \{0,1\}^*$, and proceeds as follows:

1. Re-randomize the signature by choosing $r \in_R \mathbb{Z}_Q$ and computing $T_1 = a_i^r$, $T_2 = b_i^r$, and $T_3 = c_i^r$.
2. Compute S as a signature of knowledge

$$\mathsf{SoK}\left[\quad \xi_i \quad : \quad \tfrac{e(T_3,g_2)}{e(T_1,X)} = e(T_2,X)^{\xi_i} \right] \quad m \quad .$$

3. Output group signature $\sigma = (S, T_1, T_2, T_3)$.

In the above signature generation algorithm, which leverages the re-randomizability property of ordinary Camenisch-Lysyanskaya signatures, the SoK signature proves that the signer knows ξ_i for which (T_1, T_2, T_3) is a valid signature; thus, proving that the signer has a valid signing key $\boldsymbol{gsk}[i]$.

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, X, Y)$, a message m, and a candidate group signature σ and proceeds as follows:

1. Parse σ as (S, T_1, T_2, T_3).
2. If $e(T_1, Y) = e(T_2, g_2)$ and S is a valid SoK signature on message m then output 1; otherwise output 0.

Opening procedure. The opening algorithm Open takes as input the group manager's secret key $gmsk = (gpk, x, y)$, a message m, a group signature σ, and the registration list $\boldsymbol{reg}$, and proceeds as follows:

1. Parse σ as (S, T_1, T_2, T_3).
2. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output $(0, \perp)$.
3. For all entries $\boldsymbol{reg}[i] = (w_i, r_i, K_i, \bar{\sigma}_i)$ check whether $e(T_3, g_2) = e(T_1, X)e(T_2, w_i)$ holds. If the equation holds for no entry $\boldsymbol{reg}[i]$ then output $(0, \perp)$, otherwise compute for the entry $\boldsymbol{reg}[i]$, for which the equation holds, $k_i = e(g_1, r_i)$ and J as the NIZKPoK proof

$$\mathsf{NIZKPoK}\left[\quad w_i, K_i \quad : \quad \tfrac{e(T_3,g_2)}{e(T_1,X)} = e(T_2, w_i) \text{ and } k_i = \tfrac{e(g_1,w_i)}{e(g_1,X)^{K_i}} \right].$$

4. Output (i, τ) where $\tau = (k_i, \bar{\sigma}_i, J)$.

The NIZKPoK proof J ensures that the group manager does not output some uninvolved member i for which $e(T_3, g_2) = e(T_1, X)e(T_2, w_i)$ does not hold. By putting out k_i, the verification of the signature $\bar{\sigma}_i$ on k_i becomes possible (with $e(g_1, r_i) = k_i$).

Note that the opening operation is linear in the number of users in the system, which is reasonable if the group manager has sufficient resources and the operation is not performed too often.

Judgement procedure. The judgement algorithm Judge takes as input the group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, X, Y)$, a message m, a group signature σ, an identity i, and proof τ, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.
2. Retrieve $\boldsymbol{upk}[i]$.
3. Parse τ as $(k_i, \bar{\sigma}_i, J)$.
4. If J is a valid NIZKPoK and $\mathsf{Vrfy}(\boldsymbol{upk}[i], k_i, \bar{\sigma}_i) = 1$ then output 1, otherwise output 0.

The judgement procedure ensures the validity of the group signature. Through the additional verification of the NIZKPoK proof J it obtains confidence that user i has been chosen correctly from $\boldsymbol{reg}$ in the opening step. The actual identification of the signer i is performed using the signer's PKI-certified public key $\boldsymbol{upk}[i]$ and the signature $\bar{\sigma}_i$. It is implicitly assumed that identity i points to the candidate public key $\boldsymbol{upk}[i]$ used in this final verification step.

7.3.2. Security of the BCNSW Scheme

Bichsel et al. [31] prove their scheme to be secure according to their definitions of anonymity, traceability, and non-frameability, which correspond to those we defined in Section 2.3. In the following, we highlight the intuition for the security of the BCNSW scheme.

Anonymity. The BCNSW scheme satisfies the insider anonymity notion from Section 2.3.4 in the Random Oracle Model under the SDLP assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ (cf. Definition 3.13) and the DDH assumption in $\mathbb{G}_1$, which can be assumed only for a bilinear map of Type-2 or Type-3 (cf. Definition 3.8). If an adversary is able to distinguish the re-randomization of the ordinary Camenisch-Lysyanskaya signature, then she is also able to solve the DDH problem in $\mathbb{G}_1$. The SoK signature (in the join protocol and the signature generation) has statistical ZK property and thus keeps ξ and τ_i secret. Finally, the SDLP assumption ensures that τ_i cannot be computed from s_i and r_i. The insider anonymity of the BCNSW scheme can thus be proven by using the insider anonymity adversary to break the DDH problem in $\mathbb{G}_1$. Note that the BCNSW scheme does not provide full anonymity as user's secret signing key can be used to distinguish whether a group signature was computed by that user, i.e. using ξ_i one can compute τ_i.

Traceability. The BCNSW scheme is dynamic and cannot satisfy full traceability as mentioned in Section 2.2.5. It, however, offers insider traceability from Definition 2.18 under the LRSW assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ (cf. Definition 3.12). The proof of Bichsel et al. for their notion of traceability naturally applies to our notion of insider traceability. This proof works by constructing an adversary against the existential unforgeability (cf. Definition 3.16) of ordinary Camenisch-Lysyanskaya signatures – which rests upon the LRSW assumption – incorporating an adversary breaking the traceability of BCNSW.

Non-Frameability. The BCNSW scheme satisfies the notion of full non-frameability from Definition 2.21 under the SDLP assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ (cf. Definition 3.13) and the unforgeability of the underlying digital signature scheme Σ (cf. Definition 3.16). The soundness of the SoK part of a group signature prevents its direct forging. As for the anonymity, the secret member key is protected by the ZK property of the SoK signatures and the SDLP assumption. Finally, the soundness of the Judge algorithm bases on the soundness of the NIZKPoK proof issued by Open, which itself is based on the soundness of the SoK part of the group signature. The requirements of full non-frameability for the BCNSW scheme can be proven by using the full non-frameability adversary to either forge signatures of Σ or break the SDLP problem in $(\mathbb{G}_1, \mathbb{G}_2)$. The latter problem prevents computation of τ_i from s_i and r_i, used in step 2 of the Join protocol.

8. Group Signatures with Verifier-Local Revocation

So far, we considered group signature schemes, where membership revocation (if any) was performed by the group manager through the distribution of some suitable revocation information to both the remaining group members and the verifiers (by making update information public). For example, membership revocation in the ACJT, TX, and CG schemes (in the RSA setting) and the pairing-based BBS scheme required from the group manager to update the group public key and publish some update information, that in turn was used by the remaining group members to derive their up-to-date secret signing keys. In this chapter we discuss group signature schemes, where membership revocation is handled in a more flexible way, by requiring that published revocation information is used only by verifiers such that remaining group members need not to update their secret signing keys. These schemes implement the concept of **verifier-local revocation (VLR)** introduced by Boneh and Shacham [38] and then applied and further extended by Nakanishi and Funabiki [146, 147], Libert and Vergnaud [131], and Bichsel et al. [31].

8.1. Group Signature Schemes with Verifier-Local Revocation

In this section we define algorithms and security of group signature schemes with VLR property. We also consider VLR-schemes, where the life-time of the scheme is split into distinct time intervals (TVLR-schemes), used in the corresponding signing and verification procedures, originated in [146]. The different revocation concept used in VLR/TVLR-schemes introduces some modifications to the syntax of the underlying algorithms and to the definitions of security.

8.1.1. Algorithms of VLR-Schemes and Their Correctness Property

In Definition 8.1 we specify algorithms of a VLR group signature scheme using the same syntax as introduced by Boneh and Shacham [38]. We focus on static schemes, where the number of group members is known in advance, and thus, do not explicitly model the joining protocol. That is, our model adopts definitions of static group signature schemes from Section 2.1 to address the new property.

The actual VLR property is modeled through the revocation list RL and revocation tokens $\boldsymbol{grt}[i]$ that the group manager publishes in RL, should a member i be revoked. In contrast to previous group signature schemes, the (up-to-date) revocation list RL is considered as input to the verification algorithm only. In particular, remaining group members need not to update their secret signing keys. It is implicitly assumed that the RL is authenticated and distributed by the group manager.

Definition 8.1 (Group Signature Scheme with Verifier-Local Revocation) A group signature scheme with *verifier-local revocation* $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy})$ consists of three polynomial-time algorithms:

Key generation. The randomized *group key generation* algorithm GKg takes as input the security parameter 1^κ, $\kappa \in \mathbb{N}$, and the total number of group members $n \in \mathbb{N}$, and returns a tuple $(gpk, RL, \boldsymbol{grt}, \boldsymbol{gsk})$, where gpk is the *group public key*, RL is the initially empty *revocation list*, $\boldsymbol{grt}$ is an n-element vector with $\boldsymbol{grt}[i]$ being the *revocation token for member* i, and $\boldsymbol{gsk}$ is an n-element vector with $\boldsymbol{gsk}[i]$ being the *secret signing key of member* i, $1 \leq i \leq n$.

Signature generation. The randomized *group signing* algorithm GSign takes as input a secret signing key $\boldsymbol{gsk}[i]$ and a message m, and returns a *group signature* σ.

Signature verification. The deterministic *group signature verification* algorithm GVrfy takes as input the group public key gpk, the revocation list RL, a message m, and a candidate group signature σ for m, and returns either 1 (to indicate that the signature is valid and the signer is unrevoked) or 0. ◊

Observe that Definition 8.1 does not specify any opening algorithm Open. This does not mean that group signatures σ produced by VLR-schemes cannot be opened. The actual reason for omitting Open from the syntax is that any VLR-scheme implements an opening procedure *implicitly* by using revocation tokens $\boldsymbol{grt}[i]$ — that are computed by the group manager during the key generation procedure and remain secret until the revocation of member i takes place. The opening procedure, which can be implemented for all VLR-schemes, works as follows.

Implicit opening procedure. Let $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy})$ be a VLR-scheme. The implicit opening algorithm Open takes as input the group public key gpk, vector of revocation tokens $\boldsymbol{grt}$, a message m, and a group signature σ. For all $i = 1, \ldots, n$ the algorithm checks whether

$$\mathsf{GVrfy}(gpk, \boldsymbol{grt}[i], m, \sigma) = 0,$$

until it finds *first* such i, which it then outputs as the identity of the signer. If no such i is found then the algorithm outputs 0.

The verification check used in the implicit opening procedure identifies the first group member i for whom the group signature σ becomes invalid. That is, the revocation token $\boldsymbol{grt}[i]$ can implicitly be used to identify the signer of σ.

Remark 8.1.1 In the implicit opening procedure $\boldsymbol{grt}$ takes the role of the group manager's secret key $gmsk$. Therefore, static VLR schemes do not use $gmsk$ as part of their syntax. In contrast, dynamic VLR schemes explicitly require $gmsk$, which in addition to $\boldsymbol{grt}$ would contain further secrets needed for the admission of group members.

A group signature scheme with verifier-local revocation Γ should satisfy the following correctness property, which basically guarantees that group signatures produced by some member i can be verified successfully if and only if i has not been revoked.

Definition 8.2 (Correctness : VLR) A group signature scheme with verifier-local revocation $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy})$ is *correct* if for all $\kappa, n \in \mathbb{N}$, all $(gpk, \boldsymbol{grt}, \boldsymbol{gsk}) \leftarrow \mathsf{GKg}(1^\kappa, n)$, all identities $i \in [1, n]$, and all messages $m \in \{0,1\}^*$:

$$\mathsf{GVrfy}(gpk, RL, m, \mathsf{GSign}(\boldsymbol{gsk}[i], m)) = 1 \Longleftrightarrow \boldsymbol{grt}[i] \notin RL.$$

$\diamondsuit$

The above definitions can be easily extended towards fully dynamic schemes where the group manager can admit prospective members to the group through the corresponding protocol Join. In this case, however, the group manager would need an own secret key $gmsk$, which in addition to revocation tokens $\boldsymbol{grt}$ would contain further secrets used to issue membership credentials. Note that modeling techniques from Section 2.2 can be applied to define fully dynamic VLR-schemes.

8.1.2. Verifier-Local Revocation with Time Intervals (TVLR)

A slightly different flavor of VLR-schemes are group signature schemes where for each group member $i \in [1, n]$ the group manager specifies multiple revocation tokens such that each token is valid for some specific *time interval* $j \in \{1, \ldots, T\}$, where T denotes the total number of time intervals, possibly specified during the initialization procedure. This concept was proposed by Nakanishi and Funabiki [146]. We refer to such schemes as TVLR-schemes. More precisely, in TVLR-schemes vector $\boldsymbol{grt}$ is an $(n \times T)$-element vector, in which each element $\boldsymbol{grt}[i]$ contains T revocation tokens for member i. The actual revocation token for member i at time interval $j \in \{1, \ldots, T\}$ is then denoted by $\boldsymbol{grt}[i][j]$. Similarly, for each time interval j there is a corresponding revocation list RL_j which is periodically published by the group manager.

The actual motivation for considering TVLR-schemes comes from the observation that in VLR-schemes (cf. Definition 8.1) revocation of some member i implies publication of the corresponding revocation token $\boldsymbol{grt}[i]$ in the revocation list RL. However, once $\boldsymbol{grt}[i]$ is published, it becomes possible to link all group signatures σ produced by i, including group signatures that i generated while being a legitimate (that is, unrevoked) member of the group. In contrast, TVLR-schemes can potentially prevent linkability of group signatures that were generated by member i prior to its revocation. This property is commonly called **backward unlinkability**.

In order to link the process of signature generation and verification to some specific time interval j the group signing algorithm GSign and verification algorithm GVrfy of a TVLR-scheme take j as an additional input. Furthermore, should member i be revoked at time $t \in [1, T]$ the group manager is expected to include all revocation tokens $\boldsymbol{grt}[i][j]$ with $j \geq t$ into the corresponding revocation lists RL_j.

In Definition 8.3 we specify algorithms of a TVLR-scheme by updating the syntax of VLR-schemes from Definition 8.1 to account for the use of multiple time intervals.

Definition 8.3 (VLR Group Signature Scheme with Time Intervals) A VLR group signature scheme with *time intervals* $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy})$ consists of three polynomial-time algorithms:

Key generation. The randomized *group key generation* algorithm GKg is identical to that of a VLR-scheme from Definition 8.1, except that it takes the total number of time intervals $T \in \mathbb{N}$ as an additional input and its output $\boldsymbol{grt}$ is an $(n \times T)$-element vector with $\boldsymbol{grt}[i][j]$ being the *revocation token for member i at time $j \in [1, T]$*.

Signature generation. The randomized *group signing* algorithm GSign takes as input a secret signing key $\boldsymbol{gsk}[i]$, a time interval $j \in [1, T]$, and a message m, and returns a *group signature σ*.

Signature verification. The deterministic *group signature verification* algorithm GVrfy is identical to that of a VLR-scheme from Definition 8.1, except that it takes a time interval $j \in [1, T]$ as an additional input and uses the corresponding revocation list RL_j. $\lozenge$

TVLR-schemes admit the same *implicit* opening procedure Open as VLR-schemes. More precisely, the group manager views $\boldsymbol{grt}$ as its secret key $gmsk$ and can identify the signer i of some group signature σ and message m for any given interval $j \in [1, T]$ by finding the first $i \in [1, n]$ such that $\mathsf{GVrfy}(gpk, j, \boldsymbol{grt}[i][j], m, \sigma) = 0$.

The updated syntax of TVLR-schemes leads to the modified definition of correctness where signatures produced by a group member i remain valid for all time intervals in which i was not revoked.

Definition 8.4 (Correctness : TVLR) A group signature scheme with verifier-local revocation based on time intervals $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy})$ is *correct* if for all $\kappa, n \in \mathbb{N}$, all $T \in \mathbb{N}$, all $(gpk, \boldsymbol{grt}, \boldsymbol{gsk}) \leftarrow \mathsf{GKg}(1^\kappa, n)$, all identities $i \in [1, n]$, all time intervals $j \in [1, T]$, and all messages $m \in \{0, 1\}^*$:

$$\mathsf{GVrfy}(gpk, j, RL_j, m, \mathsf{GSign}(\boldsymbol{gsk}[i], j, m)) = 1 \iff \boldsymbol{grt}[i][j] \notin RL_j.$$

$\lozenge$

Note that also TVLR-schemes can be made fully dynamic by considering the additional protocol Join in which the group manager would use its own secret key $gmsk$ to admit new members to the group.

8.1.3. Adversary Model and Oracles for VLR/TVLR-Schemes

Our formal definitions of security for VLR and TVLR group signature schemes $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy})$ will be provided through probabilistic experiments $\mathsf{Expt}_{\Gamma,\mathcal{A}}(1^\kappa)$ as mentioned in Section 1.5. In general, the adversary $\mathcal{A}$ will be given access to (a subset of) the following three oracles:

Corruption oracle. The *corruption oracle* $\mathsf{Corrupt}(\cdot)$ takes as input an identity $i \in [1, n]$ and returns the secret signing key $\boldsymbol{gsk}[i]$.

Signing oracle. The *signing oracle* $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot, \cdot)$ takes as input an identity-interval-message triple (i, j, m) with $i \in [1, n]$, $j \in [1, T]$, and returns the output of the group signing algorithm $\mathsf{GSign}(\boldsymbol{gsk}[i], j, m)$. Note that in VLR-schemes the interval j is omitted.

Revocation oracle. The *revocation oracle* $\mathsf{Revoke}(\boldsymbol{grt}, \cdot, \cdot)$ takes as input an identity-interval pair (i, j) with $i \in [1, n]$ and $j \in [1, T]$, and returns the corresponding revocation token $\boldsymbol{grt}[i][j]$. Note that in VLR-schemes the interval j is omitted.

Note that the signing oracle available to the adversary in TVLR-schemes takes time interval j as an additional input, thus allowing the adversary to request group signatures for arbitrary time intervals. Furthermore, the opening oracle Open used in the adversary model for schemes without VLR/TVLR property is replaced by the revocation oracle Revoke modeling the fact that in VLR/TVLR-schemes knowledge of revocation tokens suffices to implicitly open the group signatures.

It is further possible to extend the above adversary model towards dynamic VLR/TVLR-schemes with the appropriate oracles for handling the dynamic admission to the group as previously used in Section 2.2.3.

8.1.4. Anonymity Definitions for VLR/TVLR-Schemes

The anonymity requirement of VLR/TVLR group signature schemes aims at protection of the signers' identities and is defined similarly to the requirement of insider anonymity for schemes without VLR/TVLR properties. That is, in the anonymity experiment the adversary can corrupt all members of the group except for the two members i_0 and i_1 who serve as potential signers of the challenge group signature σ^*. The main reason for not considering full anonymity in the context of VLR/TVLR-schemes is that existing solutions allow efficient computation of revocation tokens for member i from the corresponding secret signing key $\boldsymbol{gsk}[i]$. Therefore, providing the anonymity adversary with $\boldsymbol{gsk}[i_0]$ and $\boldsymbol{gsk}[i_1]$ (as in case of full anonymity) would allow for immediate identification of the signer through the implicit opening procedure of existing VLR/TVLR-schemes. For this reason, we only discuss the insider anonymity requirement, which was also called **selfless anonymity** in [38]. In case of TVLR schemes this requirement also captures the already mentioned notion of *backward unlinkability*, which requires that group signatures σ, produced by some member i within time intervals j in which i was not revoked, remain unlinkable. This is modeled by allowing the adversary to ask for revocation tokens of potential signers i_0 and i_1 in time intervals $j > t$ where t is the interval used to generate the challenge signature σ^*. Thus, intuitively, knowledge of $\boldsymbol{grt}[i][j]$ in TVLR-schemes should not allow the adversary to efficiently compute $\boldsymbol{grt}[i][j-1]$.

Definition 8.5 (Insider Anonymity : VLR/TVLR) A VLR/TVLR group signature scheme $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy})$ provides *insider anonymity* if for all probabilistic, polynomial-time adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{I-AN}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = \quad \Pr \quad \mathsf{Expt}^{\text{I-AN}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = 1 \quad - \frac{1}{2} \quad .$$

The associated I-AN-*experiment* $\mathsf{Expt}^{\text{I-AN}}_{\Gamma,\mathcal{A}}(1^\kappa, n)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa, n)$ is executed to produce $(gpk, \boldsymbol{grt}, \boldsymbol{gsk})$.

Attack Stage I. Adversary $\mathcal{A}_1$ receives gpk. In TVLR-schemes at the beginning of every time interval $j \in [1, T]$, $\mathcal{A}$ additionally receives notification that the interval is started, i.e. that j has been incremented.

1. $\mathcal{A}_1$ can submit queries to the oracles $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot, \cdot)$, and $\mathsf{Revoke}(\boldsymbol{grt}, \cdot, \cdot)$.
2. $\mathcal{A}_1$ stops and eventually outputs a tuple (st, i_0, i_1, m^*) containing some state information st, two *uncorrupted* identities $i_0, i_1 \in [1, n]$, and a challenge message m^*. Furthermore, if Γ is a TVLR-scheme and the output of $\mathcal{A}$ occurs at time interval t then revocation oracle $\mathsf{Revoke}(\boldsymbol{grt}, \cdot, \cdot)$ must not have been queried by $\mathcal{A}$ on inputs (i_0, j) or (i_1, j) for any $j \le t$.

Challenge Stage. A bit $b \in \{0, 1\}$ is chosen at random and the signature generation algorithm $\mathsf{GSign}(\boldsymbol{gsk}[i_b], t, m^*)$ is executed to produce the challenge group signature σ^*. If Γ is a VLR-scheme then t is omitted from the input to GSign.

Attack Stage II. Adversary $\mathcal{A}_2$ receives (st, σ^*).

1. $\mathcal{A}_2$ can submit queries to the oracles $\mathsf{Corrupt}(\cdot)$, $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot, \cdot)$, and $\mathsf{Revoke}(\boldsymbol{grt}, \cdot, \cdot)$ as before, subject to the following restrictions:
   a) The corruption oracle $\mathsf{Corrupt}(\cdot)$ ignores queries of the form i_0 and i_1.
   b) The revocation oracle $\mathsf{Revoke}(\boldsymbol{grt}, \cdot, \cdot)$ ignores queries of the form (i_0, j) and (i_1, j) for any $j \le t$. If Γ is a VLR-scheme then revocation oracle ignores any query for i_0 or i_1.
2. $\mathcal{A}_2$ stops and eventually outputs a bit b^*.

Output: If $b^* = b$ then the experiment outputs 1, otherwise it outputs 0. $\diamondsuit$

8.1.5. Traceability Definitions for VLR/TVLR-Schemes

We consider the requirement of traceability in VLR/TVLR-schemes as protection against attacks mounted by a coalition of malicious group members to generate a group signature that cannot be opened through the implicit opening procedure. We provide definition of full traceability for VLR/TVLR-schemes where the adversary may form coalitions with the group manager. We consider here only static schemes, where the only secret of the group manager is a vector of revocation tokens $\boldsymbol{grt}$, which the adversary receives as input.

Definition 8.6 (Full Traceability : VLR/TVLR) A VLR/TVLR group signature scheme $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy})$ provides *full traceability* if for all probabilistic, polynomial-time adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{F-TR}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = \Pr\left[\mathsf{Expt}^{\text{F-TR}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = 1\right].$$

The associated F-TR-*experiment* $\mathsf{Expt}^{\text{F-TR}}_{\Gamma,\mathcal{A}}(1^\kappa, n)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa, n)$ is executed to produce $(gpk, \boldsymbol{grt}, \boldsymbol{gsk})$.

Attack Stage. Adversary $\mathcal{A}$ receives $(gpk, \boldsymbol{gsk}, \boldsymbol{grt})$. At some point, $\mathcal{A}$ stops and eventually outputs a tuple $(j^*, RL^*_{j^*}, m^*, \sigma^*)$. If Γ is a VLR scheme then the interval j^* is empty and the output revocation list is denoted RL^*.

Output. If $\mathsf{GVrfy}(gpk, j^*, RL^*_{j^*}, m^*, \sigma^*) = 1$ and the implicit opening procedure $\mathsf{Open}(gpk, \boldsymbol{grt}[i][j^*], m^*, \sigma^*) = 0$ for all $i \in [1, n]$ then the experiment outputs 1, otherwise it outputs 0. If Γ is a VLR-scheme then Open ignores the interval j^* and is executed using all $\boldsymbol{grt}[i]$, $i \in [1, n]$. $\diamondsuit$

The output conditions of the F-TR-experiment ensure that the forged signature σ^* is valid but cannot be opened using the revocation tokens. In case of TVLR-schemes validity of the signature and revocation tokens used to check the output conditions are bound to some time interval j^* specified by the adversary.

Remark 8.1.2 Note that dynamic VLR/TVLR-schemes cannot achieve full traceability for the same reason as dynamic group signature schemes without VLR/TVLR property cannot achieve it (cf. Section 2.2.5). That is, dynamic VLR/TVLR schemes can only achieve the weaker notion of *insider traceability* where the adversary does not receive group manager's secret key $gmsk$. An appropriate definition of insider traceability for static and dynamic VLR/TVLR-schemes can be obtained analogously to Definitions 2.5 and 2.13 for static and dynamic group signature schemes without VLR/TVLR property.

Remark 8.1.3 Our definition of full traceability for VLR/TVLR-schemes differs from the traceability definitions used by Boneh and Shacham [38] and by Nakanishi and Funabiki [146]. In particular, we do not consider attacks where a coalition of malicious members generates a valid group signature that opens to some other member of the group. We model these attacks using non-frameability definitions in Section 8.1.6 to maintain consistency with previous schemes without VLR/TVLR property.

8.1.6. Non-Frameability Definitions for VLR/TVLR-Schemes

The non-frameability requirement for VLR/TVLR-schemes protects against attacks where a coalition of malicious group members generates valid group signatures for which the implicit opening procedure identifies some honest group member as a signer. We define full non-frameability of VLR/TVLR-schemes where the adversary may form coalitions of malicious group members and is furthermore given access to the group manager's secrets (which in case of static VLR/TVLR-schemes is composed of $\boldsymbol{grt}$).

Definition 8.7 (Full Non-Frameability : VLR/TVLR) A VLR/TVLR group signature scheme $\Gamma = (\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVrfy})$ provides *full non-frameability* if for all probabilistic, polynomial-time adversaries $\mathcal{A}$, the following advantage function is negligible (in κ):

$$\mathsf{Adv}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = \Pr\left[\mathsf{Expt}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa, n) = 1\right] .$$

The associated F-NF-*experiment* $\mathsf{Expt}^{\text{F-NF}}_{\Gamma,\mathcal{A}}(1^\kappa, n)$ proceeds as follows:

Initialization. The key generation algorithm $\mathsf{GKg}(1^\kappa, n)$ is executed to produce $(gpk, \boldsymbol{grt}, \boldsymbol{gsk})$.

Attack Stage. Adversary $\mathcal{A}$ receives $(gpk, \boldsymbol{grt})$.

1. $\mathcal{A}$ can submit queries to the oracles $\mathsf{Corrupt}(\cdot)$ and $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot, \cdot)$.
2. $\mathcal{A}$ stops and eventually outputs a tuple $(j^*, RL^*_{j^*}, m^*, \sigma^*)$. If Γ is a VLR scheme then the interval j^* is empty and the output revocation list is denoted RL^*.

Output. If all of the following holds then the output of the experiment is 1:

1. $\mathsf{GVrfy}(gpk, j^*, RL^*_{j^*}, m^*, \sigma^*) = 1$
2. There exists $i^* \in [1, n]$ such that implicit $\mathsf{Open}(gpk, \boldsymbol{grt}[i^*][j^*], m^*, \sigma^*) = i^*$.
3. $\mathcal{A}$ did not submit i^* to $\mathsf{Corrupt}(\cdot)$ or $\boldsymbol{grt}[i^*][j^*] \in RL^*_{j^*}$
4. $\mathcal{A}$ did not submit (i^*, j^*, m^*) to $\mathsf{GSign}(\boldsymbol{gsk}[\cdot], \cdot, \cdot)$.

Otherwise the output is 0. $\Diamond$

The output conditions of the F-NF-experiment ensure that the forged signature σ^* is valid for some time interval j^* and that a signer i^* identified through the implicit opening procedure has not been corrupted by the adversary, unless the adversary includes the revocation token $\boldsymbol{grt}[i^*][j^*]$ into the output revocation list $RL^*_{j^*}$, meaning that verification of σ^* at interval j^* was successful even though i^* was already revoked.

Remark 8.1.4 The above definition can be extended towards dynamic VLR/TVLR-schemes by providing the adversary with the entire group manager's secret key *gmsk* and ensuring that forgery of σ^* occurred with respect to some honestly admitted group member i^* using similar restrictions as for dynamic group signature schemes in Section 2.2.6.

Remark 8.1.5 Furthermore, a weaker notion of *insider non-frameability* can be obtained from the above definition by preventing adversarial access to the group manager's secrets. The resulting definition would be similar to Definitions 2.7 and 2.14 used in the context of static and dynamic group signatures schemes without VLR/TVLR property, respectively.

8.2. The Boneh-Shacham Scheme

In this section we present the VLR group signature scheme proposed by Boneh and Shacham [38]. This scheme, which we refer to as BS, is static and was one of the first schemes that provided verifier local revocation. The BS scheme is thus a representative of VLR group signature schemes that we defined in Section 8.1. Moreover, it comes with distributed authorities for issuing the secret signing keys to prospective group members and for opening their group signatures, thus being also representative of group signature schemes that we defined in Section 2.4. In the following we describe the algorithms and security of the BS scheme.

8.2.1. The BS Scheme

The BS group signature scheme has a single security parameter $\kappa \in \mathbb{N}$ and uses bilinear groups $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, and $\mathbb{G}_T$ of prime order Q with $|Q| = \kappa$, a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, and an efficiently computable homomorphism ψ from $\mathbb{G}_2$ to $\mathbb{G}_1$ with $\psi(g_2) = g_1$. Note that the latter requirements rules out realizations of the BS scheme based on bilinear maps of Type-3 from Definition 3.8. Additionally, the scheme uses two hash functions $\mathsf{Hash}_1 : \{0,1\}^* \to \mathbb{G}_2^2$ and $\mathsf{Hash}_2 : \{0,1\}^* \to \mathbb{Z}_Q$, both modeled as random oracles. In the following we specify the core algorithms and protocols of the BS scheme. Our description follows the specification from [38].

Key generation. The key generation algorithm GKg on input 1^κ and the number of group members n performs the following steps:

1. Select $\gamma \in_R \mathbb{Z}_Q^*$ and set $w = g_2^\gamma$.
2. For each user $i \in [1, n]$, generate a tuple (A_i, x_i) with $x_i \in_R \mathbb{Z}_Q^*$ such that $\gamma + x_i = 0$ and $A_i = g_1^{1/(\gamma+x_i)}$.
3. Output $(gpk, RL, \boldsymbol{grt}, \boldsymbol{gsk})$ such that:
   - group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, w)$
   - revocation list RL is initially empty
   - n-element vector of revocation tokens of member i: $\boldsymbol{grt}[i] = (A_i)$
   - n-element vector of secret signing keys of member i: $\boldsymbol{gsk}[i] = (gpk, A_i, x_i)$.

It is assumed that key generation is performed in a trusted way. In particular, this means that the elements γ and x_i, with $i \in [1, n]$, are chosen independently at random from $\mathbb{Z}_Q^*$ and, more importantly, γ is not known to any party except for the issuer.

Since γ is not used after the key generation is performed it should be safely erased by the issuer (cf. Section 2.4.

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, A_i, x_i)$ of member i and a message $m \in \{0,1\}^*$, and proceeds as follows:

1. Pick a random nonce $r \in_R \mathbb{Z}_Q^*$. Obtain generators $(\hat{u}, \hat{v})$ in $\mathbb{G}_2$ from Hash_1 as
$$(\hat{u}, \hat{v}) = \mathsf{Hash}_1(gpk, m, r) \in \mathbb{G}_2^2$$
and compute their images in $\mathbb{G}_1$:
$$u = \psi(\hat{u}), \qquad v = \psi(\hat{v}).$$
2. Select $\alpha \in_R \mathbb{Z}_Q^*$ and compute:
$$T_1 = u^\alpha, \qquad T_2 = A_i v^\alpha.$$

3. Compute S as a signature of knowledge
$$\mathsf{SoK} \quad \alpha, x_i \quad : \quad T_1 = u^\alpha \text{ and } e(T_2 v^{-\alpha}, w g_2^{x_i}) = e(g_1, g_2) \quad m \quad .$$
4. Output group signature $\sigma = (S, r, T_1, T_2)$.

The SoK signature proves that the signer is in possession of a pair (A_i, x_i) such that $A_i = g_1^{1/(\gamma+x_i)}$; thus, proving that the signer has a valid signing key $\boldsymbol{gsk}[i]$.

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, w)$, the revocation list RL, a message m, and a candidate group signature σ and proceeds as follows:

1. Parse σ as (S, r, T_1, T_2).
2. Compute u and v as in the signature generation algorithm.
3. Check that S is a valid SoK signature on message m (output 0 if not).
4. For all $A_i = \boldsymbol{grt}[i] \in RL$ with $i \in [1, n]$, check whether A_i is encoded in (T_1, T_2) by checking if
$$e(T_2/A_i, \hat{u}) = e(T_1, \hat{v}).$$
If no element of RL is encoded in (T_1, T_2) then output 1; otherwise output 0.

Implicit opening procedure. The implicit opening algorithm Open takes as input the group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, w)$, the vector of revocation tokens $\boldsymbol{grt}$, a message m, and a group signature σ and proceeds as follows:

1. Check for all $i \in [1, n]$ whether $\mathsf{GVrfy}(gpk, \boldsymbol{grt}[i], m, \sigma) = 0$.
2. Output the *first* such i or 0 if no such i is found.

8.2.2. Security of the BS Scheme

Boneh and Shacham [38] were the first to introduce a formal model and security definitions for group signature schemes with verifier local revocation property, which we defined in Sections 8.1.4 to 8.1.6. In our model in Section 8.1 we separate their original definition of traceability into two definitions by considering non-frameability as a distinct goal. In the following, we discuss the security of the BS scheme based on our definitions.

Anonymity. The BS scheme seems to offer insider anonymity from Definition 8.5 in the Random Oracle Model under the DLIN assumption in $\mathbb{G}_2$ (cf. Definition 3.10). The SoK signature (in the signature generation) has statistical ZK property and thus keeps x_i and α secret. Moreover, T_1 and T_2 are parts of the Linear Encryption of A_i and thus hide A_i under the DLIN assumption. Boneh and Shacham prove their scheme to provide selfless anonymity, which corresponds to our notion of insider anonymity, by showing how to use a successful insider anonymity adversary to break the DLIN assumption.

Traceability. The BS scheme seems to provide full traceability under the q-SDH assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ (cf. Definition 3.9), given that γ is erased after the key generation process as recommended[1]. The adversary $\mathcal{A}$ would have to come up with the group signature $\sigma^* = (S^*, r^*, T_1^*, T_2^*)$ on some message m^* such that the signature is valid but the algorithm Open fails, i.e. returns 0. The ZK property of S^* ensures that no information about x_i and α is leaked and its soundness property ensures that the signer is in possession of a valid pair (A_i, x_i), which is unforgeable under the q-SDH assumption. The soundness property of the SoK signature S^* thus ensures that if σ^* is valid then its components T_1^*, T_2^* encrypt A_i.

Non-Frameability. The BS scheme seems to achieve the notion of full non-frameability from Definition 8.7 under the q-SDH assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ (cf. Definition 3.9). This is because the algorithm Open first ensures that (m^*, σ^*) is valid using the verification procedure. The PoK property of the SoK signature S^* (which is part of σ^*) implies that σ^* has been produced with knowledge of x_i. Although the adversary $\mathcal{A}$ knows the revocation token A_i of user i, it remains hard for $\mathcal{A}$ to actually obtain x_i due to the hardness of the DL problem in $\mathbb{G}_1$. We notice that the traceability notion used by Boneh and Shacham [38] actually implies our definition of non-frameability (cf. Remark 8.1.3).

8.3. The Nakanishi-Funabiki Scheme

In this section we present the group signature scheme proposed by Nakanishi and Funabiki [146]. This scheme, which we refer to as NF, is static and was the first scheme that considered verifier local revocation with backward unlinkability based on bilinear maps. The NF scheme is thus a representative of TVLR group signature schemes that we defined in Section 8.1. Moreover, it comes with distributed authorities for issuing the secret signing keys to prospective group members and for opening their group signatures, thus being also representative of group signature schemes defined in Section 2.4. The Nakanishi-Funabiki scheme can be seen as an extension of the Boneh-Shacham scheme from Section 8.2 towards a VLR property with time intervals. It should be noticed that security of the NF scheme has been proven in the random oracle model and that the TVLR approach of Nakanishi and Funabiki was later applied by Libert and Vergnaud [131] to the group signature scheme proposed by Boneh and Waters [40] to achieve the TVLR property in the standard model (though in a less efficient way and under somewhat stronger hardness assumptions). In the following we thus focus on the description of algorithms and security underlying the NF scheme.

[1] γ can be seen as the secret key of the issuer in the distributed authorities setting, which is not needed anymore after the key generation process. If γ would not be erased but instead stored as ik, this issuer key would be handed to the adversary in the traceability game, reducing the scheme's security to insider traceability.

8.3.1. The NF Scheme

The NF group signature scheme has a single security parameter $\kappa \in \mathbb{N}$ and uses cyclic groups $\mathbb{G} = \langle g \rangle$ and $\mathbb{G}_T$ of prime order Q with $|Q| = \kappa$, and a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. In the following we specify the core algorithms and protocols of the NF scheme. Our description follows the specification from [146].

Key generation. The key generation algorithm GKg on input 1^κ, the number of groups members n and the total number of time intervals T performs the following steps:

1. Select $\tilde{g} \in_R \mathbb{G}$.
2. Select $h_j \in_R \mathbb{G}$ for all $j \in [1, T]$.
3. Select $\gamma \in_R \mathbb{Z}_Q^*$ and set $w = g^\gamma$.
4. For each user $i \in [1, n]$, generate a tuple (A_i, x_i) with $x_i \in_R \mathbb{Z}_Q^*$ such that $\gamma + x_i = 0$ and $A_i = g^{1/(\gamma+x_i)}$.
5. Compute $B_{ij} = h_j^{x_i}$ for all $i \in [1, n]$ and $j \in [1, T]$.
6. Output $(gpk, RL, \boldsymbol{grt}, \boldsymbol{gsk})$ such that:
   - group public key $gpk = (Q, \mathbb{G}, \mathbb{G}_T, g, \tilde{g}, e, h_1, \ldots, h_T, w)$
   - revocation lists RL_j are initially empty
   - $(n \times T)$-element vector of revocation tokens for member i and interval j: $\boldsymbol{grt}[i][j] = (B_{ij})$
   - n-element vector of secret signing keys of member i: $\boldsymbol{gsk}[i] = (gpk, A_i, x_i)$.

It is assumed that key generation is performed in a trusted way. In particular, this means that the elements h_j, γ, and x_i are chosen independently at random from $\mathbb{G}$ respectively $\mathbb{Z}_Q^*$ and, more importantly, γ is not known to any party except for the issuer.

Since γ is not used after the key generation is performed it should be safely erased by the issuer (cf. Section 2.4.

Compared to the BS scheme (cf. Section 8.2), the secret signing keys stay the same whereas gpk is extended by the components $h_1, \ldots, h_T$ and $\boldsymbol{grt}$ stores entries B_{ij} which depend on h_j and x_i, thus enabling revocation in specified time intervals.

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, A_i, x_i)$ of member i, a time interval $j \in [1, T]$, and a message $m \in \{0, 1\}^*$. Thereafter, it is assumed that m includes the time interval j, binding the signature to that interval. The algorithm proceeds as follows:

1. Select $\alpha, \beta, \delta \in_R \mathbb{Z}_Q^*$ and compute:

$$T_1 = A_i \tilde{g}^\alpha, \qquad T_2 = g^\alpha \tilde{g}^\beta; \qquad T_3 = e(g^{x_i}, h_j)^\delta, \qquad T_4 = g^\delta$$

2. Compute S as a signature of knowledge

$$\mathsf{SoK}\left\{\alpha, \beta, \delta, x_i, A_i \quad : \quad \begin{array}{c} T_1 = A_i\tilde{g}^{\alpha} \text{ and } T_2 = g^{\alpha}\tilde{g}^{\beta} \text{ and } T_3 = e(g^{x_i}, h_j)^{\delta} \\ T_4 = g^{\delta} \text{ and } e(A_i, wg^{x_i}) = e(g, g) \end{array}\right\}(m) \quad .$$

3. Output group signature $\sigma = (S, T_1, T_2, T_3, T_4)$.

The SoK signature proves that the signer is in possession of a pair (A_i, x_i) such that $A_i = g^{1/(\gamma+x_i)}$; thus, proving that the signer has a valid signing key $\boldsymbol{gsk}[i]$. The elements T_3 and T_4 in the signature bind it to the specified time interval j.

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (Q, \mathbb{G}, \mathbb{G}_T, g, \tilde{g}, e, h_1, \ldots, h_T, w)$, a time interval $j \in [1, T]$, the corresponding revocation list RL_j, a message m, and a candidate group signature σ and proceeds as follows:

1. Parse σ as (S, T_1, T_2, T_3, T_4).
2. Check that S is a valid SoK signature on message m (output 0 if not).
3. For each element $\boldsymbol{grt}[i][j] = B_{ij} \in RL_j$, check whether

$$T_3 = e(T_4, B_{ij}).$$

If none of the elements in RL_j satisfies the equation then output 1; otherwise output 0.

Implicit opening procedure. The implicit opening algorithm Open takes as input the group public key $gpk = (Q, \mathbb{G}, \mathbb{G}_T, g, \tilde{g}, e, h_1, \ldots, h_T, w)$, the vector of revocation tokens $\boldsymbol{grt}$, a time interval $j \in [1, T]$, a message m, and a group signature σ and proceeds as follows:

1. Check for all $i \in [1, n]$ whether $\mathsf{GVrfy}(gpk, \boldsymbol{grt}[i][j], m, \sigma) = 0$.
2. Output the *first* such i or 0 if no such i is found.

8.3.2. Security of the NF Scheme

Nakanishi and Funabiki [146] extend the anonymity notion proposed by Boneh and Shacham [38] towards backward unlinkability as discussed in Section 8.1.4. In the following, we discuss the security of the NF scheme based on the definitions for TVLR schemes, i.e., we separate the traceability notion used in [146] into our notions of traceability and non-frameability.

Anonymity. The NF scheme seems to provide insider anonymity from Definition 8.5 in the Random Oracle Model under the DBDH assumption in bilinear groups ($\mathbb{G}$, $\mathbb{G}$) (cf. Definition 3.11). The SoK signature (in the signature generation algorithm) has statistical ZK property and does not reveal information about A_i, x_i, α, β, and δ. Backward unlinkability is provided by the fact that the computation of $h_{j'}^{x_i}$ from an other token $h_j^{x_i}$ is infeasible. Using the DBDH assumption and assuming that the hash function used in the generation of S behaves as a random oracle, Nakanishi and Funabiki could show that signature values T_1, T_2, T_3, and T_4 produced for some time interval j make it hard for the anonymity adversary to distinguish between the two possible signers of this signature as long as their secret signing keys need not to be revealed to the adversary.

Traceability. The NF scheme seems to offer full traceability under the q-SDH assumption in $(\mathbb{G}, \mathbb{G})$ (cf. Definition 3.9), given that γ is erased after the key generation process as recommended[2]. The adversary $\mathcal{A}$ has to come up with the group signature $\sigma^* = (S^*, T_1^*, T_2^*, T_3^*, T_4^*)$ on some message m^* such that the signature is valid but the algorithm Open outputs 0. The ZK property of S^* ensures that no information about A_i, x_i, α, β, and δ is leaked and the soundness property ensures that the signer is in possession of a valid secret signing key (A_i, x_i). Nakanishi and Funabiki show that it is infeasible for the traceability adversary to obtain a valid pair (A_i, x_i) satisfying the relationship $A_i = g^{1/(\gamma + x_i)}$ under the q-SDH assumption.

Non-Frameability. The NF scheme seems to satisfy the notion of full non-frameability from Definition 8.7 under the q-SDH assumption in $(\mathbb{G}, \mathbb{G})$ (cf. Definition 3.9). The reason is that the algorithm Open first ensures that (m^*, σ^*) can be successfully verified. The PoK property of the SoK signature S^* (which is part of σ^*) implies that σ^* has been produced with knowledge of x_i. Although the adversary $\mathcal{A}$ knows the revocation tokens B_{ij} of user i, it remains hard for $\mathcal{A}$ to actually obtain x_i due to the hardness of the DL problem in $\mathbb{G}$. We notice that the traceability notion used by Nakanishi and Funabiki [146] actually implies our notion of non-frameability (cf. Remark 8.1.3).

8.4. The Bichsel-Camenisch-Neven-Smart-Warinschi Scheme

In this section we present the VLR variant of the group signature scheme proposed by Bichsel, Camenisch, Neven, Smart, and Warinschi [31] described in Section 7.3. This variant, which we refer to as BCNSW-VLR, is – as the BCNSW scheme – dynamic with verifiable opening (cf. Section 2.3) and involves user PKI for potential group members (cf. Section 2.3.2). Unlike the basic BCNSW scheme, it allows for verfier-local revocation. In the following we describe algorithms of the BCNSW-VLR scheme focusing on its VLR-specific modifications, and provide a short discussion of its security with respect to the security notions for VLR schemes.

8.4.1. The BCNSW-VLR Scheme

The BCNSW-VLR group signature scheme uses the same setting as the BCNSW scheme, namely a user PKI (connected with an unforgeable digital signature scheme $\Sigma = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vrfy})$ specified in Section 3.4), a single security parameter $\kappa \in \mathbb{N}$, bilinear groups $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 =$

[2] γ can be seen as the secret key of the issuer in the distributed authorities setting, which is not needed anymore after the key generation process. If γ would not be erased but instead stored as ik, this issuer key would be handed to the adversary in the traceability game, reducing the scheme's security to insider traceability.

$\langle g_2 \rangle$, and $\mathbb{G}_T$ of prime order Q with $|Q| = \kappa$ and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, two hash functions $\mathsf{Hash}_1, \mathsf{Hash}_2 : \{0,1\}^* \to \mathbb{Z}_Q$ (modeled as random oracles), and the ordinary Camenisch-Lysyanskaya signature scheme [57, Scheme A] with re-randomizable signatures. In the following we specify the core algorithms and protocols of the BCNSW-VLR scheme.

The **key generation** procedure remains the same as for the basic BCNSW scheme and additionally outputs an empty revocation list RL and an empty list of revocation tokens $\boldsymbol{grt}$.

The **user key generation** procedure remains the same as for the basic BCNSW scheme.

The **join protocol** remains the same as for the basic BCNSW scheme; the group manager only stores the revocation token $\boldsymbol{grt}[i] = (w_i)$ in addition.

The **signature generation** procedure remains the same as for the basic BCNSW scheme.

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, X, Y)$, the revocation list RL, a message m, and a candidate group signature σ and proceeds as follows:

1. Parse σ as (S, T_1, T_2, T_3).
2. Check that $e(T_1, Y) = e(T_2, g_2)$ and that S is a valid SoK signature on message m; output 0 if not.
3. For all $\boldsymbol{grt}[i] = w_i \in RL$ check whether $e(T_3, g_2) = e(T_1, X)e(T_2, w_i)$ holds. If none of the elements in RL satisfies the equation then output 1; otherwise output 0.

The idea of Bichsel et al.'s VLR construction is to integrate a part of the opening procedure into the verification. The needed modifications are possible in the sub-class of schemes where it suffices that the Open algorithm gets only $(gpk, m, \sigma, \boldsymbol{reg})$ as input instead of $(gmsk, m, \sigma, \boldsymbol{reg})$ (i.e., just the registration list $\boldsymbol{reg}$ is needed to open a signature, not also the group manager's secret key $gmsk$). Note that the BCNSW scheme (Section 7.3) falls into this sub-class. To be exact, the element w_i of a user's entry in $\boldsymbol{reg}$ suffices to identify the issuer of a signature, thus the group manager simply has to store w_i in RL in order to revoke a user i.

Implicit opening procedure. The implicit opening algorithm Open, extended to provide verifiable opening, takes as input the group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, X, Y)$, the vector of revocation tokens $\boldsymbol{grt}$, the registration list $\boldsymbol{reg}$, a message m, and a group signature σ and proceeds as follows:

1. Parse σ as (S, T_1, T_2, T_3).
2. Check for all $i \in [1, n]$ whether $\mathsf{GVrfy}(gpk, \boldsymbol{grt}[i], m, \sigma) = 0$.

3. For the first such i compute $k_i = e(g_1, r_i)$ and J as the NIZKPoK proof

$$\mathsf{NIZKPoK} \quad w_i, K_i \quad : \quad \tfrac{e(T_3, g_2)}{e(T_1, X)} = e(T_2, w_i) \;\text{ and }\; k_i = \tfrac{e(g_1, w_i)}{e(g_1, X)^{K_i}}$$

   using $\boldsymbol{reg}[i] = (w_i, r_i, K_i, \bar{\sigma}_i)$.

4. Output (i, τ) where $\tau = (k_i, \bar{\sigma}_i, J)$.

The **judgement** procedure remains the same as for the basic BCNSW scheme.

8.4.2. Security of the BCNSW-VLR Scheme

Bichsel et al. [31] note that security properties of their basic BCNSW scheme remains preserved under the verifier-local revocation extension. We thus just recall that the BCNSW scheme (and so also the BCNSW-VLR scheme) provides insider anonymity (cf. Section 2.3.4) in the Random Oracle Model under the SDLP assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ (cf. Definition 3.13) and the DDH assumption in $\mathbb{G}_1$, insider traceability (cf. Definition 2.18) under the LRSW assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ (cf. Definition 3.12), and full non-frameability (cf. Definition 2.21 under the SDLP assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ (cf. Definition 3.13) and the unforgeability of the underlying digital signature scheme Σ (cf. Definition 3.16). These properties remain intuitively unchanged because the only modification to the BNCSW scheme in order to achieve VLR is the additional list of revocation tokens ***grt*** that contain elements w_i from the registration list ***reg*** for each revoked entry. In the anonymity, traceability, and non-frameability games for VLR schemes, the adversary learns the list ***grt***, while in the corresponding games for non-VLR schemes it learns ***reg***. Therefore, knowledge of ***grt*** in VLR scheme does not provide more information to the adversary than knowledge of ***reg*** in non-VLR schemes. Finally, we notice the BCNSW-VLR does not provide backward unlinkability, i.e. it does not capture time intervals of signatures and revocations as deployed in TVLR schemes. Therefore, revocation of a member in the BCNSW-VLR scheme allows to link group signatures generated by that member in the past.

9. Comparison of Group Signature Schemes

The choice of a suitable group signature scheme for a particular application may depend on a number of facts. Surely, functional properties offered by the scheme like its dynamic behavior, support for revocation and verifiability of the opening procedure play the key roles. Also security requirements satisfied by the scheme and in some cases their strength may influence the selection process. Last but not least, efficiency and performance considerations may become important if applications are executed on resource-constrained devices and in communication networks with lower bandwidths. In this chapter we characterize and compare the discussed group signature schemes from the previous chapters by considering their properties, security, and efficiency. We also address optional extensions of these schemes.

9.1. Functionality and Properties

9.1.1. Overview

This section compares described group signature schemes based on their functionality and properties. This comparison is done with respect to the dynamic behavior of schemes, their ability to open group signatures in a publicly verifiable way (VO property), to support separation of duties through distribution of management roles between the issuer and the opener (DA property), and the ability to revoke issued membership certificates. Results of our comparison are summarized in Table 9.1.

In the upper part of Table 9.1 we list four group signature schemes in the RSA setting from Chapter 5. All of them either provide or can be extended to provide dynamic admission of group members, while only the Ateniese-Camenisch-Joye-Tsudik (ACJT), Tsudik-Xu (TX), and the Camenisch-Groth (CG) schemes can be further used to revoke signing right of the group members. The Kiayias-Yung (KY) scheme is the only scheme, which assumes distributed authorities. The verifiable opening procedure is supported by the ACJT, TX, and KY schemes. However, TX scheme offers weaker opening functionality than ACJT and KY schemes, in particular in case of collusion attacks it cannot identify all colluding participants.

In the middle part of Table 9.1 we compare two group signature schemes in the DL setting from Chapter 6. The Ateniese-de Medeiros (AM) scheme and the Furukawa-Yonezawa (FY) scheme are both dynamic and provide support for the verifiable opening procedure. Yet, only FY assumes separation of duties between the issuer and the opener. Interestingly, none of the DL schemes offers revocation.

Table 9.1.: Functionality and Properties of Group Signature Schemes

Group Signature Scheme		Static	Dynamic	Verifiable Opening	Distributed Authorities	Revocation
ACJT	[16] [5.1]		✓	✓		(✓)
TX	[178] [5.3]		✓	(✓)		✓
CG	[49] [5.4]	✓	(✓)			(✓)
KY	[119] [5.5]		✓	✓	✓	
AM	[12] [6.1]		✓	✓		
FY	[94] [6.2]		✓	✓	✓	
BBS	[36] [7.1]	✓			(✓)	(✓)
CL	[57] [7.2]		✓		✓	
BCNSW	[31] [7.3]		✓	✓		
BS	[38] [8.2]	✓			✓	VLR
NF	[146] [8.3]	✓			✓	TVLR
BCNSW-VLR	[31] [8.4]		✓	✓		VLR

✓ — property is provided; (✓) — property is provided with a separate extension.

In the bottom part of Table 9.1 we compare several group signature schemes in the setting of bilinear maps, including schemes from Chapter 7 and schemes with the special property of verifier-local revocation from Chapter 8 (all of which require bilinear maps). The Boneh-Boyen-Shacham (BBS), Boneh-Shacham (BS), and Nakanishi-Funabiki (NF) schemes are strictly static, whereas the Camenisch-Lysyanskaya (CL) and both versions of the Bichsel-Camenisch-Neven-Smart-Warinschi (BCNSW and BCNSW-VLR) schemes are dynamic. Membership revocation is offered by all VLR and TVLR schemes and through the optional extension of the BBS scheme. Only BCNSW scheme and its VLR version can open signatures in a verifiable way. In contrast, all other schemes allow to distribute the role of the group manager amongst two different authorities. It should be noticed that BS and BBS schemes have been originally described as static schemes. The DA property of these schemes can only be obtained by assuming that after the key generation procedure the issuer goes offline and erases its issuing key. Finally, we observe that NF is the only TVLR scheme that we consider in our comparison.

In the following we discuss the importance of the different properties and draw some overall conclusions about the described group signature schemes.

9.1.2. Dynamic Behavior

Dynamic group signature schemes are usually preferable if the number and identities of potential group members are not known in advance, or if membership certificates have to be computed and issued on-demand. However, static group signatures can also be used in the dynamic setting, although with some significant limitations: In static schemes n membership credentials can be pre-computed by the group manager, who then hands over these credentials one by one to prospective group members. This requires, however, that pre-computed membership credentials are securely stored prior to being handed over and that the group manager does not use any of these credentials. Therefore, this approach inherently requires the group manager to be fully trusted. Moreover, this approach cannot be used with a user PKI, since PKI-based identities of future group members may not be known. Hence, static schemes, when applied to the dynamic setting will not able to provide verifiable opening (which requires either implicit or explicit use of a user PKI). In the opposite direction, any dynamic scheme can also be used in a static environment, assuming that the join protocol is executed with each group member as part of the initial key generation procedure.

Our comparison shows that all of the described group signature schemes in the RSA and the DL settings offer support for the admission of new group members. This functionality is also achieved by some of the schemes in the setting of bilinear maps.

9.1.3. Support for Verifiable Opening

The verifiable opening (VO) property prevents a misbehaving group manager from falsely accusing honest members of the group of having produced group signatures that they did not produce. This property requires the group manager to output an additional proof τ for the statement that "identity i belongs to the group member who generated signature σ on the message m". This property is useful only if the group manager is not sufficiently trusted or if the group signature is deployed by an application where possible prosecution of misbehaving group members is performed by a third party (i.e. judge). Observe that VO property requires deployment of a user PKI, which may not always exist. Therefore, verifiable opening should be treated as a rather optional property.

Still, verifiable opening is supported by most of the existing group signature schemes in the RSA and DL settings. The only scheme we discussed in the setting of bilinear maps that opens signatures in a verifiable way is the BCNSW scheme.

9.1.4. Support for Distributed Authorities

The ability to distribute the group management and the identification of signers between the issuer and the opener is useful in applications where a single group manager would not be trusted to perform both tasks. We observe that such separation of duties is not motivated by the willingness to balance computation resources across different parties. The latter can be done by storing the group manager's secret key at two or more servers with each server being responsible either for the admission or for the opening procedure. Furthermore, the instrument of separation is useful only if the issuer and the opener do not collude. Indeed, in case of collusion group signatures with distributed authorities do not have any advantage from the security point of view over the group signatures where the authority is centralized. That is, the DA property still requires a significant amount of trust into the both authorities. On the other hand, considering that the issuing key and the opening key are inherently stored at different locations, DA-schemes seem to offer better resilience against potential corruptions of the authorities; that is corrupting both authorities may become more difficult than breaking into only one of them.

The DA-property seems to be achievable by static schemes where the group manager's secret used to compute the secret signing key of a group member is no longer needed and does not depend on the secret used to open a group signature. In this case this secret can be though of being the issuing key and securely erased after the execution of the key generation procedure. This explains, for example, why BBS and BS schemes can be seen as schemes with distributed authorities. Nonetheless, this property in static schemes seems not very useful since the group manager must be trusted to erase parts of its secret key.

We also observe that the deployment of DA-schemes in practice would require additional work for enforcing the appropriate access control to the registration list ***reg***, which should be modifiable only by the issuer but at the same time readable by the opener. A potential misuse of access rights for ***reg*** would result in the loss of the key security properties of the DA-scheme.

Our comparison shows that an appropriate dynamic group signature scheme with distributed authorities can be found in each of the security settings, i.e. the KY scheme in the RSA setting, the FY scheme in the DL setting, and the CL scheme in the setting of bilinear maps. We remark that none of existing dynamic schemes with VLR property supports distributed group authorities.

9.1.5. Support for Membership Revocation

The ability of the group authority to revoke members from the group is equally important for static and dynamic schemes, and is of great usefulness in practice (akin to revocation in traditional PKI systems, cf. Section 1.1). Indeed, it cannot be assumed that secret signing keys of group members will remain secure for the entire life period of the group; for example, members may store their secret keys at less secure locations or even lose their keys, in which case these keys would have to be revoked. Furthermore, group members can be revoked if they misbehave and sign documents that they are not authorized to sign. If a group signature scheme does not support efficient revocation then the whole system would have to be re-initialized, which is clearly undesirable in practice.

Support for membership revocation can currently be seen as a bottleneck of existing group signature schemes. Existing revocation mechanisms are mainly of two types: The first and less flexible revocation mechanism is based on the update of secret signing keys of unrevoked group members. One of the practical problems behind this approach is that unrevoked group members must synchronize their signing keys prior to generating new group signatures and that also verifiers have to download the up-to-date information about the public key of the group. The second mechanism eliminates the first problem in that it requires only verifiers to be in possession of some up-to-date revocation information — schemes with the VLR property — but not the signers. Although more suitable from the practical point of view we remark that existing group signature schemes require linear amount of VLR checks (in the number of revoked group members). On the other hand, VLR mechanism is closer to the revocation mechanisms of traditional PKI systems, where verifiers check signer's public key with respect to the latest CRL of the certification authority.

Our comparison shows that revocation extension of the ACJT scheme, based on dynamic accumulators from Section 5.2, the revocation mechanism integrated into the TX scheme, and the revocation extension of the BBS scheme from Section 7.1.3 follows the first less practical approach. In contrast, VLR support is offered exclusively by schemes operating in the setting of bilinear maps. In particular, the only dynamic group signature scheme with VLR support is the BCNSW-VLR scheme. We remark that the NF scheme is also of special interest since it implements VLR with time intervals (TVLR) — a property that becomes useful when anonymity of revoked group members needs further protection.

9.2. Security Properties

9.2.1. Overview

In this section we compare group signature schemes with regard to the security properties. In Table 9.2 we give an overview of discussed group signature schemes and mention different flavors of anonymity, traceability, and non-frameability that these schemes seem to offer. We also indicate different cryptographic assumptions under which the security properties of respective schemes seem to hold.

Table 9.2.: Security Properties of Group Signature Schemes

Scheme		Setting	Anon.	Traceab.	Non-Fra.	Assumptions
ACJT	[16] [5.1]	RSA	full	insider	full	ROM, SRSA, DDH
TX	[178] [5.3]	RSA	insider	(insider)	full	ROM, SRSA, DDH, DSR
CG	[49] [5.4]	RSA	full	full	full	ROM, DL, SRSA
KY	[119] [5.5]	RSA	full	insider	full	ROM, SRSA, DDH, DL
AM	[12] [6.1]	DL	full	insider	full	ROM, DL, DDH
FY	[94] [6.2]	DL	full	insider	full	ROM, GGM, DL
BBS	[36] [7.1]	BM	CPA-full	full	full	ROM, q-SDH, DLIN, DL
CL	[57] [7.2]	BM	full	insider	full	ROM, LRSW, DDH, DL
BCNSW	[31] [7.3]	BM	insider	insider	full	ROM, LRSW, SDLP, DDH, DL
BS	[38] [8.2]	BM	insider	full	full	ROM, q-SDH , DLIN, DL
NF	[146] [8.3]	BM	insider	full	full	ROM, q-SDH, DBDH, DL
BCNSW-VLR	[31] [8.4]	BM	insider	insider	full	ROM, LRSW, SDLP, DDH, DL

In the upper part of Table 9.2 we list four group signature schemes in the RSA setting from

Chapter 5. We observe that three schemes, namely Ateniese-Camenisch-Joye-Tsudik (ACJT), Camenisch-Groth (CG), and Kiayias-Yung (KY), offer highest security guarantees with regard to their functionality. In particular, all these schemes are fully anonymous and fully non-frameable. The TX scheme offers the weaker notion of insider anonymity. The ACJT, TX, and KY schemes are dynamic and can therefore offer traceability against insiders as the highest traceability goal. However, TX scheme offers a somewhat weaker notion of traceability, since two colluding group members can still forge a signature, while only one of these members will be traced. In contrast, the CG scheme, which we consider here in its basic (static) version, satisfies the highest notion of full traceability. If the CG scheme is used with its dynamic extension then its traceability would also hold only against insiders. Security of the mentioned schemes holds in the Random Oracle Model (ROM) and is based, amongst others, on the Strong RSA (SRSA) assumption. The ACJT and KY schemes further assume that Decision-Diffie-Hellman (DDH) assumption holds in $QR(N)$ groups, while the KY and CG schemes also rely on the hardness of the Discrete-Logarithm (DL) problem, and TX scheme additionally relies on the Decision Small Residuosity (DSR) assumption from [65].

In the middle part of Table 9.2 we consider group signature schemes in the DL setting from Chapter 6. The two schemes, Ateniese-de Medeiros (AM) scheme and Furukawa-Yonezawa (FY), being dynamic offer the highest security guarantees of full anonymity, insider traceability, and full non-frameability. Security of both schemes is proven in ROM, while FY further relies on the Generic Group Model (GGM), which is also a very strong assumption. In contrast, AM assumes hardness of the DDH problem in subgroups of prime order. Additionally, both schemes rely on the hardness of the DL problem.

In the bottom part of Table 9.2 we compare group signature schemes in the setting of bilinear maps, including schemes from Chapter 7 and (T)VLR schemes from Chapter 8 (all of which use bilinear groups). The Camenisch-Lysyanskaya (CL) scheme, being dynamic, offers the strongest security guarantees for this type of schemes. In contrast, the Bichsel-Camenisch-Neven-Smart-Warinschi (BCNSW) scheme, which is also dynamic, cannot guarantee full anonymity and is thus weaker from this perspective. Although Boneh-Boyen-Shacham (BBS) scheme is static, it can only offer (weaker) CPA-full anonymity, that is anonymity of signers, whose group signatures were opened once, can no longer be guaranteed. In contrast, all (T)VLR schemes, i.e. Boneh-Shacham (BS), Nakanishi-Funabiki (NF), BCNSW-VLR, are only anonymous against insiders (aka. selfless anonymity). The reason for the lack of full anonymity in (T)VLR schemes is that their secret signing keys can be used to compute revocation tokens for corresponding signers and then break the anonymity property in the revocation check. The BCNSW-VLR scheme, being dynamic, can only offer insider traceability, whereas static BS and NF schemes achieve its full version. Security of all mentioned schemes is proven in ROM and often requires several different assumptions. Amongst the most frequent assumptions are q-Strong Diffie-Hellman (q-SDH) and LRSW, which take care of traceability and non-frameability of the respective schemes. In contrast, DDH and Decision Linear (DLIN) assumptions are usually required for anonymity. Interestingly, LRSW is often used together with DDH, whereas q-SDH typically with DLIN.

9.2.2. Anonymity of Signers

Anonymity of signers is one of the key security properties of group signatures and schemes we addressed in Table 9.2 offer this property in three different flavors. The strongest anonymity property is full anonymity (offered by the majority of static and dynamic schemes but not by schemes with (T)VLR property), which guarantees anonymity of signers and unlinkability of their signatures even in case when all secret signing keys become known to the adversary. One may think that such strong anonymity protection is not necessarily needed, since secret signing keys are likely to be protected and if they are revealed than group signatures become forgeable anyway. The actual benefit of full anonymity stems from the fact that even if secret signing keys are exposed in the future (due to careless protection or cryptanalysis) the anonymity of signers (which is primarily a privacy goal) still remains protected. The notion of CPA-full anonymity (as offered by the BBS schemes) turns out to be useful in situations where anonymity in the future should remain protected only for signers, whose signatures were never opened. If the opening mechanisms is thought to be a method of punishment then CPA-full anonymity essentially offers future protection for group signers who did not misbehave. The notion of insider anonymity (offered by BCNSW scheme and all schemes with (T)VLR property) is the weakest reasonable anonymity goal for group signatures, since it protects anonymity of the signer against any other group member or a coalition of group members. In practice, insider anonymity would offer sufficient privacy protection, unless the exposure of secret signing keys should be taken into account too. On the other hand, there might be situations, where some signer may deliberately want to claim the origin of some group signature to a third party by disclosing own secret signing key. This could be useful in group signature schemes without verifiable opening, where the group manager would not be able to prove this fact alone.

9.2.3. Traceability of Signers

The traceability property ensures that the group manager (or opener in DA-schemes) can always open (valid) group signatures and identify the signer. Full traceability is the strongest flavor since it guarantees traceability even in the presence of the malicious group manager (or issuer in DA-schemes). Unfortunately, this strong requirement can only be satisfied by static schemes (e.g. CG, BBS, BS, NF). In dynamic schemes the group manager, being responsible for the admission of group members, can always create "phantom" members and issue signatures on their behalf. That is for dynamic schemes the (weaker) notion of insider traceability gives the highest security protection and is fulfilled by the majority of existing dynamic group signatures. In practice, however, insider traceability seems to be sufficient. The reason is that the role of the group manager will typically be performed by administrative authorities whose prime goal is not to abuse the system but rather protect the system from abuses by group members with signing rights. Furthermore, it is likely that group manager's (or issuer's) secret keys will be well protected and there exposure (e.g. due to break-ins) will likely lead to the re-initialization of the entire system anyway.

9.2.4. Non-Frameability of Signers

The non-frameability property prevents false attribution of group signatures to group members that were not involved in their generation. That is, non-frameability protects primarily honest group members from being falsely accused of having signed some document. Full non-frameability is the strongest flavor since it offers such protection even if the group manager (or both authorities in DA-schemes) are malicious. It seems that full non-frameability is also desirable in practice. That is, even if the group is managed by authorities that are interested in preventing abuse of signing rights, these authorities should still not be able to accuse some innocent user of having abused the system. In group signature schemes with verifiable opening, i.e. in ACJT, TX, KY, AM, FY, and BCNSW(-VLR) schemes, full non-frameability can only be achieved through the deployment of a user PKI, in which case it can tolerate malicious group managers. For this reason the property can also be achieved in dynamic schemes (unlike full traceability), where user PKI is typically used to verify authenticity of a transcript that has been previously authenticated by the signer in the joining protocol and whose parts are revealed by the opening procedure for a given group signature in some verifiable way. Note that the corresponding user PKI must exist and be managed independently of the group signature scheme, in particular certification authorities forming the PKI may not coincide with the group manager (or with issuer/opener in DA-schemes). In group signature schemes where no separate user PKI is in place (CG, BBS, BS, NF), full non-frameability can only be achieved if the group manager's secret key is erased after the distribution of secret signing keys to all members of the group, which seems to be a reasonable assumption for static schemes only. Finally, we observe that all group signature schemes from Table 9.2 satisfy this strongest notion of full non-frameability.

9.2.5. Cryptographic Assumptions

In addition to the security properties offered by respective group signature schemes, cryptographic assumptions serve as another important criteria for evaluation of their security in practice. We notice that all group signature schemes in Table 9.2 obtain their security properties in ROM, which is not a standard assumption (cf. Section 3.3.1). This assumption is used because signing algorithms of these schemes typically output an SoK signature S as part of the group signature σ such that S is obtained using the ROM-based Fiat-Shamir transformation from an interactive ZKPoK protocol). On the other hand, ROM-based cryptographic schemes are in general more efficient and could thus be more suitable for practical purposes. Presented group signature schemes also differ with regard to their number-theoretic hardness assumptions. In general, more established assumptions, such as RSA, DL, or DDH, that have stood the test of time, are preferable. In contrast, cryptography based on bilinear maps is a rather new field, and many new hardness assumptions in bilinear groups were proposed and used over the past years. Meanwhile, q-SDH, DLIN, and LRSW belong to widely recognized cryptographic assumptions in the setting of bilinear maps. We observe that properties of bilinear groups, in particular their bilinear property, turned out to be very useful for the construction of group signatures. In particular, it is not known how to achieve (T)VLR property without using bilinear maps, and also signatures output by such schemes are typically shorter than signatures output by schemes with comparable functionality in the RSA and DL settings.

9.3. Computational Complexity : Costs and Impact of Different Algorithms

The amount of work required by different algorithms of a group signature schemes becomes a dominant factor, should the scheme be considered for practical use. Typically, computation costs of the scheme depend on the deployed cryptographic setting (e.g. RSA, DL, or bilinear maps) and can be further influenced by the extended properties, such as revocation, verifiability of the opening procedure, etc. In our analysis of *computational complexity* of group signature schemes we estimate the amount of most costly operations for different algorithms, which serves as a good heuristic for the evaluation of its efficiency. In general, we will consider modular exponentiations in different cyclic groups and pairing evaluations as the most expensive operations. In our analysis we will sometimes use heuristic methods to estimate the costs, especially with respect to the generation and verification of complex signatures of knowledge that all existing group signature schemes currently apply. Our analysis thus provides a high-level intuition about the efficiency of different group signature schemes, whose actual computational complexities can slightly differ, when it comes to the exact implementation of these signatures; for example, by applying different optimization techniques for computing modular (multi-base) exponentiations and pairing evaluations, which we do not consider in this analysis.

9.3.1. Computational Costs for Group Managers

In Table 9.3 we summarize computational overhead for the authorities that manage the group. In particular, we address efficiency of key generation and opening procedures. For dynamic group signature schemes we also estimate the amount of work in the group manager's part of the join protocol. Observe that key generation is performed only once and therefore, its computational efficiency is not as important as say the efficiency of the opening procedure or the amount of work required to admit new group members in dynamic groups, especially if groups are large and membership updates are frequent.

The upper part of the table contains group signature schemes in the RSA setting from Chapter 5, namely the Ateniese-Camenisch-Joye-Tsudik (ACJT), Tsudik-Xu (TX), Camenisch-Groth (CG), and the Kiayias-Yung (KY) schemes. In this setting the most expensive operation is modular exponentiation. ACJT, TX, and KY schemes are dynamic. Therefore, their key generation procedure is very efficient. Notice, however, that in the TX scheme a trusted third party is involved in the generation of group public key. The actual amount of work in dynamic schemes is performed during the join protocol. The CG scheme is static and the group manager has to generate secret signing keys for all users during the key generation procedure. We observe that while group manager's computations of the join protocol in TX and KY schemes is much more efficient in comparison to ACJT, the admission procedure of the KY scheme requires assistance of a trusted third party. The amount of work to open a group signature is similar for all schemes, if we assume that judge algorithm also verifiers the validity of the group signature.

Table 9.3.: Computational Costs for Group Managers

Group Signature Scheme		GKg	JoinM	Open
ACJT	[16] [5.1]	$1E$	$\approx 14E^{\dagger}$	$\approx 21E$
TX	[178] [5.3]	$1E$	$1E^{\dagger}$	$\approx 21E$
CG	[49] [5.4]	$2E + 4mE$	—	$\approx 13E$
KY	[119] [5.5]	$2E$	$2E^{\dagger}$	$\approx 17E$
AM	[12] [6.1]	$2E$	$5E^{\dagger}$	$\mathcal{O}(k)E$
FY	[94] [6.2]	$2E$	$\approx 3E^{\dagger}$	$\mathcal{O}(k)E$
BBS	[36] [7.1]	$3E + 1mE$	—	$\approx 20E + 10P$
CL	[57] [7.2]	$7E$	$6E + 1P^{\dagger}$	$\approx 20E + 8P$
BCNSW	[31] [7.3]	$2E$	$17E + 1P^{\dagger}$	$4E + 9P + 3mP$
BS	[38] [8.2]	$1E + 1mE$	—	$\approx 10E + 4P + 2mP$
NF	[146] [8.3]	$1E + 1mE + 1mtE$	—	$\approx 14E + 6P + 1mP$
BCNSW-VLR	[31] [8.4]	$2E$	$17E + 1P^{\dagger}$	$2E + 4P + m(2E + 5P)$

E — modular exponentiations; P — pairing evaluations; k — output length of a hash function.
m — number of group members; r — number of revoked members; t — number of time intervals.
† with further costs to verify the user-authenticated information prior to storing it in $\boldsymbol{reg}[i]$.

In the middle part we list two group signatures in the DL setting from Chapter 6, namely the Ateniese-de Medeiros (AM) and Furukawa-Yonezawa (FY) schemes, where the most expensive operation is also modular exponentiation. The amount of work in both schemes is quite similar. The opening procedure of both schemes scheme is very expensive, i.e. linear in the length of the security parameter, due to the use of a very inefficient signature of knowledge, which is part of the signature and has to be verified in the opening phase.

The bottom part of the table contains group signatures in the setting of bilinear maps from Chapter 7, including schemes with verifier-local revocation properties from Chapter 8. In this setting two operations are considered as expensive, namely modular exponentiations in input and target groups as well as the actual pairing operation, which is even more expensive. Note that modular exponentiations in this setting are actually scalar-point multiplications. We observe that none of the schemes evaluates pairings in the key generation procedure. The dynamic Camenisch-Lysyanskaya (CL) scheme requires much less work from the group manager to admit new group members and to open their signatures than the dynamic Bichsel-Camenisch-Neven-Smart-Warinschi (BCNSW) scheme or the static Boneh-Boyen-Shacham (BBS) scheme. In particular, the amount of work in the opening procedure of the BCNSW scheme grows linearly with the number of group members, whereas for the BBS and CL schemes it remains constant. In group signature schemes with the (T)VLR property, that is in the Boneh-Shacham (BS), Nakanishi-Funabiki (NF), and BCNSW-VLR schemes, the opening procedure has linear complexity in the total number of users, i.e. users that were admitted to the group and possibly revoked later. The reason is that in order to identify the signer while also ensuring that the signature is valid, the (implicit) opening procedure might need to test each ever issued revocation token. Therefore, the static BS scheme can be seen as the most efficient one in terms of the opening procedure since it requires a linear amount of modular exponentiations rather than pairing operations, yet this scheme does not implement VLR with time intervals to protect anonymity of revoked signers, which is offered by the NF scheme.

In practice, we may expect that the computations of the group manager are performed on some powerful device. Therefore, the amount of work of the group manager is not as important as the computational overhead imposed on the members of the group and on the verifiers of group signatures.

9.3.2. Computational Costs for Group Members and Verifiers

In Table 9.4 we provide efficiency comparison of the above mentioned schemes regarding the most important algorithms for the generation and verification of group signatures, while also considering the computational complexity of the member's part in the joining procedure (in dynamic schemes) and for the publicly executable judgement procedure (in schemes with verifiable opening).

Table 9.4.: Computational Costs for Members and Verifiers

Group Signature Scheme		JoinU	GSign	GVrfy	Judge
ACJT	[16] [5.1]	$\approx 11E^{\dagger}$	$\approx 19E$	$\approx 18E$	$\approx 22E^{\ddagger}$
TX	[178] [5.3]	$1E^{\dagger}$	$\approx 17E$	$\approx 16E$	$\approx 21E^{\ddagger}$
CG	[49] [5.4]	—	$\approx 10E$	$\approx 12E$	—
KY	[119] [5.5]	$0E^{\dagger}$	$\approx 13E$	$\approx 14E$	$\approx 19E^{\ddagger}$
AM	[12] [6.1]	$6E^{\dagger}$	$\mathcal{O}(k)E$	$\mathcal{O}(k)E$	$\mathcal{O}(k)E^{\ddagger}$
FY	[94] [6.2]	$\approx 5E^{\dagger}$	$\mathcal{O}(k)E$	$\mathcal{O}(k)E$	$\mathcal{O}(k)E^{\ddagger}$
BBS	[36] [7.1]	—	$12E + 5P$	$\approx 18E + 10P$	—
CL	[57] [7.2]	$2E^{\dagger}$	$16E + 4P$	$\approx 16E + 8P$	—
BCNSW	[31] [7.3]	$15E + 3P^{\dagger}$	$4E + 3P$	$2E + 5P$	$5E + 10P^{\ddagger}$
BS	[38] [8.2]	—	$\approx 7E + 2P$	$\approx 10E + 4P + 2rP$	—
NF	[146] [8.3]	—	$\approx 13E + 4P$	$\approx 14E + 6P + 1rP$	—
BCNSW-VLR	[31] [8.4]	$15E + 3P^{\dagger}$	$4E + 3P$	$2E + 5P + 3rP$	$5E + 10P^{\ddagger}$

E — modular exponentiations; P — pairing evaluations; k — output length of a hash function.
m — number of group members; r — number of revoked members; t — number of time intervals.
† with further costs to create user-authenticated information that the group manager stores in $\boldsymbol{reg}[i]$.
‡ with further costs to verify the group signature and the user-authenticated information from $\boldsymbol{reg}[i]$.

Amongst constructions in the RSA setting the most efficient signature generation can be performed with the static CG scheme. The dynamic KY scheme requires a trusted third party to be present during the joining protocol. Users do not need to perform heavy computations in the KY scheme during their admission process, as they receive their secret signing keys from this trusted party. In contrast, the joining protocol of the dynamic ACJT and TX schemes proceeds without third parties. Although we focus on the static variant of the CG scheme here, we observe that its dynamic extension would have comparable costs to the admission step of the ACJT scheme.

In the DL setting, for both schemes FY and AM the amount of expensive operations increases linearly with the length of the security parameter, in particular the amount of modular exponentiations is proportional to the output length of the hash function used in the signature of knowledge. This makes DL-based schemes clearly inefficient in comparison to schemes that use other cryptographic settings.

In the setting of bilinear maps the dynamic BCNSW scheme seems to have the most efficient signature generation and verification procedures, when compared to BBS and CL schemes. Note that BCNSW does not offer full anonymity though, yet it provides verifiable opening. Regarding schemes with VLR property, we make an observation that their signature generation procedure requires only constant amount of computation, while the amount of work for verification increases linearly with the number of revoked group members. Given that pairing evaluations are more expensive in comparison to modular exponentiations, we can safely assume that verification in the BS scheme becomes more efficient than verification in the NF and BCNSW-VLR schemes with the increasing number of revocations.

9.3.3. Costs and Impact of Key Generation

The key generation algorithm GKg is executed once by the group manager to create the group public key *gpk*, the group manager secret key *gmsk* and possibly – in case of static schemes – the group secret keys $\boldsymbol{gsk}[i]$. Being executed only once, the computational costs of this algorithm do not add significantly to the overall costs for the group manager. Obviously, the costs are much higher for static schemes, because they have to produce all the secret keys in this algorithm. But on the other hand, dynamic schemes add additional cost to the group manager via the join protocol. All static schemes require only a constant small number of modular exponentiations for the key generation, while with most of the dynamic schemes the amount of exponentiations necessary only grows linearly with the number of group members. The only notable exception is the NF scheme, which provides TVLR property and in which the amount of modular exponentiations also grows with the number of time intervals that are used to ensure backward unlinkability for revoked signers.

9.3.4. Costs and Impact of Admission Procedure

The admission process to the group is handled in dynamic group signature schemes through the join protocol Join, which is executed between the group manager and the prospective member. Therefore, its computational costs have to be considered for the group manager and for the joining member separately. The efficiency of the interactive algorithm JoinU on the group member's side becomes significant, if the user's device is constrained in its computational resources, which could be the case for many mobile devices. The costs of the interactive algorithm JoinM on the group manager's side has in turn less impact since these computations are typically performed on devices with rich computational resources or dedicated hardware. The computational complexity of admission procedures differ greatly among the different schemes and there is no apparent pattern regarding the cryptographic settings that are used. In general, schemes offering full non-frameability add additional costs to the JoinU algorithm, because they require the joining member to authenticate parts of the transcript in the join protocol to prevent framing attacks of possibly malicious group managers. A digital signature scheme would be typically used for this purpose. Note that these additional costs are not considered in our comparison table because they heavily depend on the authentication mechanism being used.

9.3.5. Costs and Impact of Group Signature Generation

The signature generation algorithm GSign is executed by a group member each time he signs a message. It is likely that each group member will execute this algorithm many times, possibly using resource-constrained devices, e.g. mobile phones or even smart cards. That is, computational costs of signature generation impose the highest impact on group members and in particular, are much more important than costs in the admission process. Computation complexity for signature generation in the RSA and bilinear map settings is constant, i.e. the number of expensive operations does not increase with the length of the security parameter or with the number of admitted group members. In practice, we can expect that signature generation in schemes that use bilinear groups is more efficient than in schemes that use RSA due to the shorter exponents and modulus sizes, as long as the number of more expensive pairing evaluations remains not too high. In particular, signing algorithms of CG, BCNSW, and BS schemes seem to have better efficiency in this respect.

9.3.6. Costs and Impact of Group Signature Verification

The signature verification algorithm GVrfy is public and can be executed by any third party, which is willing to check the validity of some given group signature. A group signature of some particular group member will typically be verified several times. That is, verification costs have also the highest impact on the efficiency of the group signature scheme. As with signature generation, schemes in the RSA setting and in bilinear groups seem to have more efficient verification procedures than DL-based constructions, and in practice, verification in schemes that use bilinear groups might be more efficient than in those that use RSA parameters. Observe that in all discussed group signature schemes, except for those that provide (T)VLR property (and the inefficient AM and FY schemes), verification costs do not depend on the size of the group. (T)VLR schemes commonly have a problem that revocation check, performed as part of the verification procedure, has to process each published revocation token $\boldsymbol{grt}[i]$ within some expensive operation, e.g. modular exponentiation in the BS scheme or (less efficient) pairing evaluation in the NF and BCNSW-VLR schemes.

9.3.7. Costs and Impact of Opening Procedure

The opening procedure Open is executed by the group manager (or opener in schemes with distributed authorities) whenever a signer of a group signature has to be identified. It is unlikely that Open will be executed too often since the actual idea of group signatures is to keep signers anonymous. We can expect that Open is executed as part of the punishment procedure, e.g. if a group member misused his signing rights, that in turn could lead to the revocation of these rights. In this case, high costs of the opening procedure would have only minor impact since the group manager has typically rich computational resources. Another observation is that group signature schemes with verifiable opening, i.e. ACJT, TX, KY, AM, FY, and BCNSW schemes, typically have higher computational costs due to the additional computation of proofs with regard to the output signer's identity. Moreover, the (implicit) opening procedure of (T)VLR schemes suffers from linear computational overhead in the number of admitted and possibly revoked group members.

9.3.8. Costs and Impact of Judgement Procedure

The judgement procedure Judge is publicly executable and is used to check whether the identification of the signer was performed by the group manager correctly. For some particular group signature, execution of Judge can be done only after the execution of Open. Therefore, it is likely that judgement procedure will be executed more often than the opening procedure and not necessarily on devices with rich computational resources. Nonetheless, high costs of the judgement procedure have only minor impact on the overall efficiency of the group signature scheme. In general the computational costs of the judgement procedure are lower than those of the opening procedure, especially for the schemes that use bilinear maps, in particular for schemes with (T)VLR property.

9.4. Space Complexity : Lengths and Impact of Private and Public Parameters

Another important indicator of the efficiency of a group signature scheme and its practical relevance is the *space complexity*, under which we subsume sizes and lengths of different public and private parameters used in the scheme. These lengths typically depend on the security parameter, which is in turn influenced by the applied cryptographic setting and evolves further over the time. That is why our comparison will be performed based on the amount of group elements in different cyclic groups that are needed to represent the components. Lengths and sizes of public and private parameters may vary across different schemes, due to the deployed cryptographic setting, offered functionality, and security properties of the scheme.

9.4.1. Overview

In Table 9.5 we give an overview of space complexity for various group signature schemes, indicating costs of its components, by measuring the number of needed group elements. We focus on several such components, namely on the group public key gpk, individual secret signing keys $\boldsymbol{gsk}[i]$, the length of group signatures σ, the group manager's secret key $gmsk$, and the storage costs for the revocation lists RL or update information $\boldsymbol{upd}$, depending on the underlying revocation mechanism of the scheme. In schemes with distributed authorities $gmsk$ contains elements for both the issuing key ik and the opening key ok. In some schemes that deploy complex signatures of knowledge we further estimate lengths of produced group signatures using appropriate heuristics. We also assume that hash outputs used in signatures of knowledge have roughly the same length as group elements (which is slightly in favor of the DL and bilinear map settings, whose group elements are typically shorter than elements in the RSA setting).

Table 9.5.: Space Complexity of Group Signature Schemes

Group Signature Scheme		gpk	$\boldsymbol{gsk}[i]$	σ	$gmsk$	$RL/\boldsymbol{upd}$
ACJT	[16] [5.1]	10	3	≈ 16	5	m
TX	[178] [5.3]	8	4	≈ 13	3	m
CG	[49] [5.4]	9	4	≈ 9	$1+m$	—
KY	[119] [5.5]	7	3	≈ 12	4	—
AM	[12] [6.1]	9	4	$\mathcal{O}(k)$	2	—
FY	[94] [6.2]	10	3	$\mathcal{O}(k)$	2	—
BBS	[36] [7.1]	6	2	9	$3+m$	—
CL	[57] [7.2]	8	4	11	7	—
BCNSW	[31] [7.3]	4	4	5	2	—
BS	[38] [8.2]	3	2	7	m	r
NF	[146] [8.3]	$3+t$	2	12	mt	r
BCNSW-VLR	[31] [8.4]	4	4	5	$2+m$	r

all lengths are given in group elements; k — output length of a hash function.
m — number of group members; r — number of revoked members.
t — number of time intervals.

The upper part of the table contains group signature schemes in the RSA setting from Chapter 5, namely the Ateniese-Camenisch-Joye-Tsudik (ACJT), Tsudik-Xu (TX), Camenisch-Groth (CG), and the Kiayias-Yung (KY) schemes. In this setting the length of a group element typically corresponds to the length of the RSA modulus N. We observe that the number of elements for each component of these three schemes is constant, except for the group manager's secret key *gmsk* in the CG scheme. The reason is that we consider a static variant of the CG scheme, where the group manager has to pre-compute individual secret signing keys for each group member and store them as long as they are not handed over to respective group members. For ACJT and TX schemes we observe that their revocation mechanisms require the group manager to publish update information ***upd***, which contains one group element per each revoked member. The KY scheme has the shortest group public key, while the CG scheme outputs shortest signatures.

In the middle part we list two group signatures in the DL setting from Chapter 6, namely the Ateniese-de Medeiros (AM) and Furukawa-Yonezawa (FY) schemes. In this setting the length of a group element corresponds to the length of the prime modulus used to define the group. We observe that while keys remain of constant size, the length of group signatures, although it does not depend on the number of group members, still depends on the output length of the hash function used in the generation of the signature of knowledge. From this perspective, their space complexity is clearly worse than the complexity of schemes that use RSA or bilinear maps.

The bottom part of the table contains group signatures in the setting of bilinear maps from Chapter 7, including schemes with verifier-local revocation properties from Chapter 8. These schemes rely on bilinear groups, that is group elements used to measure space complexity of these schemes can be either elements of the input groups $\mathbb{G}_1$ and $\mathbb{G}_2$, or elements of the target group $\mathbb{G}_T$, depending on the specification of the scheme. Our analysis lists the total number of group elements, without distinguishing between different groups. We remark, however, that in practice, most compact representation is available for Type 3 pairings (cf. Definition 3.8), in particular for the elements of the input group $\mathbb{G}_1$. Still, group elements in bilinear groups of prime order, used in all discussed group signature schemes, take typically less space in comparison to group elements in the RSA setting. We observe that also the number of group elements used for keys and signatures in group signature schemes in the bilinear map setting is typically lower in comparison to other settings. The Bichsel-Camenisch-Neven-Smart-Warinschi (BCNSW) scheme has currently shortest group signatures. In schemes with (T)VLR property the group manager has to additionally store revocation tokens for all admitted members of the group. In the NF schemes, which offers TVLR property this amount also depends on the number of time intervals representing the life time of the scheme. Due to the use of time intervals the group public key of the NF scheme grows linearly with the number of assumed intervals.

In practice not all components of a group signature scheme may have the same impact as we discuss in the following.

9.4.2. Length and Impact of Group Manager's Secret Keys

The group manager's secret key *gmsk* is known only to the group manager and is used to admit new group members and to open their group signatures. In static schemes *gmsk* typically contains individual secret signing keys $\boldsymbol{gsk}[i]$ that have to be computed in advance and then handed over to the corresponding members i. Once $\boldsymbol{gsk}[i]$ is handed over the group manager no longer needs to store it. That is, for static CG, BBS, BS, and NF schemes the effective length of *gmsk* can be obtained by removing the m group elements dedicated to individual secret signing keys of group members. Furthermore, group manager's algorithms will be typically executed on powerful devices. Therefore, the size of *gmsk* has only minor impact on the space complexity of the scheme. We observe that the effective length of *gmsk* in all group signature schemes is constant.

9.4.3. Length and Impact of Group Public Keys

The group public key *gpk* represents all the information required to verify the validity of generated group signatures and it is used in all algorithms of the scheme. Its length has, therefore, an important impact on the overall efficiency of the scheme. Clearly, it is desirable that the length of *gpk* remains constant, in particular it should be independent from the number of group members. This property is achieved by all considered group signature schemes, except for NF, where for each time interval there exists a special group element used in the verification procedure. In practice, *gpk* may be accompanied with additional PKI certificates to ensure binding between *gpk* and some particular group or organization.

9.4.4. Length and Impact of Secret Signing Keys

The secret signing key $\boldsymbol{gsk}[i]$ is typically known only to the corresponding group member i, who must store it (typically in some protected form) over the whole life time of the group membership. The length of $\boldsymbol{gsk}[i]$ may become an issue only if it is stored on some device with low storage resources, e.g. in a smart card or on some cryptographic hardware processor. We observe that all mentioned group signature schemes have secret signing keys composed of very few group elements.

9.4.5. Length and Impact of Output Group Signatures

As we noticed before, generation of group signatures and their verification belongs to the most frequently executed operations. Therefore, the length of the group signature σ (together with the length of gpk) is the most important parameter for the space complexity of the scheme. The AM and FY schemes in the DL setting have here a clear disadvantage. The number of group elements in their signatures corresponds to the number of bits used in the output of a hash function, which is needed to compute their signatures of knowledge. The most compact group signatures can be obtained today in the setting of bilinear maps, with the BCNSW scheme being most efficient in that respect.

9.4.6. Length and Impact of Revocation Lists and Public Update Information

Many group signature schemes allow the group manager to revoke the signing abilities of group members. In schemes with VLR property the group manager adds a revocation token $\boldsymbol{grt}[i]$ to the revocation list RL whenever i is revoked. In schemes where also signers need to update their secret signing keys, whenever some member gets revoked, the group manager has to publish appropriate update information $\boldsymbol{upd}$. Therefore, lengths of RL or $\boldsymbol{upd}$ should be considered as another important indicator for the space complexity of group signature schemes. In general, length of RL in VLR schemes (e.g. BS and BCNSW-VLR) increases proportionally to the number of revoked members; in TVLR schemes (e.g. NF) this length grows further with the number of time intervals in which some particular member is considered as revoked; in schemes that require $\boldsymbol{upd}$ (e.g. ACJT and TX) published information is linear in the total number of group members since $\boldsymbol{upd}$ is also updated with information about joining group members.

Part II.

Group Signatures in Practice

10. Schemes, Parameters, and Test Environment

Our theoretical analysis and comparison of existing group signature schemes, performed in the previous part of this work, offers a good indication on how certain schemes would perform in practice. In this part we extend this analysis and evaluate practical performance of selected schemes. We consider three group signature schemes that seem to offer a good balance between functionality and security, and have most promising relevance in practice. We evaluate the performance of the most frequently executed algorithms of these schemes by choosing appropriate security parameters and measuring timings for their costliest operations.

10.1. Selected Group Signature Schemes and Their Properties

In our performance analysis we will consider three group signature schemes:

- The dynamic group signature scheme by Camenisch and Groth(CG) [49] with full revocation support.
- The static group signature scheme by Boneh and Shacham (BS) [38] with verifier-local revocation.
- The dynamic group signature scheme by Bichsel et al. (BCNSW-VLR) [31] with verifier-local revocation.

As motivated through our comparison of reviewed group signature schemes in Chapter 9 the CG, BS, and BCNSW-VLR schemes seem to offer the better balance between efficiency, functionality, and security, in comparison to the other constructions, especially with their optional extensions. In this part, more precisely in Chapters 12 to 14, we will detail the specifications of those schemes by incorporating mechanisms that were previously considered as optional and by detailing the operations that were previously described in a black-box fashion. Note that we use the CG group signature scheme with its dynamic extension and full revocation support. Therefore, its description and properties may differ from those in Part I.

We summarize functionality and properties of the three schemes in Table 10.1. Observe that all schemes offer support for revocation. The two schemes CG and BCNSW-VLR can handle dynamic admissions of new group members. The BS scheme distributes generation of secret signing keys and handling of revocation amongst the two authorities (key issuer and group

Table 10.1.: Functionality and Properties of CG, BS, and BCNSW-VLR Schemes

Group Signature Scheme		Static	Dynamic	Verifiable Opening	Distributed Authorities	Revocation
CG	[49] [5.4]		✓			✓
BS	[38] [8.2]	✓			✓	VLR
BCNSW-VLR	[31] [8.4]		✓	✓		VLR

manager). The BCNSW-VLR scheme offers mechanisms to open group signatures and identify signers in a verifiable way.

In Table 10.2 we recall the different security properties offered by resulting specifications. Observe that dynamic CG and BCNSW-VLR schemes offer traceability against insiders, which is the strongest form of traceability dynamic schemes can offer. The BS and BCNSW-VLR schemes achieve insider anonymity, which offers a better trade-off between security and efficiency and is also sufficient for most applications. Notably all three schemes achieve the highest level of full non-frameability. The CG scheme works in the RSA setting, whereas BS and BCNSW-VLR require bilinear maps. Security properties of all three constructions were proven in the Random Oracle Model (ROM).

Table 10.2.: Security Properties of CG, BS, and BCNSW-VLR Schemes

Scheme		Setting	Anonymity	Traceab.	Non-Fra.	Assumptions
CG	[49] [5.4]	RSA	full	insider	full	ROM, DL, SRSA
BS	[38] [8.2]	BM	insider	full	full	ROM, q-SDH, DLIN, DL
BCNSW-VLR	[31] [8.4]	BM	insider	insider	full	ROM, LRSW, SDLP, DDH, DL

In Table 10.3, preceding the detailed specification of the scheme, we highlight their computation costs for the operations that should be executed by group members and verifiers, since computations on the group manager's side are less relevant in practice and are likely to be executed on powerful computing devices, e.g. servers. Our analysis of practical performance, however, will further be limited to the timings of the signature generation GSign and verification GVrfy algorithms. These are the most frequently used algorithms and their performance has the most significant impact on the schemes' practicality. In the CG scheme we will consider modular exponentiations, whereas in BS and BCNSW-VLR schemes we will refine the costs by considering pairing operations and exponentiations in the three different algebraic groups (input and target groups) that define the applied bilinear map setting. The signing procedure in all three constructions requires constant amount of work. However, since all three schemes support revocation, the running time of their verification algorithms remains linear in the number of revoked users, which in turn raises further scalability questions. The linear amount of work is also present in the Judge algorithm provided by the BCNSW-VLR scheme due to its implicit use of the verification procedure. The joining procedure is only offered by

CG and BCNSW-VLR schemes, whereby JoinU is the corresponding (interactive) algorithm on the member's side. However, a member performs JoinU only once during its admission and the costs of that algorithm become less significant in practice.

Table 10.3.: Computational Costs for Members and Verifiers in CG, BS, and BCNSW-VLR Schemes

Group Signature Scheme		JoinU	GSign	GVrfy	Judge
CG	[49] [5.4]	$10E$	$13E$	$13E + rE$	—
BS	[38] [8.2]	—	$8E + 3P$	$8E + 6P + rP$	—
BCNSW-VLR	[31] [8.4]	$15E + 3P$	$4E + 3P$	$2E + 5P + rP$	$3E + 5P + \text{cost}_{\mathsf{Vrfy}}$ $+\text{cost}_{\mathsf{GVrfy}}$

E — modular exponentiations; P — pairing evaluations; r — number of revoked members.
$\text{cost}_{\mathsf{Vrfy}}$ — costs of (ordinary) signature verification; $\text{cost}_{\mathsf{GVrfy}}$ — costs of GVrfy.

As part of our performance analysis and comparison of the three schemes we also consider the required amount of space to store the different parameters of these schemes. Of prime interest for practice are the lengths of group public keys gpk, individual secret signing keys $\boldsymbol{gsk}[i]$, output group signatures σ, and the amount of revocation information that has to be published by the group manager, i.e., the length of revocation lists RL in VLR schemes and the length of the public update information $\boldsymbol{upd}$ in the CG scheme. On the group manager's side we can count the length of the group manager's secret key $gmsk$, including the size of the (initially secret) registration information $\boldsymbol{reg}$, which is used by the group manager to identify the signers.

We observe that for the purpose of better protection it might be desirable to store secret keys in some tamper-resistant cryptographic hardware chip. Whether this is possible or not will certainly depend on the resulting length of those keys. In Table 10.4 we give an initial overview of the mentioned lengths, measured in the number of components (e.g. group elements). We will refine this table in our analysis later by measuring the required storage complexity in bits, depending on the concrete choice of security parameters for the utilized cryptographic settings. The selection process of appropriate security parameters is discussed in the next section.

Table 10.4.: Space Complexity of CG, BS, and BCNSW-VLR Schemes

Group Signature Scheme		gpk	$\boldsymbol{gsk}[i]$	σ	$gmsk$	$RL/\boldsymbol{upd}$
CG	[49] [5.4]	11	6	11	$1 + m$	$2r$
BS	[38] [8.2]	3	2	7	m	r
BCNSW-VLR	[31] [8.4]	4	4	5	$2 + m$	r

m — number of group members; r — number of revoked members.

10.2. Choice of Security Parameters

10.2.1. General Overview

Security of group signature schemes is proven in the computational setting (cf. Section 1.5) in dependency of the (global) *security parameter* 1^κ, $\kappa \in \mathbb{N}$. In practice κ is a *bit-length* and remains valid for a certain period of time, assuming that existing cryptanalytic potential does not experience a significant "break-through" in that period. This bit-length varies for different cryptographic settings and its estimation takes into account existing lower bounds of cryptanalytic algorithms.

The estimation of the security parameter is often performed using heuristics that further take into account possible progress in technology, e.g., increase of computing power, often measured in "million instructions per second" over a year of computation (MIPS year), yet also criticized for not offering exact estimates [172], and the actual cost of the cryptanalytic attack, i.e. costs for building a special-purpose architecture that would solve the basic hard problem behind the cryptographic setting (e.g., factorization of RSA moduli or computation of discrete logarithms).

Security level of cryptographic settings based on hard problems from number theory are often defined in relation to the security level of symmetric primitives, where security parameter represents the acceptable bit-length of secret keys to withstand brute force attacks. More generally, a *κ-bit security level* for some cryptographic operation is given for some period of time and typically assumes that it is practically infeasible to perform 2^κ executions of that operation with computing power that will become available during that period. A lot of work in the literature, e.g., [128, 127, 104, 92], has been done on establishing the heuristics for the appropriate choice of security parameters in different cryptographic settings. Meanwhile, various agencies [152, 153, 10, 32] and research groups [87] provide their own recommendations for the concrete choice of security parameters and their validity periods, sometimes also distinguishing amongst different levels of protection such as "secret" or "top secret" [152]. A nice overview of available recommendations with an interactive tool to assist in the selection process can be found at `www.keylength.com`.

In our performance measurements of the three mentioned group signature schemes we will mostly rely on recommendations of the German Federal Network Agency (BNetzA, `www.bundesnetzagentur.de`) and German Federal Office for Information Security (BSI, `www.bsi.bund.de`), and of the European Network of Excellence in Cryptography (ECRYPT, `www.ecrypt.eu.org`). These recommendations are summarized in Table 10.5 and differ mainly in the length of the RSA modulus, i.e. the ECRYPT recommendation suggests modulus length of 3248 bits in comparison to 2048 bits recommended by BNetzA/BSI. We stress that the indicated lengths correspond to security levels of 100-bits [32] resp. 128 bits [87], which is generally believed to offer sufficient protection in the mid-term.

That is, for the RSA modulus $N = pq$ we will provide measurements with respect to $|N| = 2048$ bits and $|N| = 3248$ bits. Since the cryptographic setting of bilinear maps currently uses instantiations that rely on elliptic curve cryptography, at least 250 bit-length must be chosen for the prime order of bilinear groups. Since hash functions are often used in the construction of group signatures, we will assume that the typical length of a hash value is 256 bits. This general set of parameters will allow us to analyze and compare performance and scalability of

Table 10.5.: Recommended Parameters for 100- to 128-bit Security

Agency	RSA Setting	DL Setting		Elliptic Curves	Hash
	Modulus Size	Order $\mathbb{Z}_P^*$	Order $\mathbb{G} \subset \mathbb{Z}_P^*$	Order $\mathbb{G}$	
BNetzA/BSI [32]	2048 bits	2048 bits	256 bits	250 bits	256 bits
ECRYPT [87]	3248 bits	3248 bits	256 bits	256 bits	256 bits

the mentioned group signature schemes at the assumed security level of 100 resp. 128 bits.

10.2.2. Security Parameters for $QR(N)$ Groups

The group signature scheme by Camenisch and Groth (CG) [49] uses the group of *quadratic residues* modulo a safe RSA modulus N. This group is denoted by $QR(N)$. An appropriate safe RSA modulus N of length κ can be generated as the product of two safe prime numbers p and q, each of bit-length $\lfloor \kappa/2 \rfloor$, whereby $p = 2p' + 1$ and $q = 2q' + 1$ with p', q' being two randomly chosen prime numbers, each of bit-length $\lfloor \kappa/2 \rfloor - 1$. In order to choose an appropriate generator g of the $QR(N)$ group one first picks some random integer $a \in \mathbb{Z}_N^*$ satisfying condition $gcd(a \pm 1, N) = 1$ and defines $g = a^2 \mod N$. The order of the $QR(N)$ group is given by the product $p'q'$. In the CG scheme modular exponentiations $g^x \mod N$ are mostly performed with exponents x, whose length can be approximated with $\lfloor \kappa/2 \rfloor$ bits.

In our measurements we will use two different sizes of RSA moduli — 2048 bits and 3248 bits — to match the recommendations from [32, 87]. By fixing the modulus size we implicitly define lengths of further security parameters behind the $QR(N)$ setting of the CG scheme. The resulting set of parameters is summarized in Table 10.6.

Table 10.6.: Parameter Lengths in $QR(N)$ Setting

Parameter	Lengths	
modulus N	2048 bits	3248 bits
prime factors p, q	1024 bits	1624 bits
generator, element of $QR(N)$	2048 bits	3248 bits
exponent x	1024 bits	1624 bits

10.2.3. Security Parameters for $\mathbb{Z}_P^*$ Groups

Another group used in the algorithms of the Camenisch-Groth (CG) scheme [49] is a cyclic subgroup of integers $\mathbb{Z}_P^*$ of prime order Q, where the length of Q defines the security parameter length κ, and P is a prime number. The CG scheme uses this group in combination with $QR(N)$ and assumes that parameters of both groups are chosen such that factoring N is as hard as the computation of discrete logarithms in $\mathbb{Z}_P^*$.

Following the recommended parameter lengths, we will assume that $|P| = |N|$ and perform two types of measurements for the security level of 128 bits: our first measurements will use

Table 10.7.: Parameter Lengths in $\mathbb{Z}_P^*$ Setting

Parameter	Lengths	
modulus P	2048 bits	3248 bits
generator, element of $\mathbb{Z}_P^*$	2048 bits	3248 bits
exponent x	256 bits	256 bits

$|P| = 2048$ bits and $|Q| = 256$ bits, while our second measurements will assume that $|P| = 3248$ bits and $|Q| = 256$ bits. Note that the length of the prime order Q implicitly defines the length of the typical exponent $x \in \mathbb{Z}_Q$ in a modular exponentiation operation $g^x \mod P$. The resulting bit-lengths of $\mathbb{Z}_P^*$ parameters used in our measurements are summarized in Table 10.7.

10.2.4. Security Parameters for Bilinear Groups with Type-2 Pairing

The group signature schemes of Boneh and Shacham (BS) [38] and Bichsel et al. (BCNSW-VLR) [31] can be implemented over bilinear groups $(\mathbb{G}_1, \mathbb{G}_2)$ that provide a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ of Type-2. Note that groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are called input groups, whereas $\mathbb{G}_T$ is the target group. All three groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ have the same prime order Q of length κ. The generator of $\mathbb{G}_T$ is given by $g_T = e(g_1, g_2)$, where g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Bilinear maps of Type-2 assume existence of an efficiently computable homomorphism $\psi : \mathbb{G}_2 \mapsto \mathbb{G}_1$ with $g_1 = \psi(g_2)$ (cf. Definition 3.8). Despite the same order, the three groups differ in the representation of their elements. In the case of Type-2 pairings – as used in modern cryptography – the first input group $\mathbb{G}_1$ is typically defined as a subgroup of points of an elliptic curve E over a (prime or binary) finite field $\mathbb{F}_q$, $q \in \mathbb{N}$ a power of a prime, that is $\mathbb{G}_1 \subset E(\mathbb{F}_q)$. The second input group $\mathbb{G}_2$ is given as a subgroup of $E(\mathbb{F}_{q^k})$, which denotes the group of points of E over an extension field $\mathbb{F}_{q^k}$, for some *embedding degree* $k \in \mathbb{N}$. It is important to notice that exponentiations g_1^x, g_2^x, and g_T^x depend on the actual group operation in the respective input or target group.

Parameters of the cryptographic setting in Type-2 bilinear maps are chosen such that the Discrete Logarithm problem in groups $E(\mathbb{F}_q)$, $E(\mathbb{F}_{q^k})$, and $\mathbb{F}_{q^k}^*$ is assumed to be hard. In particular, under existing cryptanalytic techniques, if Q is the order of the largest prime-order subgroup of $E(\mathbb{F}_q)$ then the size q^k of the extension field must be significantly larger than Q. With the increasing size of q^k, or alternatively, with the increasing embedding degree, computations in $\mathbb{G}_2$ and $\mathbb{G}_T$ become slower and their elements require more space for representation. It is challenging to find *pairing-friendly* curves E that would admit small embedding degree k in combination with the sufficiently large prime-order subgroup in $E(\mathbb{F}_q)$. We observe that a typical embedding degree k used in current pairing-based cryptographic settings ranges between 1 and 12. At this point we refer the reader to [92] for a detailed exposition of different parameters and techniques related to the choice of suitable parameters for cryptographic use of bilinear maps. We remark that pairing-based cryptography is still an ongoing area of research and that the entire potential behind efficient implementations of cryptographic algorithms in this setting has not been explored yet. This also means that our measurements with regard to the group signature schemes from [38, 31] that we will perform based on existing libraries for pairing-based cryptography should not be taken as a benchmark.

In our analysis, we will use specially chosen parameters, as summarized in Table 10.8. These

Table 10.8.: Parameter Lengths in Type-2 Bilinear Group with $k = 6$

Parameter	Length
size of $q = \|\mathbb{F}_q\|$	347 bits
size of $q^k = \|\mathbb{F}_{q^k}\|$	2082 bits
size of group order Q	332 bits
size of elements of $\mathbb{G}_1$	360 bits
size of elements of $\mathbb{G}_2$	1064 bits
size of elements of $\mathbb{G}_T$	2112 bits

parameters reflect the view from [92] for the case where $\kappa = |Q| = 332$ and embedding degree $k = 6$. The size of the underlying finite field $\mathbb{F}_q$ is $|q| = 347$ bits. We motivate our choice with the availability of a pairing-friendly curve with such parameters in the open source library for pairing-based cryptography that will be used in our practical tests (cf. Section 10.3.2).

In Table 10.8 we reproduce the sizes of elements in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ as reported by this library. The reason why these lengths do not perfectly coincide with the expected values (e.g., element size in $\mathbb{G}_1$ is 360 bits instead of expected 347 bits) is that, when serializing the internal representation of elements to a sequence of full bytes, the library introduces a marginal padding between 8 and 15 bits. Moreover, for the specific curve that we chose, elements of $\mathbb{G}_1, \mathbb{G}_2$ are stored in a compressed form that saves roughly 50% (75%) of storage space (precisely: elements of $\mathbb{G}_1$ are encoded in about q bits instead of $2q$ bits, and elements of $\mathbb{G}_2$ are encoded in about $3q$ bits instead of $2kq = 12q$ bits).

10.3. Test Environment, Utilities, and Methodology

10.3.1. Reference Platforms

In order to estimate the expected performance of the three group signature schemes in practice several experiments were performed. These experiments involved two reference platforms that differ in their characteristics as detailed in the following.

Reference platform "PC". Our first reference platform, denoted for simplicity as "PC", is a single core of a Lenovo X60s laptop, equipped with an Intel Core Duo LV L2400 1.66 GHz processor. The libraries and benchmarks for this platform are compiled with the *GCC* [99] in its version 4.4.4 and 32-bit mode.

Reference platform "Smartphone". Our second reference platform, denoted for simplicity as "Smartphone", is the small handheld device HTC Desire, equipped with an 1 GHz Scorpion processor (Qualcomm QSD8250 Snapdragon chipset). The libraries and benchmarks for this platform are cross-compiled with the *GCC* [99] in its version 4.4.3, using the *arm-linux-androideabi-gcc* of the Android NDK r6b [9] for the ARM instruction set. The used smartphone runs Android 2.2 with Kernel 2.6.32.15-g2633d94.

10.3.2. Utilized Libraries

In order to estimate practical performance of group signature schemes from [49, 38, 31] the reference platforms were used to measure timings of the most significant cryptographic operations that appear in the corresponding algorithms of those schemes.

For this purpose we utilize several libraries, freely available on the Internet. In particular, big integer operations are implemented using the optimized *GNU Multi-Precision Library* (GMP) [101] in its version 4.3.2 for our PC reference platform and in version 5.0.2 for our Smartphone platform. For the evaluation of operations in the setting of bilinear maps we deployed Stanford University's *Pairing-Based Cryptography* (PBC) library [157] in version 0.5.12, which links against the GMP library to perform the required big integer operations and is often used for "proof of concept" implementations of cryptographic applications. Both libraries as well as our test routines are written in C and/or Assembler.

10.3.3. Test Methodology and Heuristics

Our analysis of performance and scalability of the three group signature schemes is based on the following methodology that deploys several heuristics. First, we focus mainly on algorithms with the highest practical relevance, namely on the signature generation and verification procedures, whereby for the latter we also include performance analysis for identifying signatures produced by revoked group members. As motivated in Section 9.3 the signing and verification procedures belong to the most frequently executed algorithms and their performance counts as the most dominant factor in the practical deployment of group signature schemes. We nonetheless briefly evaluate performance of the remaining algorithms for group management and (verifiable) opening that are executed less often than signing and verification procedures.

Further, our performance analysis does not make use of complete implementations. Instead, we perform a preliminary analysis in Chapter 11, where we specify dominant operations that are used in the algorithms of the three schemes and measure their timings on our reference platforms. For pairing-based group signatures we also take into account state-of-the-art timings that have been reported in the literature, for which, however, no implementation details were disclosed into the public domain or documentation was not sufficient to reproduce the results. The additional use of literature sources is meaningful since existing open-source libraries for cryptographic pairings are not optimal and admit mostly high-level "proof of concept" implementations. We will apply timings from the literature to estimate the resulting costs of the most frequent signing and verification procedures in each of the three group signature schemes in a heuristic way; namely, by counting the total amount of dominant cryptographic and big integer operations in those algorithms and by calculating their resulting execution time.

We consider our heuristic approach as suitable for two main reasons: first, expensive cryptographic operations, such as modular exponentiations and pairing evaluations, admit many optimization strategies, speeding up their performance. The optimization potential depends on the context in which the operations are executed and may further be triggered by the actual application in which the cryptographic scheme is deployed. Our heuristic approach does not consider computational gains that may result from any optimized implementations of the deployed operations; although, along the lines, we mention some optimization strategies and provide corresponding references. On the other hand, our approach does not take into account the additional costs, contributed by other big integer operations such as additions and multiplications or by cheap cryptographic operations such as hash functions that are used in the signing and verification procedures. These costs, however, would introduce an additional overhead and by this decrease the gain obtained from optimization. More importantly, our heuristic approach seems fully sufficient to answer the central question of this work, namely whether the state-of-the-art group signatures are amenable to practical use with modern computing technology.

11. Dominant Operations and Measured Timings

The group signature schemes from [49, 38, 31] require different cryptographic settings: The CG scheme [49] combines $QR(N)$ and $\mathbb{Z}_P^*$ groups, whereas BS [38] and BCNSW-VLR [31] require pairings, whereby Type-2 pairing is mandatory for the BS scheme, while BCNSW-VLR scheme may also be implemented with Type-3 pairings. In the following we describe our time measurements for the most dominant operations in $QR(N)$, $\mathbb{Z}_P^*$, and bilinear groups of Type-2. These timings will be used as a basis for the estimation of running times of the most significant algorithms of the three group signature schemes. We remark that timings were measured on our reference platforms using test routines in which each operation was performed sufficiently many times to exclude variances. Moreover, each operation was tested with randomly generated instances (e.g. base elements and exponents) and the amount of repetitions was chosen such that generation of these random instances had only a negligible impact on the resulting time per operation.

11.1. Computation Costs in $QR(N)$ Groups

The most dominant operation in $QR(N)$ groups is modular exponentiation. In Table 11.1 we provide time measurements of this operation. These timings will be used later to evaluate performance of the CG scheme. They were measured on our reference platforms, PC and smartphone, using the test source code written in C and compiled with GCC. The numbers represent the average runtime of one modular exponentiation in a $QR(N)$ group with a randomly chosen but fixed safe RSA modulus N and multiple randomly chosen base elements g and exponents x of the given length. To measure an appropriate average runtime per operation, the runtime of 200 (for 2048-bit modulus) resp. 50 (for 3248-bit modulus) exponentiations have been measured and averaged. The amount of 200 resp. 50 repeated exponentiations turned out to be sufficient to prevent variances on our reference platforms.

Table 11.1.: Timings of Operations in $QR(N)$ Groups

Operation	Modulus	Exponent	Time (PC)	Time (Smartphone)
modular exponentiation	2048 bits	1024 bits	24.0 ms	60.7 ms
($g^x \mod N$)	3248 bits	1624 bits	125.2 ms	214.7 ms

The algorithms of the CG scheme sometimes use *multi-base exponentiations* in $QR(N)$ groups of the form $g_1^{x_1} \cdots g_n^{x_n}$ with different bases g_i and exponents x_i. These operations bear further optimization potential to reduce the execution time in comparison to n single-base exponentiations $g_i^{x_i}$ for all $i \in [1, n]$. Another optimization potential arises in case where several exponentiations have to be performed using the same base g. In this case there exist efficient algorithms that use pre-computation to further speed-up computation of multiple exponentiations with the same base. Here we refer to the survey by Gordon [102] on available algorithms for optimizing the computation of modular exponentiations in different cryptographic settings. However, we will not consider any of these optimization in our analysis, as the expected improvement would not be significant for a more general statement about the practicality of the CG scheme.

11.2. Computation Costs in $\mathbb{Z}_P^*$ Groups

The most dominant operation in $\mathbb{Z}_P^*$ groups is modular exponentiation, the exponent however is shorter than in the $QR(N)$ setting. In Table 11.2 we provide time measurements of this operation. These timings will be combined with those for $QR(N)$ groups and used later to evaluate performance of the CG scheme. They were measured on our reference platforms, PC and smartphone, using the test source code written in C and compiled with GCC. The numbers represent the average runtime of one modular exponentiation in a $\mathbb{Z}_P^*$ group with a randomly chosen but fixed prime modulus P and multiple randomly chosen base elements g and exponents x of the given length. To measure an appropriate average runtime, the runtime of 500 (for 2048-bit modulus) resp. 300 (for 3248-bit modulus) exponentiations have been measured and averaged. The amount of 500 resp. 300 repeated exponentiations turned out to be sufficient to prevent variances on our reference platforms.

Table 11.2.: Timings of Operations in $\mathbb{Z}_P^*$ Groups

Operation	Modulus	Exponent	Time (PC)	Time (Smartphone)
modular exponentiation	2048 bits	256 bits	6.3 ms	16.0 ms
($g^x \mod P$)	3248 bits	256 bits	21.0 ms	35.4 ms

We remark that $\mathbb{Z}_P^*$ group admits the same optimization tricks with regard to multi-base exponentiations $g_1^{x_1} \cdots g_n^{x_n}$ and multiple exponentiations with the same base g as mentioned in the previous section. Again the expected improvement would not be significant for a more general statement about the practicality of the CG scheme and so we will not consider any of these optimization in our analysis.

11.3. Computation Costs in Bilinear Groups with Type-2 Pairings

The reference platform was further used to measure timings of most significant operations in the setting of bilinear groups $(\mathbb{G}_1, \mathbb{G}_2)$ with a Type-2 pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$. These measurements were performed for the selected pairing-friendly curve and bilinear groups. In addition to the actual evaluation of the bilinear map e, significant costs arise for exponentiation operations in the input groups $\mathbb{G}_1$ and $\mathbb{G}_2$ and in the target group $\mathbb{G}_T$. Although all groups have the same prime order Q, the representation of the respective group elements and the actual specification of the exponentiation operation differs from group to group. In particular, exponentiation costs in $\mathbb{G}_2$ are much higher in comparison to those in other groups, whereby exponentiations in $\mathbb{G}_1$ are most efficient. Combined with the fact that elements in $\mathbb{G}_1$ have the shortest representation, it is desirable for the group signature scheme to mitigate its computation and storage costs such that the costly operations and signature components can be accounted to $\mathbb{G}_1$ rather than to other groups. In Table 11.3 we summarize our measurements for the operations that will be later used to evaluate performance of the BS and BCNSW-VLR schemes.

Table 11.3.: Timings of Operations in Bilinear Groups with Type-2 Pairing

Operation	Element Size	Exponent	Time (PC)	Time (Smartphone)
exponentiation in $\mathbb{G}_1$	360 bits	332 bits	10.7 ms	24.0 ms
exponentiation in $\mathbb{G}_2$	1064 bits	332 bits	92.6 ms	297.4 ms
exponentiation in $\mathbb{G}_T$	2112 bits	332 bits	24.5 ms	80.7 ms
pairing evaluation $e(\cdot,\cdot)$	—	—	91.8 ms	283.2 ms

These measurements were obtained on our reference platforms, PC and smartphone, using the PBC library [157], which provides routines for elliptic curve generation, elliptic curve arithmetic, and pairing computations. This library is written in C and links against the GMP library [101]. The PBC library offers implementations of different bilinear map settings, covering both symmetric (i.e., $\mathbb{G}_1 = \mathbb{G}_2$) and asymmetric (i.e., $\mathbb{G}_1 = \mathbb{G}_2$) pairings. Specifically, we opted for (asymmetric) *MNT curves* discovered by Miyaji, Nakabayashi, and Takano [144]. These curves have the embedding degree $k = 6$ and thus offer a good balance between performance and size of element representation.

The PBC library provides implementation of MNT curves. For our measurements we used a concrete MNT curve, denoted `D476971` in the PBC classification. This curve is defined over a prime field $\mathbb{F}_q$ for a 347-bit prime number q. Considering the size of the prime field $\mathbb{F}_q$ and the embedding degree of the elliptic curve, the target group $\mathbb{G}_T$ is a subset of $\mathbb{F}_{q^6}$, which in turn is a field of approximately 2082 bits. See also Section 10.2.4 for a discussion on the element sizes for groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, as reported by PBC and reflected in Table 11.3. We measured timings on our reference platforms, PC and smartphone, using the test source code written in C and compiled with GCC. Timing entries in Table 11.3 represent the average runtime of the performed operations using the above MNT curve with multiple randomly chosen base elements $g_1 \in_R \mathbb{G}_1$, $g_2 \in_R \mathbb{G}_2$, $g_T \in_R \mathbb{G}_T$, and exponents x of the appropriate length. These timings correspond to the average runtime of 1000 (in $\mathbb{G}_1$) resp. 100 (in $\mathbb{G}_2$) resp. 400 (in $\mathbb{G}_T$) exponentiations and 100 pairing evaluations. These different amounts of repetitions for each operation turned out to be sufficient to remove variances on our reference platforms.

We observe that implementations in the PBC library are not necessarily optimized and that the library does not include support for all types of curves and parameters that can be found in the literature. Therefore, our timings obtained with this library should also be used with care. In particular, computations behind the pairing-based BS and BCNSW-VLR schemes, when measured with other possibly optimized implementations of pairings, are likely to result in a significantly better performance gain. However, it is still helpful to compare estimated costs for these group signature schemes based on non-optimized, yet freely available pairing implementations with estimated costs of the CG group signature scheme on the same reference platform. Nonetheless, in our performance analysis of the BS and BCNSW-VLR schemes we will also use timings obtained from other sources as we will discuss in the next section.

Finally, note that in some cases exponentiations in the input and target groups can further be optimized. In particular, faster exponentiation algorithms for multi-base exponentiations and for multiple exponentiations with the same base elements via pre-computation can be applied in all three groups. In addition to the survey by Gordon [102], we refer to the book of Hankerson, Menezes, and Vanstone [106] for the description of various exponentiation algorithms in elliptic curve cryptography that can also be applied to the input pairing groups. In addition, the bilinearity property of the pairing operation $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ (cf. Definition 3.7) admits further optimizations in the computation of pairing products. For example, the product of pairings $e(g_1, g_2^{x_1})e(g_1, g_2^{x_2}) \cdots e(g_1, g_2^{x_n})$ can be computed at the cost of only one pairing operation as $e(g_1, g_2^{x_1} \cdots g_2^{x_n})$.

11.3.1. Timings of Type-2 Pairing Evaluations in the Literature

Since the open-source PBC library does not necessarily reflect the state-of-the-art in efficient implementation of pairings, it is advisable to also address implementations and their timings based on other sources. In Table 11.4 we provide an overview of time measurements for various pairing computations from the existing literature. These measurements have been performed on different computing platforms and in various contexts such as hardware-based timings on FPGAs or software-based timings on PCs and in smartcards. In this overview we focus on timings for the security level of (approximately) 128 bits to achieve comparability with the PBC library-based measurements on our reference platforms. The only exception are the listed measurements on smartcards that are available for the reduced security level of 80 bits. In general, all timings mentioned in Table 11.4 were performed using optimal ate pairings, except for the results of Scott et al. [168] on smartcards, for which regular ate pairings have been used.

The fastest pairing evaluations were measured on FPGAs, independently in the work of Duquesne and Guillermin [86] and by Yao et al. [187], where timings of 1 ms or even less are possible. The majority of timing results on PCs lie in the range between 1 ms and 10 ms. These measurements were partially carried out as a comparison to hardware-based implementations. Observe, however, that measurements on PCs were made on computing platforms with a much higher clock speed, in comparison to the rates that are feasible on FPGAs. Therefore, it is evident (and not really surprising) that hardware-based implementations of pairing operations outperform their implementations in software. The measured time of about 50 ms for a pairing evaluation in the work of Acar et al. [7] is especially interesting when looking at the rising popularity of smartphones. Their measurements were performed on the ARM architecture that prevails today in embedded and mobile devices. With a significant loss in performance, far behind the measurements on FPGAs and PCs, follow the timing results on smartcards. In particular, pairing evaluations by Scott et al. [168] were performed using Phillips HiPerSmart$^{\text{TM}}$, which is an instantiation of the SmartMIPS™architecture, and take between 380 ms and over one second, depending on the processor's clock speed.

Table 11.4.: Timings of Type-2 Pairing Evaluations from the Literature

Platform	Source	Context	Security	Time
Xilinx Virtex-6 XC6VLX240T-2 @ 250 MHz	[187]	FPGA	≈128 bits	0.66 ms
Stratix III (EP3SE50) @ 165 MHz	[86]	FPGA	128 bits	1.15 ms
Stratix II (EP2S30) @ 154 MHz	[86]	FPGA	128 bits	1.23 ms
Altera Cyclone II (EP2C35) @ 91 MHz	[86]	FPGA	128 bits	2.09 ms
AMD Phenom II X4 955 @ 3.2 GHz	[145]	PC	≈128 bits	1.55 ms
Intel Core 2 Quad Q9550 @ 2.8 GHz	[145]	PC	≈128 bits	1.58 ms
Intel Core 2 Quad Q6600 @ 2.4 GHz	[145]	PC	≈128 bits	1.86 ms
Intel Xeon E5504 @ 2.0 GHz	[145]	PC	≈128 bits	2.37 ms
AMD Athlon 64 X2 3800+ @ 2.0 GHz	[145]	PC	≈128 bits	5.48 ms
Intel Core 2 E6600 @ 2.4 GHz (64bit)	[7]	PC	128 bits	6.30 ms
Intel Core 2 E6600 @ 2.4 GHz	[7]	PC	128 bits	11.72 ms
Dual-Core Cortex A9 ARM @ 1.0 GHz	[7]	PC	128 bits	54.19 ms
Phillips HiPerSmart™@ ˜36 MHz	[168]	Smartcard	80 bits	380.00 ms
Phillips HiPerSmart™@ 21 MHz	[168]	Smartcard	80 bits	590.00 ms
Phillips HiPerSmart™@ 9 MHz	[168]	Smartcard	80 bits	1210.00 ms

12. Specification and Performance of the Camenisch-Groth Scheme

12.1. Detailed Specification of the Camenisch-Groth Scheme

In this section we specify the core algorithms and protocols of the dynamic Camenisch-Groth (CG) group signature scheme with full revocation support. Our description follows the specification from [49].

We start with the description of the security parameters deployed in the CG scheme and provide recommendations for their lengths. The CG scheme uses the following lengths: ${}_c$, ${}_e$, ${}_s$, ${}_E$, ${}_Q$, ${}_N$, ${}_P$, all of which depend polynomially on the security parameter κ. In addition, these lengths must satisfy the following conditions:

- Let $\mathsf{Hash}_1 : \{0,1\}^* \to \{0,1\}^{\ell_c}$ and $\mathsf{Hash}_2 : \{0,1\}^* \to \{0,1\}^{\ell_c}$ denote hash functions,
- for every integer a, an integer r of the length $|a| + {}_s$ can be chosen randomly, so that $a + r$ and r are statistically indistinguishable,
- ${}_e$ is large enough to assign different numbers to all members and to make E_i prime, and
- the following two relations must hold:

$$ {}_c + {}_e + {}_s + 1 < {}_Q \quad \text{and} \quad {}_c + {}_Q + {}_s + 1 < {}_E < {}_N/2. $$

These conditions stem partly from the specification of the Camenisch-Lysyanskaya signature scheme from [55, 133], which is used by the algorithms of the CG scheme. The length of the safe RSA modulus N, denoted ${}_N$ determines the hardness of the factorization problem in the $QR(N)$ group, which is used in the CG scheme, and is therefore fixed to 2048 bits resp. 3248 bits according to the desired security levels, following the recommendations of BNetzA/BSI [32] resp. ECRYPT [87]. Since the hardness of the factorization problem of an ${}_N$-bit RSA modulus and the hardness of the discrete logarithm problem in subgroups of $\mathbb{Z}_P^*$ should be kept at the same level, the length of the prime modulus $|P|$, denoted by ${}_P$, should also be fixed to 2048 bits resp. 3248 bits.

The lengths ${}_e$ and ${}_Q$ result from further constraints put on the Camenisch-Lysyanskaya signature scheme. Here the length ${}_e$ must ensure that the group manager can choose an independent prime number E_i for each prospective group member i. On the one hand, ${}_e$

should be set such that the appropriate choice of E_i becomes feasible even for very large groups (e.g., several millions of members), and on the other hand, $_e$ must be sufficiently large so that the probability of guessing an E_i becomes negligible. To satisfy both conditions $_e$ must be set to at least 80 bits.

The length $_s$ stands for the parameter that defines statistical closeness in utilized NIZKPoK proofs; this parameter, too, must be set to at least 80 bits. The CG scheme utilizes cryptographic hash functions and the length of their outputs, denoted by $_c$, should be set to 256 bits to achieve the security level of 128 bits, as recommended in [32, 87]. This choice, in combination with the above constraints, leads to the lower bound of 418 bits for the length $_Q$, which can be increased at the cost of larger signatures and higher amount of work for the algorithms of the CG scheme. Finally, the above constraints imply that the length $_E$, which defines the length of each later chosen prime number E_i and is also inherent to the Camenisch-Lysyanskaya signature scheme, may vary between 580 bits and 1024 bits resp. 1624 bits (depending on the modulus size $_N$). In our measurements we will use an average value of $_E = 756$ bits.

Our choice of security parameters for the CG scheme to achieve mid-term protection can thus be summarized as follows: based on the recommendations of BNetzA/BSI [32] and ECRYPT [87] we will consider two cases, namely

$$_N = {}_P = |N| = |P| = 2048 \text{ bits} \quad \text{and} \quad {}_N = {}_P = |N| = |P| = 3248 \text{ bits}$$

and recommend further parameters to be set as follows:

$$_E = 756 \text{ bits}, \quad {}_Q = 418 \text{ bits}, \quad {}_c = 256 \text{ bits}, \quad {}_e = {}_s = 80 \text{ bits}.$$

We are now able to specify the core algorithms and protocols.

Key generation. The key generation algorithm GKg on input 1^κ performs the following steps:

1. Compute safe RSA modulus N and its factorization p and q using the $\mathsf{RSAGen}(1^{\ell_N})$ algorithm from Section 3.2.1.
2. Choose random elements $a, h, f, w \in_R QR(N)$.
3. Compute $g = h^\xi \bmod N$ using $\xi \in_R \mathbb{Z}^{\ell_N/2}$.
4. Choose random primes Q, P of length $_Q$, $_P$ with $Q|P-1$.
5. Pick $X_G, X_H \in_R \mathbb{Z}_Q$ with $G = F^{X_G} \bmod P$ and $H = F^{X_H} \bmod P$ where $F \in_R \mathbb{Z}_P^*$ of order Q.
6. Output $(gpk, gmsk, \boldsymbol{reg})$ such that:
   - group public key $gpk = (N, a, g, h, F, G, H, P, Q, w, f)$
   - group manager's secret key $gmsk = (gpk, p, q, X_G)$
   - registration list $\boldsymbol{reg}$ is initially empty.

It is assumed that key generation is performed in a trusted way. Otherwise a zero knowledge proof of knowledge of the value ξ can assure the correctness of the group public key. The additional value of w allows the user to chose his secret x_i on his own and allows him to prove knowledge of a root w in the signing process.

Join protocol. The join protocol Join is executed between the group manager with input $gmsk = (gpk, p, q, X_G)$ and a prospective member i with input $gpk = (N, a, g, h, F, G, H, P, Q, w, f)$. It proceeds as follows:

1. Member i picks random $x_i, s_i \in_R \mathbb{Z}_Q$ and $r'_i \in_R \mathbb{Z}_N$, computes $Y_i = G^{x_i} \bmod P$ and $C_i = g^{x_i} h^{r'_i} \bmod N$ and sends (Y_i, C_i, s_i) to the group manager together with (c, s_x, s_r) representing

   $$\mathsf{NIZKPoK}\quad x_i, r'_i \quad : \quad Y_i = G^{x_i} \bmod P \ \text{ and } \ C_i = g^{x_i} h^{r'_i} \bmod N$$

   computed as follows:

   - Choose $r_{x_i} \in_R \mathbb{Z}_Q$ and $r_{r'_i} \in_R \mathbb{Z}_N$ at random.
   - Compute $R_1 = G^{r_{x_i}} \bmod P$ and $R_2 = g^{r_{x_i}} h^{r_{r'_i}} \bmod N$.
   - Compute $c = \mathsf{Hash}_1(gpk, Y_i, C_i, R_1, R_2)$.
   - Compute $s_{x_i} = r_{x_i} + cx_i \bmod P$ and $s_{r'_i} = r_{r'_i} + cr'_i \bmod N$.

2. The group manager verifies the NIZKPoK as follows:

   - Compute $R'_1 = G^{s_x} Y_i^{-c} \bmod P$ and $R'_2 = g^{s_x} h^{s_r} C_i^{-c} \bmod N$.
   - Check that $c \stackrel{?}{=} \mathsf{Hash}_1(gpk, Y_i, C_i, R'_1, R'_2)$, otherwise abort.

   Then the group manager proceeds as follows:

   - Pick $e_i \in_R \{0,1\}^{\ell_e}$ such that $E_i = 2^{\ell_E} + e_i$ is prime.
   - Compute $w_i = w^{E_i^{-1}} \bmod N$.
   - Select $r''_i \in_R \mathbb{Z}_{\ell_e}$ and compute $y_i = (af^{s_i} C_i h^{r''_i})^{E_i^{-1}} \bmod N$ using the factors p and q.
   - Define $\boldsymbol{reg}[i] = (w_i, Y_i, E_i, s_i)$ and send (w_i, y_i, E_i, r''_i) back to member i.

3. Member i verifies that $y^{E_i} \stackrel{?}{=} af^{s_i} C_i h^{r''_i} \bmod N$ and that $w_i^{E_i} \stackrel{?}{=} w \bmod N$ and aborts if the check fails. Finally, member i stores $\boldsymbol{gsk}[i] = (gpk, w_i, x_i, r_i, y_i, e_i, s_i)$ with $r_i = r'_i + r''_i$ and $e_i = E_i - 2^{\ell_E}$ as his secret signing key.

Note that the group member stores (y_i, e_i, r_i, s_i), which corresponds to the group manager's signature (issued using the Camenisch-Lysyanskaya signature scheme from [55]). This signature certifies the membership of i in the group. Value w_i will be used by member i during the signing procedure to prove possession of a root of w, which will be modified upon each revocation event. The extra value s_i will be used by the group manager for revocation purposes, namely to make all group signatures issued by a revoked member i publicly linkable.

Revocation procedure. The revocation algorithm Revoke takes as input the group manager's secret key $gmsk = (gpk, p, q, X_G)$, the identity $i \in [1, n]$ of a member to be revoked, the registration entry $\boldsymbol{reg}[i] = (w_i, Y_i, E_i, s_i)$ and the current update information $\boldsymbol{upd}$ and proceeds as follows:

1. Publish $(i, E_i, s_i, \texttt{del})$ in $\boldsymbol{upd}$.
2. Replace w in gpk with $w^{E_i^{-1}} \bmod N$, whereby the inverse E_i^{-1} is computed using the factors p and q from $gmsk$.

Observe that each revocation event results in the growth of the published update information $\boldsymbol{upd}$.

Update procedure. The randomized update algorithm UpdM takes as input the current secret signing key $\boldsymbol{gsk}[i] = (gpk, w_i, x_i, r_i, y_i, e_i, s_i)$ and the update information $\boldsymbol{upd}$, and results in a modification of $\boldsymbol{gsk}[i]$. For each new entry $(j, E_j, s_j, \texttt{del})$ in $\boldsymbol{upd}$ it proceeds as follows:

1. Find α, β such that $\alpha E_i + \beta E_j = 1$.
2. Replace w_i in $\boldsymbol{gsk}[i]$ with $w_i^{\beta} w^{\alpha} \bmod N$.

Notice that the update procedure provides each unrevoked group member i with a new value w_i, which corresponds to the E_i-th root of the current value w published in gpk. To see this denote the updated witness by w_i and the previous witness by w_i'. Let w be the current accumulator value that was updated in gpk together with publication of E_j and let w' denote the previous accumulator value. In particular, the following relations hold:

$$w' = w_i'^{E_i} \bmod N \qquad \text{and} \qquad w = w'^{E_j^{-1}} \bmod N.$$

Correctness of the update procedure for w_i then follows immediately from the following equality:

$$\begin{aligned} w_i^{E_i} &= (w_i'^{\beta} w^{\alpha})^{E_i} = w_i'^{\beta E_i} w^{\alpha E_i} = w'^{\beta} w'^{E_j^{-1} \alpha E_i} \\ &= w'^{\beta + E_j^{-1} \alpha E_i} = w'^{(\beta E_j + \alpha E_i) E_j^{-1}} = w'^{E_j^{-1}} = w \bmod N. \end{aligned}$$

We observe that our specification considers the strongest revocation variant from [49], where revocation of member i allows to link all group signatures ever produced by i. This is due to publication of both E_i and s_i in $\boldsymbol{upd}$. If the group manager does not publish s_i then a revoked member i would be able to present group signatures that would be valid under older group public keys.

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, w_i, x_i, r_i, y_i, e_i, s_i)$ of member i, where $gpk = (N, a, g, h, F, G, H, P, Q, w, f)$, and a message $m \in \{0, 1\}^*$. The algorithms proceeds as follows:

1. Pick random $r \in_R \{0, 1\}^{\ell_N/2}$ and $R \in_R \mathbb{Z}_Q$ and compute

$$T_1 = h^r y_i w_i \bmod N, \quad T_2 = F^R \bmod P, \quad T_3 = G^{R+x_i} = G^R Y_i \bmod P$$

$$T_4 = H^{R+e_i} \bmod P, \quad T_5 = T_2^{s_i} \bmod P.$$

2. Compute a signature of knowledge $S = (c, s_\psi, s_\xi, s_\rho, s_\varepsilon, s_\tau)$

$$\mathsf{SoK}\left[\begin{array}{ccc} & & aw = T_1^{2^{\ell_E}+\varepsilon} f^{-\psi} g^{-\xi} h^\rho \bmod N \text{ and } T_2 = F^\tau \bmod P \\ & & T_3 = G^{\tau+\xi} \bmod P \text{ and } T_4 = H^{\tau+\varepsilon} \bmod P \\ \psi, \xi, \rho, \varepsilon, \tau & : & T_5 = T_2^\psi \bmod P \\ & & \varepsilon \in \{-2^{\ell_e+\ell_c+\ell_s}, +2^{\ell_e+\ell_c+\ell_s}\} \\ & & \psi, \xi \in \{-2^{\ell_Q+\ell_c+\ell_s}, +2^{\ell_Q+\ell_c+\ell_s}\} \end{array}\right] m$$

as follows:

- Choose $r_\psi, r_\xi \in_R \{0,1\}^{\ell_Q+\ell_c+\ell_s}, r_\rho \in_R \{0,1\}^{\ell_N/2+\ell_c+\ell_s}, r_\varepsilon \in_R \{0,1\}^{\ell_e+\ell_c+\ell_s}$ and $r_\tau \in_R \mathbb{Z}_Q$.
- Compute

$$R_1 = T_1^{r_\varepsilon} f^{-r_\psi} g^{-r_\xi} h^{r_\rho} \bmod N, \quad R_2 = F^{r_\tau} \bmod P, \quad R_3 = G^{r_\tau + r_\xi} \bmod P$$

$$R_4 = H^{r_\tau + r_\varepsilon} \bmod P, \quad R_5 = T_2^{r_\psi} \bmod P$$

- Compute $c = \mathsf{Hash}_2(gpk, T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, R_4, R_5, m)$.
- Compute

$$s_\psi = r_\psi + cs_i, \quad s_\xi = r_\xi + cx_i, \quad s_\rho = r_\rho + c(-r_i - rE_i)$$

$$s_\varepsilon = r_\varepsilon + ce_i, \quad s_\tau = r_\tau + cR \bmod Q.$$

3. Output group signature $\sigma = (S, T_1, T_2, T_3, T_4, T_5)$.

The pair (T_2, T_3) is an ElGamal encryption of Y_i and the SoK signature additionally proves that the signer knows the corresponding x_i. With values T_1, T_4, and T_5 the SoK signature further proves that the signer is in possession of a membership certificate (y_i, e_i, r_i, s_i) issued by the group manager. As part of the SoK signature, the signer moreover proves knowledge of a pair (w_i, e_i) satisfying the relationship $w = w_i^{2^{\ell_E}+e_i} \bmod N$.

Signature verification. The signature verification algorithm GVrfy takes as input the (current) group public key $gpk = (N, a, g, h, F, G, H, P, Q, w, f)$, a message m and a candidate group signature σ. It is further assumed that GVrfy has access to ***upd*** for the current list of s_i values. The algorithm proceeds as follows:

1. Parse σ as $(S, T_1, T_2, T_3, T_4, T_5)$.
2. Parse S as $(c, s_\psi, s_\xi, s_\rho, s_\varepsilon, s_\tau)$ and check its validity as follows:
   - Check that $s_\varepsilon \in \{0,1\}^{\ell_e+\ell_c+\ell_s}$ and $s_\psi, s_\xi \in \{0,1\}^{\ell_Q+\ell_c+\ell_s}$; otherwise output 0.
   - Compute

$$R_1' = (aw)^{-c} T_1^{c2^{\ell_E}+s_\varepsilon} f^{-s_\psi} g^{-s_\xi} h^{s_\rho} \bmod N, \quad R_2' = T_2^{-c} F^{s_\tau} \bmod P,$$

$$R_3' = T_3^{-c} G^{s_\tau + s_\xi} \bmod P, \quad R_4' = T_4^{-c} H^{s_\tau + s_\varepsilon} \bmod P, \quad R_5' = T_5^{-c} T_1^{s_\psi} \bmod P.$$

- Check that

$$c \overset{?}{=} \mathsf{Hash}_2(gpk, T_1, T_2, T_3, T_4, T_5, R'_1, R'_2, R'_3, R'_4, R'_5, m).$$

  If the check fails then output 0.

3. For all s_i published in ***upd*** check whether the following equation is fulfilled:

$$T_5^{\frac{P-1}{Q}} \bmod P = T_2^{\frac{P-1}{Q}s_i} \bmod P.$$

4. If no such s_i exists and S is a valid SoK signature on message m, then output 1; otherwise output 0.

We observe that access to ***upd*** is only needed to obtain published s_i values. These values enable linkability of all signatures produced by the corresponding signer i. This prevents the revoked signer i from claiming validity of his group signatures under older group public keys. Note that values s_i can also be published separately from ***upd***, e.g. in an additional revocation list. More importantly, linkability checks result in the linear running time of the algorithm GVrfy, depending on the size of ***upd***.

Opening procedure. The opening algorithm Open takes as input the group manager's secret key $gmsk = (gpk, p, q, X_G)$, the registration information ***reg***, a message m, and a group signature σ. The algorithm proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.
2. Parse σ as $(S, T_1, T_2, T_3, T_4, T_5)$.
3. Compute $Y = T_3/T_2^{X_G} \bmod P$.
4. If there exists an i such that ***reg***$[i]$ contains $Y_i \overset{?}{=} Y$, then output i; otherwise output 0.

Note that the opening procedure can be executed only by the group manager, who is the only party being in possession of the exponent X_G, which is needed to perform the decryption of Y_i from the ElGamal ciphertext pair (T_2, T_3). Recall that for each group member there is a unique value Y_i stored secretly by the group manager.

12.2. Performance Heuristics for Group Management and Opening

We start our performance analysis of the CG scheme with algorithms and protocols that have only marginal impact on the performance on the scheme in practice. This includes: key generation algorithm GKg that is executed only once during the life time of the group; algorithms for the management of the group (JoinM, Revoke, UpdM) of which JoinM and Revoke are typically executed on a powerful device representing the group manager and UpdM is performed by unrevoked members whenever someone is revoked, and algorithms such as JoinU, which is performed only once by each joining member, and Open, which is rarely executed by the group manager. We analyze performance of these algorithms using our heuristic approach. That is, in Table 12.1 we first count the amount of most dominant operations in $QR(N)$ and $\mathbb{Z}_P^*$ groups used by the CG scheme.

Notable is that all algorithms in Table 12.1 have constant amount of exponentiations, except for the Open algorithm, which is linear due to the implicit verification procedure. We would like to highlight that although algorithm UpdM requires computing $w_i^\beta w^\alpha \bmod N$, the length of exponents α and β is likely to be very short and not comparable to a full modular exponentiation in the $QR(N)$ group. As already mentioned, the opening procedure, in addition to one modular exponentiation in the $QR(N)$ group, is dominated by the time t_{GVrfy}, which is needed to verify a CG group signature; we refer for the latter costs to our analysis of CG verification performance in Section 12.3.

Table 12.1.: CG Scheme: Dominant Operations in Group Management and Opening

Operation	GKg	JoinM	JoinU	Revoke	UpdM	Open
modular exponentiation ($g^x \mod P$)	2	2	2	0	0	1
modular exponentiation ($g^x \mod N$)	1	5	8	0	0	0
further significant costs	–	–	–	–	–	t_{GVrfy}

As a second step, we use Table 12.1 to obtain heuristics for running times of all mentioned algorithms. For this purpose, we use our reference platforms and timing measurements for modular exponentiations in $QR(N)$ and $\mathbb{Z}_P^*$ groups from Sections 11.1 and 11.2, respectively. The resulting estimates are summarized in Table 12.2. For 2048-bit modulus we observe that timings of most algorithms remain below one second on both reference platforms. This is almost the case for 3248-bit modulus on the PC platform. As for the 3248-bit modulus on the smartphone platform, we observe that about two seconds for JoinU algorithm may still be acceptable in practice since this algorithm is typically executed only once.

Table 12.2.: CG Scheme: Performance Heuristics for Group Management and Opening

Platform	Modulus Size	Estimated Time					
		GKg	JoinM	JoinU	Revoke	UpdM	Open
PC	2048 bits	36.6 ms	132.6 ms	204.6 ms	fast	fast	6.3 ms
PC	3248 bits	167.2 ms	668.0 ms	1043.6 ms	fast	fast	21.0 ms
Smartphone	2048 bits	92.7 ms	335.5 ms	517.6 ms	fast	fast	16.0 ms
Smartphone	3248 bits	285.5 ms	1144.3 ms	1788.4 ms	fast	fast	35.4 ms
further significant costs		–	–	–	–	–	t_{GVrfy}

12.3. Performance Heuristics for Signature Generation and Verification

In order to provide intuition on the expected execution time of the signature generation and verification procedures of the CG scheme, we first break down its GSign and GVrfy algorithms into most dominant operations. That is, in Table 12.3 we summarize the total amount of modular exponentiations in $QR(N)$ and $\mathbb{Z}_P^*$ groups used by the CG scheme. The amount of modular exponentiations modulo N is identical in both algorithms, whereas the verification procedure has linear amount of exponentiations modulo P in the number of revoked members. For the prescribed choice of parameters, in particular due to the equal length of N and P, the difference between the two exponentiation operations is not significant. The linear amount of work in the verification procedure, however, will play an important role in our scalability analysis of the scheme. In general we observe that, if there are revoked members, the verification procedure of the CG scheme has higher costs than its signing procedure.

Table 12.3.: CG Scheme: Dominant Operations in Signature Generation and Verification

Operation	GSign	GVrfy
modular exponentiation ($g^x \mod P$)	8	$8 + r$
modular exponentiation ($g^x \mod N$)	5	5

Further, we use Table 12.3 to obtain heuristics for running times of both algorithms. For this purpose we use our reference platforms and timing measurements for modular exponentiations in $QR(N)$ and $\mathbb{Z}_P^*$ groups from Sections 11.1 and 11.2, respectively. The resulting estimates are summarized in Table 12.4. Notable is the significant increase in time, when moving from 2048-bit to 3248-bit RSA modulus, where the signing and verification procedures take four times longer on average.

Table 12.4.: CG Scheme: Performance Heuristics for Signature Generation and Verification

Platform	Modulus Length	Estimated Time	
	(128-bit security)	GSign	GVrfy
PC	2048 bits	170.4 ms	$170.4 + 6.3r$ ms
PC	3248 bits	794.0 ms	$794.0 + 21.0r$ ms
Smartphone	2048 bits	431.5 ms	$431.5 + 16.0r$ ms
Smartphone	3248 bits	1356.7 ms	$1356.7 + 35.4r$ ms

It would thus take on average about 170 ms for 2048-bit modulus and about 794 ms for 3248-bit modulus to generate a group signature on our PC platform. On our smartphone platform we should expect an increase to about 431 ms and 1357 ms, respectively. A more detailed analysis of the verification procedure and its scalability is provided in the next section.

12.3.1. Scalability of the Verification Procedure

Due to the revocation checks performed during the verification procedure, the execution time of GVrfy in the CG scheme depends on the length of the public update information $\boldsymbol{upd}$. Therefore, scalability of the verification procedure is of importance for the practical use of this scheme. If the group signature was produced by a revoked user i, then its verification procedure will terminate as soon as the revocation check is performed using the revocation token of i. That is, in the worst case the verification procedure would have to perform r revocation checks prior to its termination.

In Figure 12.1 we illustrate scalability of the verification procedure for the modulus size of 2048 bits. We observe that on our PC platform, the verification operation would require about 170 ms plus additional 6.3 ms for each revoked member, whereas on our smartphone platform the expected time will increase by several factors to about 431 ms plus additional 16 ms per revoked member. Note that for groups with up to 1000 revoked members the corresponding time for verification would require on average up to 10 seconds on our PC resp. about 17 seconds on our smartphone platform. In case of 100,000 revoked users, the verification time on PC would take about 10 minutes and on smartphone about 27 minutes, which seems impractical for most applications, not talking about 1.5 hours on PC and about 4.4 hours on smartphone once r reaches one million. Note that these timings are obtained based on our heuristics and thus provide a high-level intuition about the practicality of the CG scheme.

In Figure 12.2 we extend our scalability analysis to the verification procedure for the modulus size of 3248 bits. Here the expected timings are generally higher. On our PC platform verification would take about 794 ms plus additional 21 ms per revoked member, whereas on our smartphone platform it would take about 1357 ms plus additional 35.4 ms per revoked member to verify a single group signature. In groups with up to 1000 revoked members the corresponding time for verification would on average amount to up to 22 seconds on our PC resp. about 37 seconds on our smartphone platform, which is almost half as efficient than for 2048-bit modulus. In case of 100,000 revoked users, the verification time on our PC platform would take almost half an hour and almost an hour on our smartphone platform. Once r reaches one million, we should expect about 5 hours on PC and about 9.8 hours on smartphone.

Although the constant amount of time needed to verify one signature without performing any revocation checks could suit some practical applications, especially using 2048-bit modulus, the poor scalability behavior of the CG scheme represents a major bottleneck and constrains its use in practice.

12.4. Space Requirements for the Main Parameters

In Table 12.5 we estimate the required amount of space to store the most significant parameters used in the CG scheme. More precisely, we consider lengths of group public keys gpk, individual secret signing keys $\boldsymbol{gsk}[i]$, group signatures σ, group manager's secret keys $gmsk$ and the registration lists $\boldsymbol{reg}$ that the group manager has to keep private, and the public update information $\boldsymbol{upd}$ that is published by the group manager and allows unrevoked group members to update their secret signing keys after occurring revocation events. We estimate storage requirements

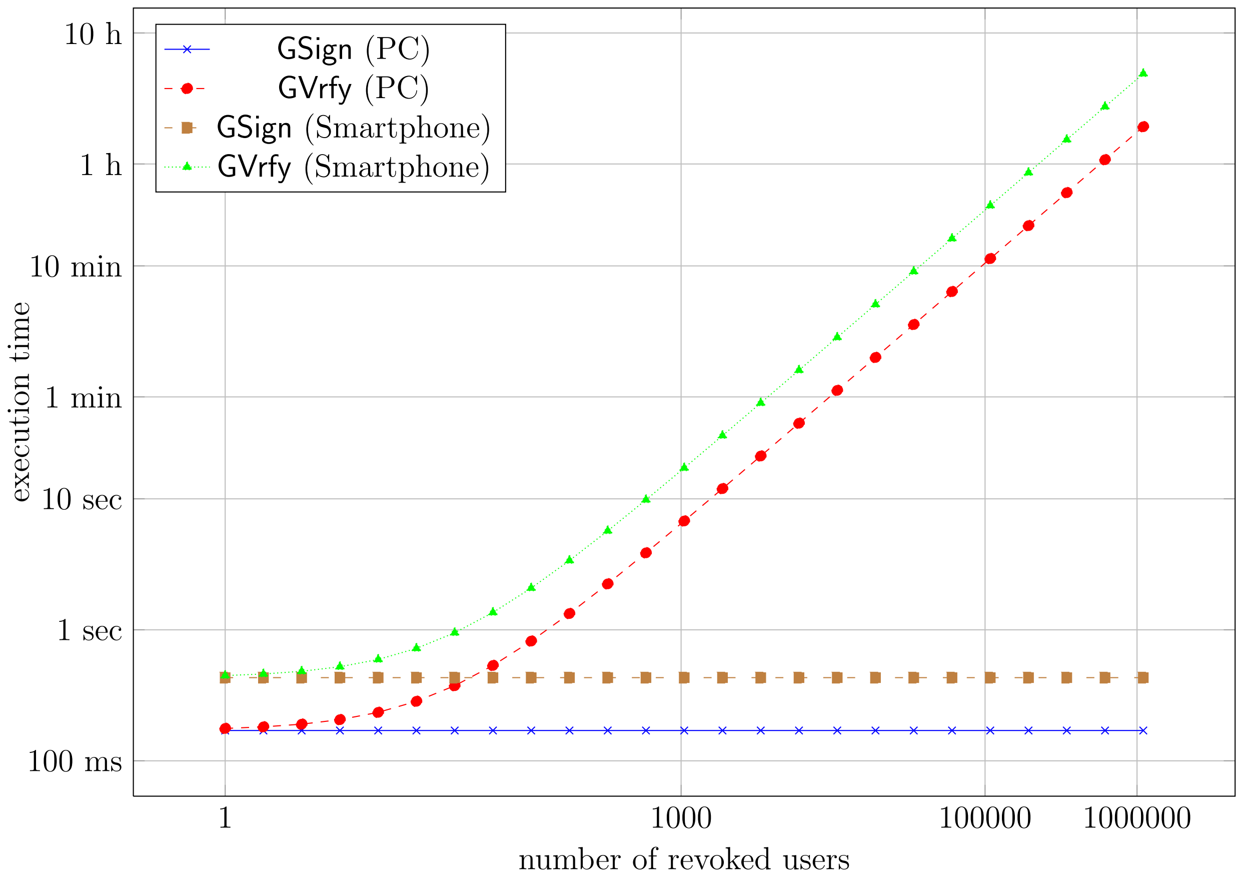


Figure 12.1.: CG Scheme: Scalability of Signing and Verification Procedures (2048-bit modulus)

for these parameters using our previous analysis of space complexity from Table 10.4 and applying concrete parameter lengths for the addressed security levels. As measurement unit in the table we use 1 Kb = 1024 bits.

We first observe that several parameters of the CG scheme remain of constant length. This refers to group public keys, secret signing keys, and generated group signatures. The secret information in each $\boldsymbol{gsk}[i]$ that could potentially be stored in a protected memory is lower than the size of group public keys and signatures that are used in the verification procedure.

Table 12.5.: CG Scheme: Space Requirements

Modulus Length	gpk	$\boldsymbol{gsk}[i]$	σ	$gmsk$ and $\boldsymbol{reg}$	$\boldsymbol{upd}$
2048 bits	20.41 Kb	6.89 Kb	14.79 Kb	$2.41 + 5.15(m - r)$ Kb	$1.15r$ Kb
3248 bits	32.13 Kb	10.41 Kb	21.82 Kb	$3.58 + 7.49(m - r)$ Kb	$1.15r$ Kb

m — number of group members; r — number of revoked members.

The length of the public update information $\boldsymbol{upd}$ increases over the time, as members get revoked. This increase can be seen as another impact factor on the scalability of the CG

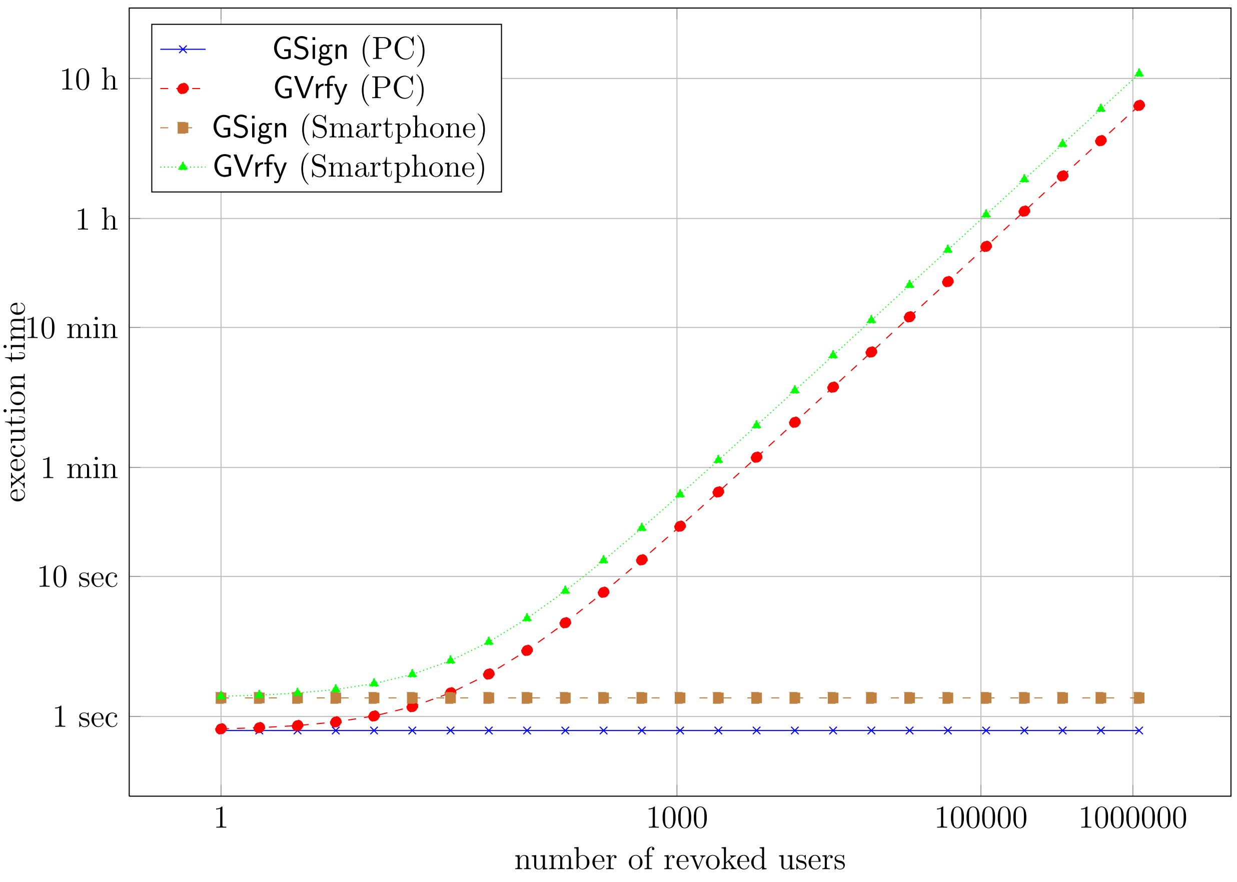


Figure 12.2.: CG Scheme: Scalability of Signing and Verification Procedures (3248-bit modulus)

scheme. As shown in Figure 12.3, with every newly revoked member the size of ***upd*** increases by 1.15 Kb. That is, if the number of revoked members reaches 1000 then about 143 KB space will be needed to store the published update information. If this number exceeds one million then the required amount of space increases to over 139 MB. We observe, however, that unrevoked members need not to download the entire list ***upd*** to perform the update of their signing keys, i.e., they only need those elements that have been added to the list since their last update procedure. Therefore, the increasing length of ***upd*** in the CG scheme wouldn't count as a severe limitation in practice.

Furthermore, we notice that the required amount of space for storing the secret information on the side of the group manager also increases over the life time of the group. This secret information is in fact comprised of the group manager's secret key *gmsk* and the registration information ***reg*** that is used by the group manager in the opening procedure and whose size depends linearly on the number of unrevoked members in the system. Each time a new member joins the group, the amount of space for this information increases by 5271 bits for 2048-bit modulus and by 7671 bits for 3248-bit modulus. Note that, each time a member is revoked, the size of the secret information in ***reg*** can be decreased by the same amount, i.e., once the group manager publishes the revocation token of that member. In the worst case, i.e., if no users are revoked, ***reg*** would require about 643 KB (for 2048-bit modulus) resp. 936 KB (for 3248-bit

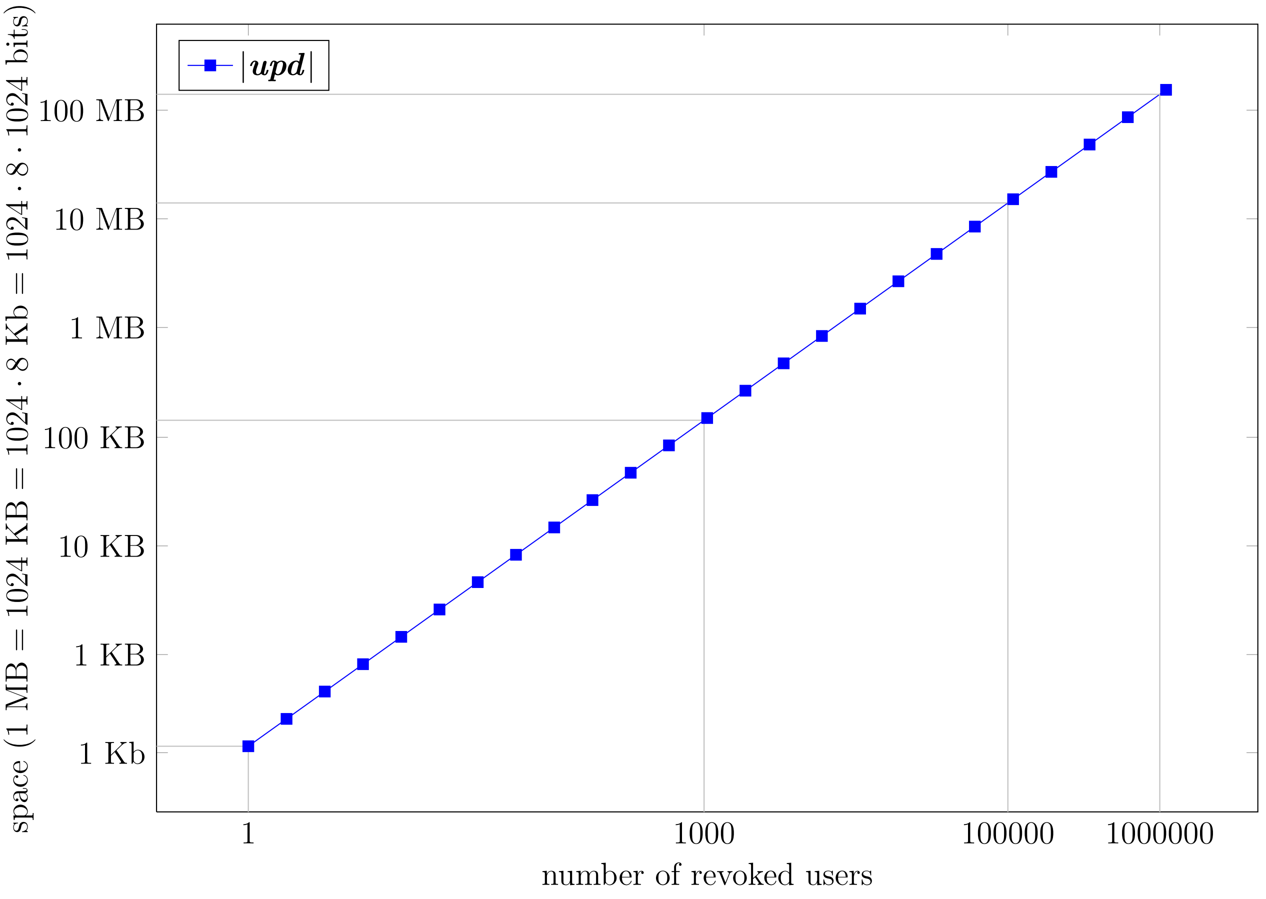


Figure 12.3.: CG Scheme: Scalability of Published Update Information

modulus) for groups of 1000 members and over 628 MB resp. 914 MB for groups of one million users. In practice the actual content of *gmsk* can be stored in a tamper-resistant hardware module for better protection. The plain size of *gmsk* (i.e., not counting secret revocation information) is constant and relatively small, namely 2.41 Kb or 3.58 Kb, depending on the modulus size.

13. Specification and Performance of the Boneh-Shacham Scheme

13.1. Detailed Specification of the Boneh-Shacham Scheme

The Boneh-Shacham (BS) group signature scheme has a single security parameter $\kappa \in \mathbb{N}$ and uses bilinear groups $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, and $\mathbb{G}_T$ of prime order Q with $|Q| = \kappa$, a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, and an efficiently computable homomorphism ψ from $\mathbb{G}_2$ to $\mathbb{G}_1$ with $\psi(g_2) = g_1$. Additionally, two hash functions $\mathsf{Hash}_1 : \{0,1\}^* \to \mathbb{G}_2^2$ and $\mathsf{Hash}_2 : \{0,1\}^* \to \mathbb{Z}_Q$ are used, treated as random oracles. Note that the required homomorphism ψ implies, that the BS scheme is not implementable using Type-3 pairings. In the following we specify the core algorithms and protocols of the BS scheme. Our description follows the specification from [38].

Key generation. The key generation algorithm GKg on input 1^κ and the number of group members n performs the following steps:

1. Select $\gamma \in_R \mathbb{Z}_Q^*$ and set $w = g_2^\gamma$.
2. For each user $i \in [1, n]$, generate a tuple (A_i, x_i) with $x_i \in_R \mathbb{Z}_Q^*$ such that $\gamma + x_i = 0$ and $A_i = g_1^{1/(\gamma + x_i)}$.
3. Output $(gpk, RL, \boldsymbol{grt}, \boldsymbol{gsk})$ such that:
   - group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, w)$
   - revocation list RL is initially empty
   - n-element vector of revocation tokens of member i: $\boldsymbol{grt}[i] = (A_i)$
   - n-element vector of secret signing keys of member i: $\boldsymbol{gsk}[i] = (gpk, A_i, x_i)$.

It is assumed that key generation is performed in a trusted way. In particular, this means that the elements γ and x_i, with $i \in [1, n]$, are chosen independently at random from $\mathbb{Z}_Q^*$ and, more importantly, γ is not known to any party except for the issuer.

Since γ is not used after the key generation is performed it should be safely erased by the issuer (cf. Section 2.4).

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, A_i, x_i)$ of member i and a message $m \in \{0,1\}^*$, and proceeds as follows:

1. Pick a random nonce $r \in_R \mathbb{Z}_Q^*$. Obtain generators $\hat{u}, \hat{v}$ in $\mathbb{G}_2$ from Hash_1 as
$$(\hat{u}, \hat{v}) = \mathsf{Hash}_1(gpk, m, r) \in \mathbb{G}_2^2$$
and compute their images in $\mathbb{G}_1$:
$$u = \psi(\hat{u}), \qquad v = \psi(\hat{v}).$$
2. Select $\alpha \in_R \mathbb{Z}_Q^*$ and compute:
$$T_1 = u^\alpha, \qquad T_2 = A_i v^\alpha.$$
3. Set $\delta = x_i \alpha \in \mathbb{Z}_Q^*$.
4. Compute a signature of knowledge $S = (c, s_\alpha, s_x, s_\delta)$
$$\mathsf{SoK} \quad \alpha, \delta, x_i \quad : \quad \begin{array}{c} T_1 = u^\alpha \text{ and } T_1^{x_i} = u^\delta \\ e(T_2 v^{-\alpha}, w g_2^{x_i}) = e(g_1, g_2) \end{array} \quad m$$
as follows:
   - Choose r_α, r_x and $r_\delta \in_R \mathbb{Z}_Q$.
   - Compute
   $$R_1 = u^{r_\alpha} \qquad R_2 = e(T_2, g_2)^{r_x} \cdot e(v, w)^{-r_\alpha} \cdot e(v, g_2)^{-r_\delta} \qquad R_3 = T_1^{r_x} \cdot u^{-r_\delta}.$$
   - Compute $c = \mathsf{Hash}_2(gpk, m, r, T_1, T_2, R_1, R_2, R_3)$.
   - Compute
   $$s_\alpha = r_\alpha + c\alpha \qquad s_x = r_x + cx_i \qquad s_\delta = r_\delta + c\delta.$$
5. Output group signature $\sigma = (S, r, T_1, T_2)$.

The SoK signature proves that the signer is in possession of a pair (A_i, x_i) such that $A_i = g_1^{1/(\gamma + x_i)}$; thus, proving that the signer has a valid signing key $\boldsymbol{gsk}[i]$.

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, w)$, the revocation list RL, a message m, and a candidate group signature σ and proceeds as follows:

1. Parse σ as (S, r, T_1, T_2).
2. Parse S as $(c, s_\alpha, s_x, s_\delta)$ and check its validity as follows:
   - Obtain generators $\hat{u}, \hat{v}$ in $\mathbb{G}_2$ as $(\hat{u}, \hat{v}) = \mathsf{Hash}_1(gpk, m, r)$ and compute their images in $\mathbb{G}_1$:
   $$u = \psi(\hat{u}), \qquad v = \psi(\hat{v}).$$

- Compute:

$$\begin{aligned} R'_1 &= u^{s_\alpha}/T_1^c \\ R'_2 &= e(T_2, g_2)^{s_x} e(v, w)^{-s_\alpha} e(v, g_2)^{-s_\delta} \cdot (e(T_2, w)/e(g_1, g_2))^c \\ R'_3 &= T_1^{s_x} u^{-s_\delta} \end{aligned}$$

- Check that:

$$c \stackrel{?}{=} \mathsf{Hash}_2(gpk, m, r, T_1, T_2, R'_1, R'_2, R'_3).$$

  If the check fails, output 0.

3. For all $A_i = \boldsymbol{grt}[i] \in RL$ with $i \in [1, n]$, check whether A_i is encoded in (T_1, T_2) by checking if

$$e(T_2/A_i, \hat{u}) \stackrel{?}{=} e(T_1, \hat{v}).$$

   If no element of RL is encoded in (T_1, T_2) then output 1; otherwise output 0.

Implicit opening procedure. The implicit opening algorithm Open takes as input the group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, w)$, the vector of revocation tokens $\boldsymbol{grt}$, a message m, and a group signature σ and proceeds as follows:

1. Check for all $i \in [1, n]$ whether $\mathsf{GVrfy}(gpk, \boldsymbol{grt}[i], m, \sigma) = 0$.

2. Output the *first* such i; or 0 if no such i is found.

13.2. Performance Heuristics for Group Management and Opening

We start our performance analysis of the BS scheme with algorithms that have only marginal impact on the performance of the scheme in practice. This includes: key generation algorithm GKg that is executed only once during the life time of the group and the implicit opening procedure Open. Note that both of these operations are typically executed on powerful devices representing the group manager. We analyze performance of both algorithms using our heuristic approach.

The BS scheme is static. Hence, the group manager or issuer (since BS may have distributed authorities) computes an individual secret signing key $\boldsymbol{gsk}[i]$ for each member of the group. Let m denote the total number of group members. In this case GKg will require m exponentiations in the input group $\mathbb{G}_1$ and only one exponentiation in the input group $\mathbb{G}_2$, which is the most costly operation in the assumed setting, according to Section 11.3.

Using the timing measurements with the PBC library from Section 11.3, our heuristic approach indicates roughly 10.8 seconds for the generation of 1000 keys, roughly 17.8 minutes for 100,000 keys, and about 2.9 hours for one million keys on the PC platform, and at least twice more on the Smartphone platform.

The opening procedure Open of the BS scheme is implicitly given via the GVrfy algorithm, except that instead of the revocation list RL, which GVrfy uses normally as input, in Open it uses the list of revocation tokens $\boldsymbol{grt}$. In the worst case this execution of GVrfy would have to process one entry $\boldsymbol{grt}[i]$ for each member of the group. This corresponds to the execution time of GVrfy with at most m revocation checks. We postpone the corresponding analysis of GVrfy to the next section.

13.3. Performance Heuristics for Signature Generation and Verification

We provide intuition about the running time of the signature generation and verification procedures in the BS scheme as follows. In Table 13.1 we first break down algorithms GSign and GVrfy into their dominant operations within the assumed setting of bilinear maps. That is, we count the total amount of exponentiations in groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ and the amount of pairing evaluations $e(\cdot, \cdot)$ performed by these algorithms. We notice that none of these algorithms requires any exponentiation in $\mathbb{G}_2$, which has the highest costs amongst the three groups according to Section 11.3. Nonetheless, the verification procedure of the BS scheme requires linear amount of work for performing the revocation checks. These costs are dominated by pairing evaluations that have comparable costs to exponentiations in $\mathbb{G}_2$. Leaving out those costs we observe that the verification procedure would still be costlier than the signing procedure.

Table 13.1.: BS Scheme: Dominant Operations in Signature Generation and Verification

Operation	GSign	GVrfy
exponentiations in $\mathbb{G}_1$	5	4
exponentiations in $\mathbb{G}_2$	0	0
exponentiations in $\mathbb{G}_T$	3	4
pairing evaluations $e(\cdot, \cdot)$	3	$6 + r$

r — number of revoked members.

Further, we use Table 13.1 to obtain heuristics for running times of both algorithms for the chosen security level of approximately 128 bits. For this purpose we use our reference platforms and timing measurements of dominant operations in the bilinear map setting from Section 11.3. The resulting estimates are summarized in Table 13.2.

We observe that it would thus take less than half a second to generate a BS group signature on our PC platform and more than one second on our Smartphone. In the next section we discuss scalability of its verification procedure, based on our measurements and those obtained from the literature.

Table 13.2.: BS Scheme: Performance Heuristics for Signature Generation and Verification

Platform	Order of $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$	Estimated Time	
	(≈128-bit security)	GSign	GVrfy
PC	332 bits	402.4 ms	$691.6 + 91.8r$ ms
Smartphone	332 bits	1211.7 ms	$2118.0 + 283.2r$ ms

r — number of revoked members.

13.3.1. Scalability of the Verification Procedure

The verification time of BS group signatures depends on the length of the revocation list RL. Hence, we are interested in the scalability of the verification procedure considering the potential increase of $|RL| = r$ over the life time of the group. As demonstrated in Figure 13.1, the increase of r has linear impact on the verification time, i.e. verification procedure would require about 700 ms plus roughly 100 ms per each revoked member on our PC platform and about 2000 ms plus 250 ms per revoked member on our Smartphone platform.

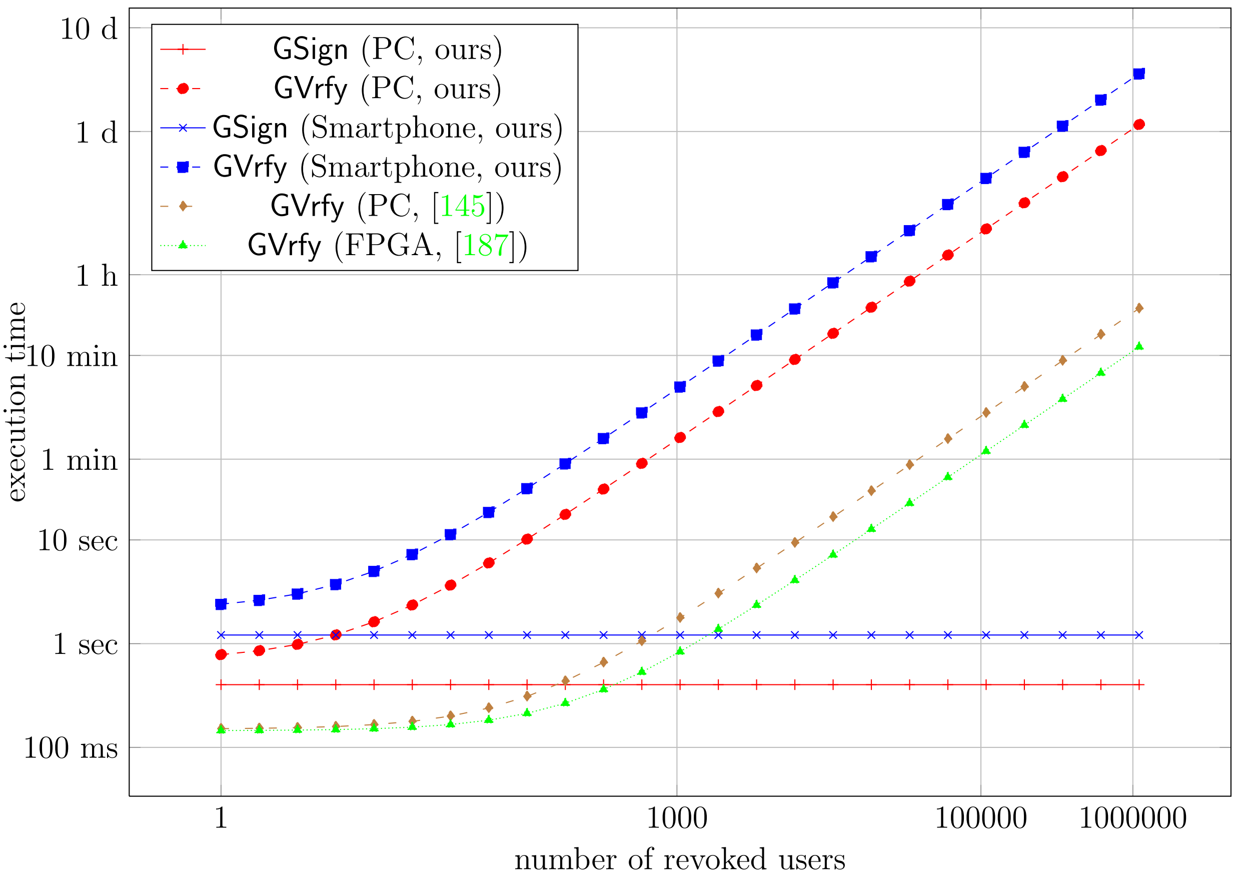


Figure 13.1.: BS Scheme: Scalability of Signing and Verification Procedures

Notice that if the group signature has been issued by a revoked member then up to r revocation checks will be needed in the worst case to mark the signature as invalid. For groups with up to 1000 revoked members the entire verification procedure may take several minutes, which can already be critical for the practical use of the scheme. In larger groups, with up to 100,000 resp. one million revoked members it would take over two hours resp. more than one day, based on our measurements, which is far from being practical. We observe that these timings are estimated on our reference platforms using the PBC library, which is, however, not as efficient as some measurements from the literature report. Therefore, we are interested in estimations that can be obtained based on other sources than the PBC library.

For this purpose, we utilize reports obtained from our literature survey, summarized earlier in Table 11.4. We will consider the fastest measurement of 1.55 ms on a PC platform from [145] and the even faster measurement of 0.66 ms, which was reported in [187] on an FPGA platform. We stress that these measurements drastically reduce the required amount of time for pairing-based operations in comparison to the open-source PBC library.

In Figure 13.1 we illustrate the impact of these state-of-the-art pairing implementations on the verification procedure of the BS scheme. In particular, for the PC platform the estimated time is significantly reduced to about a second for up to 1000 revoked members, about a couple of minutes for 100,000 revoked users and about 25 minutes for one million revoked users. With the fastest implementation on an FPGA platform those timings can be reduced even further, i.e. to less than a second for 1000 revocation checks, about one minute for 100,000 revocation checks, and about 11 minutes for one million of revoked members.

Although the amount of time for performing the revocation checks can be significantly sped up using state-of-the-art pairing implementations, those costs still represent the main performance bottleneck of the BS scheme.

13.4. Space Requirements for the Main Parameters

In Table 13.3 we estimate the required amount of space to store the main private and public parameters of the BS scheme. More precisely, we consider lengths of the group public key gpk, individual secret signing keys $\boldsymbol{gsk}[i]$, group signatures σ, revocation tokens in $\boldsymbol{grt}$ that are considered as a secret information of the group manager, and the revocation list RL that is publicly known and used during the verification procedure. We estimate storage requirements for these parameters using our previous analysis of space complexity from Table 10.4 and applying the concrete lengths of parameters for the addressed security levels. As a measurement unit in the table we use 1 Kb = 1024 bits.

We observe first that several parameters of the BS scheme remain of constant length. This refers to group public keys, secret signing keys, and generated group signatures. The secret information in each $\boldsymbol{gsk}[i]$ that could potentially be stored in a protected memory is lower than the size of group the public keys and signatures, used in the verification procedure.

The BS scheme is static. Therefore, the group manager has to initially keep secret the revocation tokens in $\boldsymbol{grt}$ for all members of the group. That is the amount of secret information stored by the group manager initially corresponds to about 360 bits for each member of the group. The amount of secret information in $\boldsymbol{grt}$ accounts to about 44 KB for groups with up to 1000 members, to about 4.3 MB for groups of 100,000 members, and to about 42.8 MB for

Table 13.3.: BS Scheme: Space Requirements

Order of $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$	gpk	$\boldsymbol{gsk}[i]$	σ	$\boldsymbol{grt}$	RL
332 bits	2.75 Kb	0.68 Kb	2.32 Kb	$0.35(m-r)$ Kb	$0.35r$ Kb

m — number of group members; r — number of revoked members.

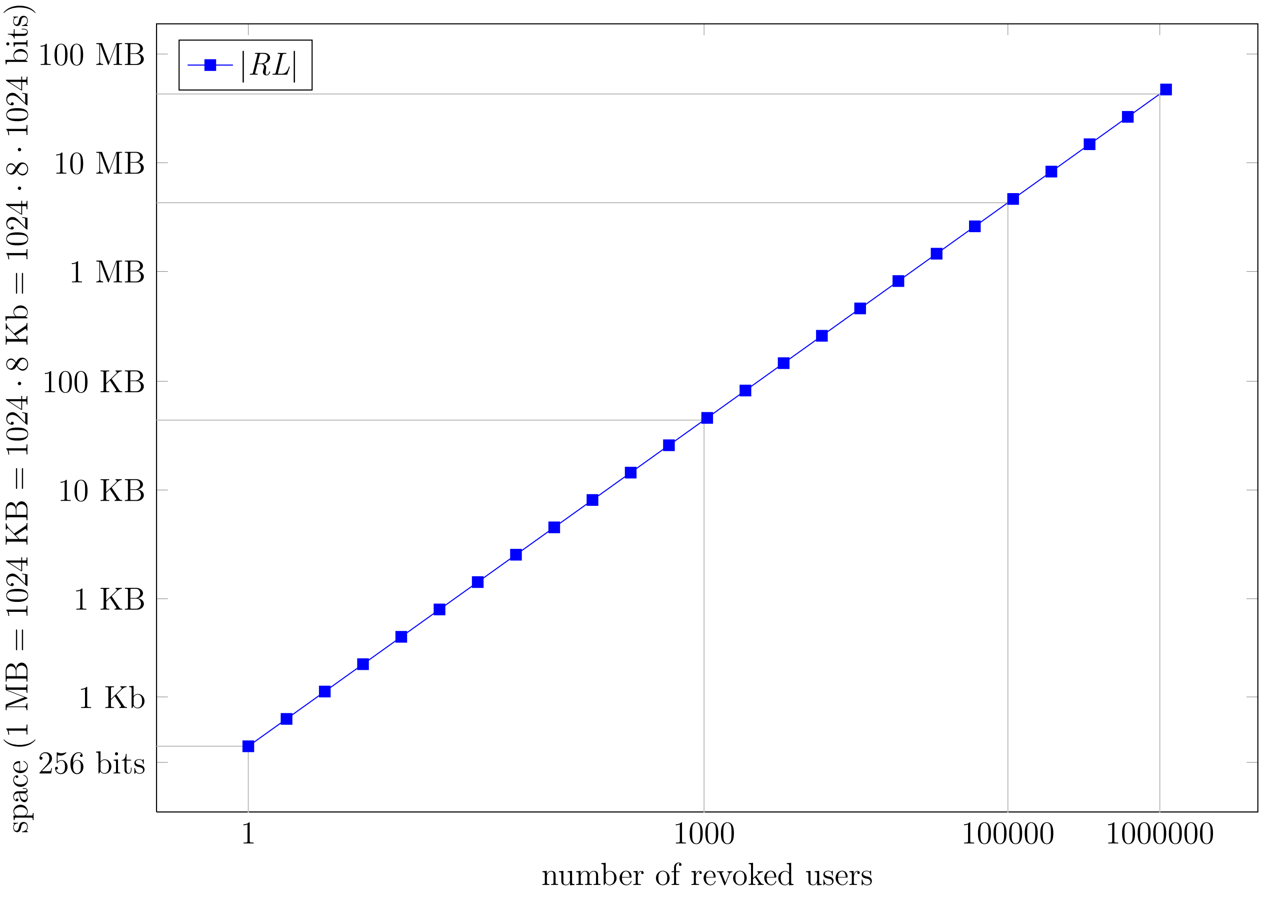


Figure 13.2.: BS Scheme: Scalability of Published Revocation Lists

groups with one million members.

However, as soon as member i is revoked and its token $\boldsymbol{grt}[i]$ is published in the revocation list RL the group manager can erase this token from $\boldsymbol{grt}$. Hence, the amount of secret information that has to be stored on the side of the group manager will decrease over the life time of the group by 360 bits for each revoked user. Since the entire revocation list must be used as input to the verification procedure it would have to be stored or fetched by verifiers to recognize signatures of revoked members. Therefore, the increasing size of the revocation list represents another impact factor on the scalability of the scheme, as illustrated in Figure 13.2.

14. Specification and Performance of the Bichsel-Camenisch-Neven-Smart-Warinschi Scheme

14.1. Detailed Specification of the Bichsel-Camenisch-Neven-Smart-Warinschi Scheme

The BCNSW-VLR group signature scheme is a dynamic scheme with verifiable opening employing a user PKI (connected with an unforgeable digital signature scheme $\Sigma = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vrfy})$ specified in Section 3.4). It has a single security parameter $\kappa \in \mathbb{N}$ and uses an asymmetric bilinear pairing, namely groups $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, and $\mathbb{G}_T$ of prime order Q with $|Q| = \kappa$ and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Additionally, two hash functions $\mathsf{Hash}_1, \mathsf{Hash}_2 : \{0,1\}^* \to \mathbb{Z}_Q$ are used and modeled as random oracles in the proof of security. The BCNSW-VLR scheme is based on the ordinary Camenisch-Lysyanskaya signature scheme [57, Scheme A] and especially makes use of the fact that these signatures are re-randomizable, i.e. given a valid signature $(a, b, c) \in \mathbb{G}_1^3$ on a message $m \in \mathbb{Z}_Q$, the tuple (a^r, b^r, c^r) for any $r \in_R \mathbb{Z}_Q^*$ will also be a valid signature on m.

In the following we specify the core algorithms and protocols of the BCNSW-VLR scheme. Our description follows the specification from [31].

Key generation. The key generation algorithm GKg on input 1^κ performs the following steps:

1. Select $x \in_R \mathbb{Z}_Q$, $y \in_R \mathbb{Z}_Q$ and set $X = g_2^x$, $Y = g_2^y$.
2. Output $(gpk, gmsk, \boldsymbol{reg})$ such that:
   - group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, X, Y)$
   - group manager's secret key $gmsk = (gpk, x, y)$
   - registration list $\boldsymbol{reg}$ is initially empty
   - revocation list RL is initially empty
   - list of revocation tokens $\boldsymbol{grt}$ is initially empty.

It is assumed that key generation is performed in a trusted way. In particular, this means that the elements x, y are chosen independently at random from $\mathbb{Z}_Q$. This assumption is necessary to ensure trust into the group public key gpk.

User key generation. The user key generation algorithm UKg on input 1^κ computes and returns the private/public key pair $(\boldsymbol{usk}[i], \boldsymbol{upk}[i])$ using the key generation algorithm $(\boldsymbol{usk}[i], \boldsymbol{upk}[i]) \leftarrow_R \mathsf{Kg}(1^\kappa)$ of the connected (unforgeable) digital signature scheme Σ, where $\boldsymbol{upk}[i]$ is assumed to be certified. As noticed in Section 2.3.2, the user PKI is modeled here using the list of registered public keys $\boldsymbol{upk}$.

Join protocol. The join protocol Join is executed between the group manager with input $gmsk = (gpk, x, y)$ and a prospective member i with input $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, X, Y)$ and an own PKI-certified key pair $(\boldsymbol{usk}[i], \boldsymbol{upk}[i])$. It proceeds as follows:

1. The group manager chooses a random $K_i \in_R \mathbb{Z}_Q$, computes $t_i = \mathsf{Hash}_2(K_i)$, and sends t_i to the member i.

2. Member i chooses $\tau_i \in_R \mathbb{Z}_Q$, computes $s_i = g_1^{\tau_i}$, $r_i = X^{\tau_i}$, $k_i = e(g_1, r_i)$, as well as $\bar{\sigma}_i \leftarrow_R \mathsf{Sign}(\boldsymbol{usk}[i], k_i)$, sends $(s_i, r_i, \bar{\sigma}_i)$ to the group manager together with a proof (c, s_{τ_i}) representing
$$\mathsf{NIZKPoK} \quad \tau_i \quad : \quad s_i = g_1^{\tau_i} \text{ and } r_i = X^{\tau_i}$$
computed as follows:
   - Choose $r_{\tau_i} \in_R \mathbb{Z}_Q$.
   - Compute $R_1 = g_1^{r_{\tau_i}}$ and $R_2 = X^{r_{\tau_i}}$.
   - Compute $c = \mathsf{Hash}_1(gpk, s_i, r_i, R_1, R_2)$.
   - Compute $s_{\tau_i} = r_{\tau_i} + c\tau_i \bmod Q$.

3. The group manager checks that $\mathsf{Vrfy}(\boldsymbol{upk}[i], e(g_1, r_i), \bar{\sigma}_i) \stackrel{?}{=} 1$. She then verifies the NIZKPoK as follows:
   - Compute $R_1' = g_1^{s_{\tau_i}}/s_i^c$ and $R_2' = X^{s_{\tau_i}}/r_i^c$.
   - Check that $c \stackrel{?}{=} \mathsf{Hash}_1(gpk, s_i, r_i, R_1', R_2')$; otherwise abort.

   The group manager now computes $z_i = s_i g_1^{K_i}$ and $w_i = r_i X^{K_i}$, stores $(w_i, r_i, K_i, \bar{\sigma}_i)$ in $\boldsymbol{reg}[i]$ and w_i in $\boldsymbol{grt}[i]$, chooses $\rho_i \in_R \mathbb{Z}_Q$, computes $a_i = g_1^{\rho_i}$, $b_i = a_i^y$, and $c_i = a_i^x z_i^{\rho_i xy}$, and sends (a_i, b_i, c_i, K_i) to the user together with a proof $(c, s_x, s_y, s_{\rho_i}, s_{\rho_i xy})$ representing
$$\mathsf{NIZKPoK} \quad x, y, \rho_i \quad : \quad \begin{array}{c} c_i = a_i^x z_i^{\rho_i xy} \text{ and } a_i = g_1^{\rho_i} \text{ and } X = g_2^x \\ Y = g_2^y \text{ and } 1 = b_i^x / g_1^{\rho_i xy} \end{array}$$
computed as follows:
   - Choose $r_x, r_y, r_{\rho_i}, r_{\rho_i xy} \in_R \mathbb{Z}_Q$.

- Compute

$$R_1 = a_i^{r_x} z_i^{r_{\rho_i xy}}, \qquad R_2 = g_1^{r_{\rho_i}}, \qquad R_3 = g_2^{r_x}, \qquad R_4 = g_2^{r_y}, \qquad R_5 = b_i^{r_x}/g_1^{r_{\rho_i xy}}$$

- Compute $c = \mathsf{Hash}_1(gpk, s_i, r_i, z_i, R_1, R_2, R_3, R_4, R_5)$.
- Compute

$$s_x = r_x + cx \bmod Q, \qquad s_y = r_y + cy \bmod Q, \qquad s_{\rho_i} = r_{\rho_i} + c\rho_i \bmod Q,$$

$$s_{\rho_i xy} = r_{\rho_i xy} + c\rho_i xy \bmod Q$$

4. Member i computes $\xi_i = \tau_i + K_i \bmod Q$ and checks whether $t_i \overset{?}{=} \mathsf{Hash}_2(K_i)$. She also verifies $e(a_i, Y) = e(b_i, g_2)$ and the proof $(c, s_x, s_y, s_{\rho_i}, s_{\rho_i xy})$ as follows:
   - Compute

   $$R'_1 = a_i^{s_x} g_1^{\xi_i s_{\rho_i xy}}/c_i^c, \qquad R'_2 = g_1^{s_{\rho_i}}/a_i^c, \qquad R'_3 = g_2^{s_x}/X^c,$$
   $$R'_4 = g_2^{s_y}/Y^c, \qquad R'_5 = b_i^{s_x}/g_1^{s_{\rho_i xy}}$$

   - Check that $c \overset{?}{=} \mathsf{Hash}_1(gpk, s_i, r_i, g_1^{\xi_i}, R'_1, R'_2, R'_3, R'_4, R'_5)$, otherwise abort.

   Finally she stores her secret key $\boldsymbol{gsk}[i] = (gpk, \xi_i, a_i, b_i, c_i)$.

The (a_i, b_i, c_i) part of the secret signing key represents an ordinary Camenisch-Lysyanskaya signature on message ξ_i.

Signature generation. The signing algorithm GSign takes as input the secret signing key $\boldsymbol{gsk}[i] = (gpk, \xi_i, a_i, b_i, c_i)$ of member i, with $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, X, Y)$, and a message $m \in \{0,1\}^*$, and proceeds as follows:

1. Re-randomize the Camenisch-Lysyanskaya signature of $\boldsymbol{gsk}[i]$ by choosing $r \in_R \mathbb{Z}_Q$ and computing $T_1 = a_i^r$, $T_2 = b_i^r$, and $T_3 = c_i^r$.

2. Compute a signature of knowledge $S = (c, s_{\xi_i})$

   $$\mathsf{SoK} \quad \xi_i \quad : \quad \tfrac{e(T_3, g_2)}{e(T_1, X)} = e(T_2, X)^{\xi_i} \qquad m$$

   as follows:

   - Choose $r_{\xi_i} \in_R \mathbb{Z}_Q$.
   - Compute $R_1 = e(T_2, X)^{r_{\xi_i}}$.
   - Compute $c = \mathsf{Hash}_1(gpk, \tfrac{e(T_3, g_2)}{e(T_1, X)}, R_1)$.
   - Compute $s_{\xi_i} = r_{\xi_i} + c\xi_i \bmod Q$.

3. Output group signature $\sigma = (S, T_1, T_2, T_3)$.

In the above signature generation algorithm, which leverages the re-randomization property of ordinary Camenisch-Lysyanskaya signatures, the SoK signature proves that the signer knows ξ_i for which (T_1, T_2, T_3) is a valid signature; thus, proving that the signer has a valid signing key $\boldsymbol{gsk}[i]$.

Signature verification. The signature verification algorithm GVrfy takes as input the group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, X, Y)$, the revocation list RL, a message m, and a candidate group signature σ and proceeds as follows:

1. Parse σ as (S, T_1, T_2, T_3).
2. If $e(T_1, Y) = e(T_2, g_2)$ then output 0.
3. Check that S is a valid SoK signature on message m as follows:
   - Parse S as (c, s_{ξ_i}).
   - Compute $R_1' = e(T_2, X)^{s_{\xi_i}} / \quad \frac{e(T_3,g_2)}{e(T_1,X)}^{\;c}$.
   - Check that
   $$c \overset{?}{=} \mathsf{Hash}_1(gpk, \tfrac{e(T_3,g_2)}{e(T_1,X)}, R_1').$$
   If the check fails then output 0.
4. For all $\boldsymbol{grt}[i] = w_i \in RL$ check whether $e(T_3, g_2) \overset{?}{=} e(T_1, X)e(T_2, w_i)$ holds. If no such w_i exists then output 1; otherwise output 0.

The idea of Bichsel et al.'s VLR construction is to integrate a part of the opening procedure into the verification. The needed modifications are possible in the sub-class of schemes, where it suffices for the Open algorithm to take only $(gpk, m, \sigma, \boldsymbol{reg})$ as input instead of $(gmsk, m, \sigma, \boldsymbol{reg})$ (i.e., just the registration list $\boldsymbol{reg}$ is required to open a signature, not the group manager's secret key $gmsk$). Note that the BCNSW scheme (Section 7.3) falls into this sub-class. To be exact, the element w_i of a user's entry in $\boldsymbol{reg}$ suffices to identify the issuer of a signature, thus the group manager simply has to store w_i in RL in order to revoke a user i.

Implicit opening procedure. The implicit opening algorithm Open, extended to provide verifiable opening, takes as input the group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, X, Y)$, the vector of revocation tokens $\boldsymbol{grt}$, the registration list $\boldsymbol{reg}$, a message m, and a group signature σ and proceeds as follows:

1. Parse σ as (S, T_1, T_2, T_3).
2. Check for all $i \in [1, n]$ whether $\mathsf{GVrfy}(gpk, \boldsymbol{grt}[i], m, \sigma) \overset{?}{=} 0$. If no such i exists output 0.
3. For the first such i compute $k_i = e(g_1, r_i)$ and $J = (c, s_{w_i}, s_{K_i})$ as the NIZKPoK proof
   $$\mathsf{NIZKPoK} \quad w_i, K_i \quad : \frac{e(T_3,g_2)}{e(T_1,X)} = e(T_2, w_i) \text{ and } k_i = \frac{e(g_1,w_i)}{e(g_1,X)^{K_i}}$$
   using $\boldsymbol{reg}[i] = (w_i, r_i, K_i, \bar{\sigma}_i)$. This proof can be computed as follows:

- Choose $r_{w_i} \in_R \mathbb{G}_2$ and $r_{K_i} \in_R \mathbb{Z}_Q$.
- Compute $R_1 = e(T_2, r_{w_i})$ and $R_2 = e(g_1, r_{w_i})/e(g_1^{r_{K_i}}, X)$.
- Compute $c = \mathsf{Hash}_1(gpk, \sigma, m, k_i, \bar{\sigma}_i, R_1, R_2)$.
- Compute $s_{w_i} = r_{w_i} w_i^c$ and $s_{K_i} = r_{K_i} + cK_i \bmod Q$.

4. Output (i, τ) where $\tau = (k_i, \bar{\sigma}_i, J)$.

The NIZKPoK proof J prevents the group manager pointing to some member i for which the verification equation $e(T_3, g_2) = e(T_1, X)e(T_2, w_i)$ does not hold. Moreover, inclusion of k_i into the proof τ allows verification of the signature $\bar{\sigma}_i$, which uniquely identifies the signer.

Note that the opening operation is linear in the number of users in the system, which is reasonable if the group manager has sufficient resources and the operation is not performed too often.

Judgement procedure. The judgement algorithm Judge takes as input the group public key $gpk = (Q, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, X, Y)$, a message m, a group signature σ, an identity i, and proof τ, and proceeds as follows:

1. If $\mathsf{GVrfy}(gpk, m, \sigma) = 0$ then output 0.
2. Parse τ as $(k_i, \bar{\sigma}_i, J)$.
3. Retrieve $\boldsymbol{upk}[i]$.
4. If $\mathsf{Vrfy}(\boldsymbol{upk}[i], k_i, \bar{\sigma}_i) = 0$ then output 0.
5. Parse J as (c, s_{w_i}, s_{K_i}).
6. Check the validity of proof J as follows:
   - Compute $R_1' = e(T_2, s_{w_i})e(T_1^c, X)/e(T_3^c, g_2)$ and $R_2' = e(g_1, s_{w_i})/ \ \ e(g_1^{s_{K_i}}, X) \cdot k_i^c$.
   - Check that $c \stackrel{?}{=} \mathsf{Hash}_1(gpk, \sigma, m, k_i, \bar{\sigma}_i, R_1', R_2')$. If the check succeeds then output 1, otherwise output 0.

Through the verification of the NIZKPoK proof J the judgement procedure obtains confidence that user i has been chosen correctly from $\boldsymbol{reg}$ in the opening step. Furthermore, it ensures the validity of the group signature. The actual identification of the signer i is performed using the signer's PKI-certified public key $\boldsymbol{upk}[i]$ and the signature $\bar{\sigma}_i$. It is implicitly assumed that identity i points to the candidate public key $\boldsymbol{upk}[i]$ used in the final verification step.

14.2. Performance Heuristics for Group Management and Verifiable Opening

We start our performance analysis of the BCNSW-VLR scheme with algorithms and protocols that have only marginal impact on the performance on the scheme in practice. This includes:

the key generation algorithm GKg that is executed only once during the life time of the group; the algorithms JoinM and JoinU for handling the admission of new group members, the opening procedure Open, and the judgement procedure Judge that can be used to verify the correctness of the opening procedure. Note that GKg, JoinM, and Open are typically executed on powerful devices, representing the group manager, whereas Judge is performed by users with, potentially, less powerful devices. However, Judge can be seen as a relatively rare operation. The user's part of the admission procedure, the algorithm JoinU, is performed only once per member.

We analyze the performance of these algorithms using our heuristic approach. That is, in Table 14.1 we first count the amount of most dominant operations in the bilinear map setting used by the BCNSW-VLR scheme.

Notable is that all algorithms in Table 14.1 have a constant number of exponentiations and pairing evaluations, except for Open and Judge that have linearly increasing costs due to the implicit execution of the GVrfy algorithm, whose performance we analyze in Section 14.3. We stress, however, that t'_{GVrfy} used in further costs of Open denotes the running time of GVrfy on input the (secret) list of revocation tokens $\boldsymbol{grt}$ instead of the (public) revocation list RL, which is normally used for verification; t'_{GVrfy} thus corresponds to the execution of GVrfy with up to m revocation checks, where m is the total size of the group. In contrast, t_{GVrfy} used in further costs of Judge is the regular execution time of GVrfy on input RL. We also observe that in the JoinM part of the joining protocol the group manager has to verify a digital signature $\bar{\sigma}_i$ that she receives from the joining member. The costs for its generation are added to the user's algorithm JoinU. This signature can be computed using any (unforgeable) digital signature scheme Σ and its specification is left open in the BCNSW-VLR scheme.

Table 14.1.: BCNSW-VLR Scheme: Dominant Operations in Group Management and Verifiable Opening

Operation	GKg	JoinM	JoinU	Open	Judge
exponentiations in $\mathbb{G}_1$	0	12	9	1	3
exponentiations in $\mathbb{G}_2$	2	5	6	1	0
exponentiations in $\mathbb{G}_T$	0	0	0	0	1
pairing evaluations $e(\cdot,\cdot)$	0	0	3	4	5
further significant costs	–	t_{Vrfy}	t_{Sign}	t'_{GVrfy}	$t_{\mathsf{Vrfy}} + t_{\mathsf{GVrfy}}$

As a second step in our analysis, we use Table 14.1 to obtain heuristic running times of all mentioned algorithms at the chosen security level of approximately 128 bits. For this purpose, we use our reference platforms and timing measurements for exponentiations and pairing evaluations from Section 11.3 and summarize the resulting estimates in Table 14.2.

In the assumed bilinear map setting exponentiations in $\mathbb{G}_2$ and pairing evaluations $e(\cdot,\cdot)$ belong to the most expensive operations and thus dominate the execution time of the analyzed algorithms. We observe that estimated timings of all algorithms that have constant costs, namely GKg, JoinM, and JoinU remain significantly below one second on our PC platform and may reach up to three seconds on our Smartphone platform. These timings, including

Table 14.2.: BCNSW-VLR Scheme: Performance Heuristics for Group Management and Verifiable Opening

Platform	Order of $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$	Estimated Time GKg	JoinM	JoinU	Open	Judge
PC	332 bits	185.2 ms	591.4 ms	927.3 ms	470.5 ms	515.6 ms
Smartphone	332 bits	594.8 ms	1775.0 ms	2850.0 ms	1454.2 ms	1568.7 ms
further significant costs		–	t_{Vrfy}	t_{Sign}	t'_{GVrfy}	$t_{\mathsf{Vrfy}} + t_{\mathsf{GVrfy}}$

the running time of about three seconds for JoinU on commodity smartphones might still be acceptable in practice due to the expectedly rare execution of those algorithms.

14.3. Performance Heuristics for Signature Generation and Verification

In the following we provide intuition about the performance of BCNSW-VLR signature generation and verification procedures, which we estimate in two steps. First, in Table 14.3, we summarize the amount of most dominant operations within the assumed setting of bilinear maps. That is we count the total amount of exponentiations in groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, and of pairing evaluations $e(\cdot,\cdot)$ used by these algorithms. We notice that none of these algorithms requires exponentiations in $\mathbb{G}_2$, which according to Section 11.3 have the highest costs. Nonetheless, the verification procedure of the BCNSW-VLR scheme is dominated by a linear number of pairing evaluations, which are only slightly more efficient than exponentiations in $\mathbb{G}_2$. The linear increase in the number of pairing evaluations depends on the number of revoked group members. Leaving out those costs we observe that the verification procedure would still be costlier than the signing procedure of the scheme.

Table 14.3.: BCNSW-VLR Scheme: Dominant Operations in Signature Generation and Verification

Operation	GSign	GVrfy
exponentiations in $\mathbb{G}_1$	3	0
exponentiations in $\mathbb{G}_2$	0	0
exponentiations in $\mathbb{G}_T$	1	2
pairing evaluations $e(\cdot,\cdot)$	3	$5 + r$

r — number of revoked members.

Further, we use Table 14.3 to obtain heuristics for running times of signature generation and verification algorithms. For this purpose we use our reference platforms and measured timings of dominant operations in the bilinear map setting from Section 11.3. The resulting estimates are provided in Table 14.4.

Table 14.4.: BCNSW-VLR Scheme: Performance Heuristics for Signature Generation and Verification

Platform	Order of $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ ($\approx$ 128-bit security)	Estimated Time GSign	Estimated Time GVrfy
PC	332 bits	332.0 ms	$508.0 + 91.8r$ ms
Smartphone	332 bits	1002.3 ms	$1577.4 + 283.2r$ ms

r — number of revoked members.

We observe that it would take about 332 ms to generate a BCNSW-VLR group signature on our PC platform and slightly more than a second on our Smartphone. In the next section we discuss scalability of the verification procedure, based on our measurements and those obtained from the literature.

14.3.1. Scalability of the Verification Procedure

The verification time of BCNSW-VLR group signatures depends on the length of the revocation list RL. Therefore, we are interested in the scalability of the verification procedure considering the potential increase of $|RL| = r$. As shown in Figure 14.1 the increase of r has linear impact on the verification time, i.e. the verification procedure would require somewhat more than 500 ms plus roughly 92 ms per each revoked member on our PC platform and more than 1.5 seconds plus roughly 284 ms per each revoked member on our Smartphone platform.

Notice that it would take up to r revocation checks to detect a group signature issued by a revoked group member in the worst case. For groups with up to 1000 revoked members, the entire verification procedure may take between one and a half and almost five minutes, depending on the platform, which can already be critical for the practical use of the scheme. In larger groups with up to 100,000 resp. one million revoked members it would take over two hours resp. more than one day based on our measurements, which is far from being practical. We observe that these timings are estimated on our platforms using the open-source PBC library, which however is not as efficient as some recent literature reports show.

Therefore, we again refer to our literature survey from Table 11.4 and provide further estimations, using the fastest PC implementation of pairings from [145], which takes roughly 1.55 ms, and the fastest implementation from [187] that achieved 0.66 ms on an FPGA platform.

In Figure 14.1 we illustrate the impact of these state-of-the-art pairing implementations on the verification procedure of the BCNSW-VLR scheme. We observe that the expected verification time on a PC platform can be drastically reduced to about one and a half seconds for 1000 revocation checks, less than three minutes for 100,000 revoked users, and less than 26 minutes for one million revocation checks. With the fastest implementation on FPGAs these timings could be reduced even further, i.e. to less than a second for 1000 revocation checks, slightly more than one minute for 100,000 checks, and about 11 minutes for one million checks. Although

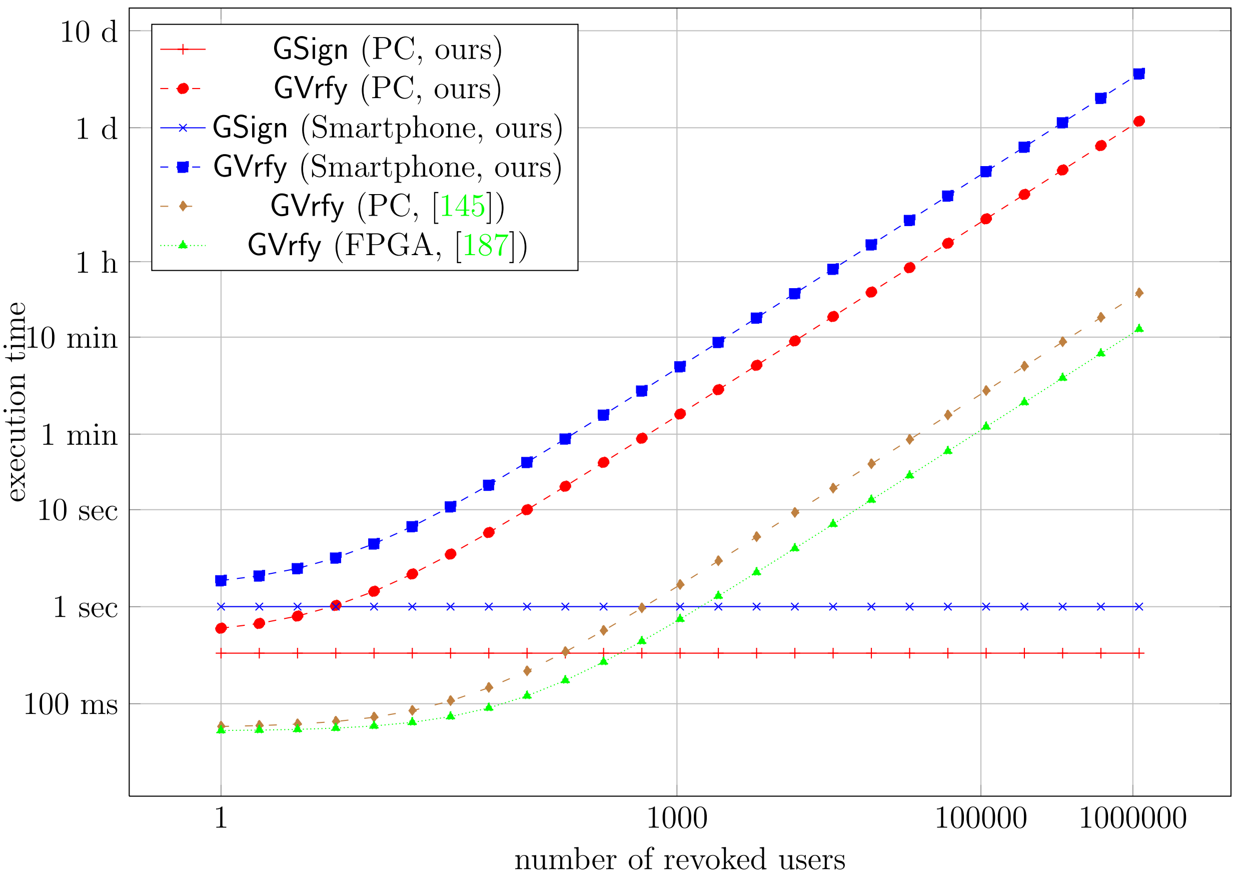


Figure 14.1.: BCNSW-VLR Scheme: Scalability of Signing and Verification Procedures

scalability of the BCNSW-VLR verification procedure can be significantly improved using state-of-the-art implementations of pairings, its costs remain the main performance bottleneck of the scheme.

14.4. Space Requirements for the Main Parameters

In Table 14.5 we estimate the required amount of space to store the main private and public parameters of the BCNSW-VLR scheme. More precisely, we consider lengths of the group public key gpk, individual secret signing keys $\boldsymbol{gsk}[i]$, output group signatures σ, the group manager's secret key $gmsk$, whereby also counting the size of (secret) registration lists $\boldsymbol{reg}$, and of the public revocation list RL. Note that we do not count sizes of ordinary signatures $\bar{\sigma}_i$ which are produced by member i during the admission procedure and are stored by the group manager as part of $\boldsymbol{reg}[i]$ since the signature scheme is left unspecified.

We estimate storage requirements for these parameters using our previous analysis of space complexity from Table 10.4 and applying the concrete lengths of parameters for the chosen security level of 128 bits. As a measurement unit in the table we use 1 Kb = 1024 bits.

We observe first that several parameters of the BCNSW-VLR scheme have constant length.

Table 14.5.: BCNSW-VLR Scheme: Space Requirements

Order of $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$	gpk	$\boldsymbol{gsk}[i]$	σ	$gmsk$ and $\boldsymbol{reg}$	RL
332 bits	3.79 Kb	1.38 Kb	1.70 Kb	$0.65 + (2.40 + \lvert\bar{\sigma}_i\rvert)(m - r)$ Kb	$1.04r$ Kb

m — number of group members; r — number of revoked members.

This refers to group public keys, secret signing keys, and output group signatures. The secret information in each $\boldsymbol{gsk}[i]$ that could potentially be stored in a protected memory is lower than the size of group public keys and signatures that are used in the verification procedure. We also observe that group signatures are smaller than group public keys by at least a factor of two.

The BCNSW-VLR scheme is dynamic. Therefore, each time a new member is added to the group through the joining procedure, the group manager has to add a new entry into its registration list $\boldsymbol{reg}$, which must be kept private in addition to the group manager's secret

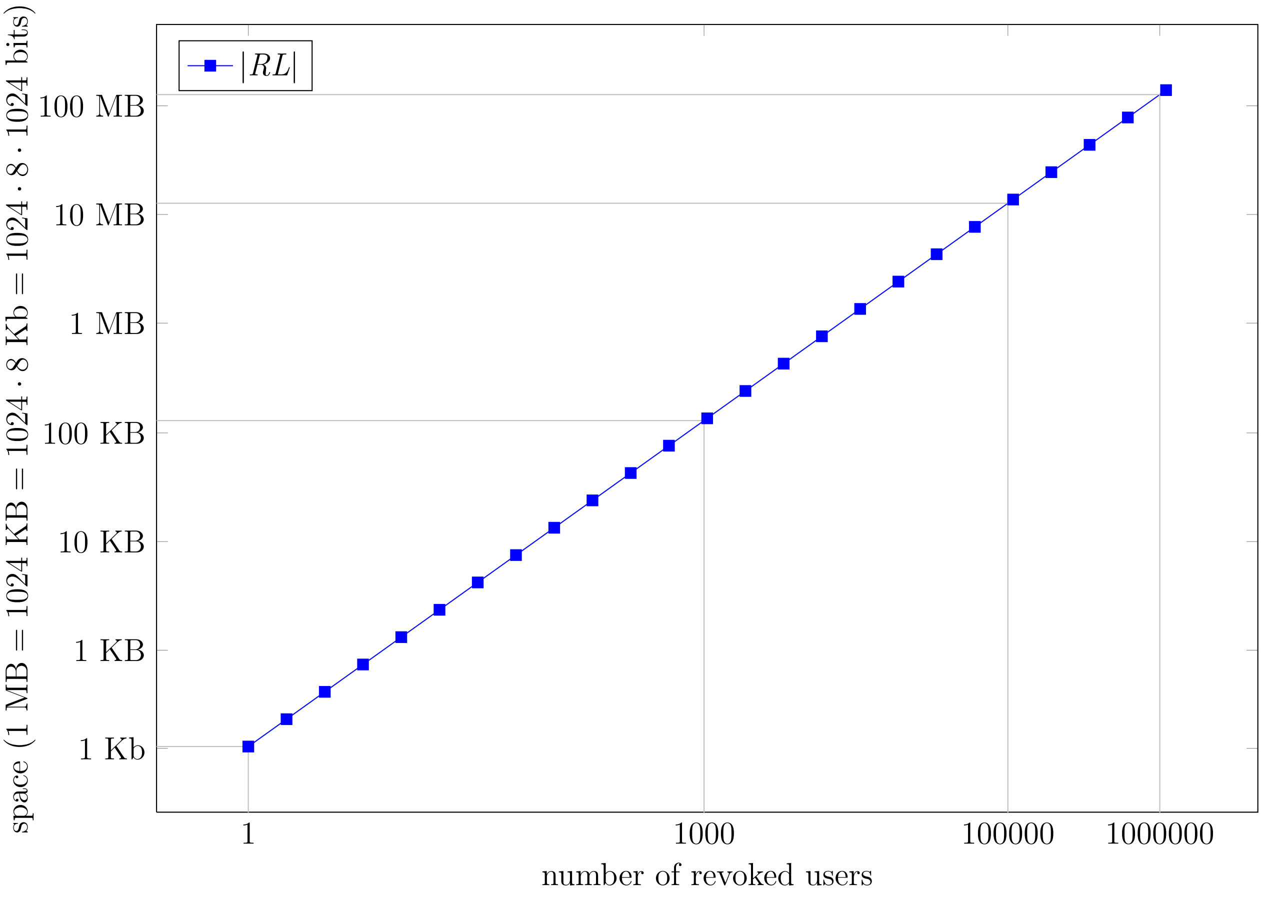


Figure 14.2.: BCNSW-VLR Scheme: Scalability of Published Revocation Lists

key *gmsk*, which has constant size of about 665 bits and can be stored in a tamper-resistant memory. That is, the amount of secret information stored on the group manager's side increases by $2460+|\bar{\sigma}_i|$ bits with each joined member. The corresponding entry $\boldsymbol{reg}[i]$ must be kept secret until member i is revoked and can possibly be erased after the publication of the token in the revocation list RL.

Each time a group member is revoked the length of RL increases by 1064 bits. The scalability impact resulting from this increase is plotted in Figure 14.2. For up to 1000 revoked members we observe an increase by about 130 KB, for up to 100,000 entries in RL the amount of space increases by roughly 12.7 MB, and for up to one million entries the size of RL amounts to almost 127 MB.

15. Performance and Scalability Comparison

In this section we apply our heuristic evaluations from Chapters 12 to 14 to provide a direct comparison of the Camenisch-Groth (CG) [49], Boneh-Shacham (BS) [38], and Bichsel et al. (BCNSW-VLR) [31] group signature schemes with regard to their performance and scalability.

15.1. Performance and Scalability

15.1.1. Performance Comparison for Group Management and Opening

In this section we compare the three schemes based on their performance heuristics with regard to the group management and the opening of group signatures. We start with algorithms that appear in all three constructions, namely GKg for generation of keys and Open for identification of signers.

The key generation procedure GKg takes constant amount of time in the dynamic CG and BCNSW-VLR schemes and has linear increase in the total number of members m for the static BS scheme. The CG scheme results in the most efficient key generation procedure on our reference platforms (PC and Smartphone), if we base comparison on our measurements with PBC library. By using state-of-the-art implementation of pairings the BCNSW-VLR scheme would become more efficient though. However, this improvement is not significant in practice, as key generation is a fairly practical operation in both schemes. Since GKg is executed only once, the linear amount of time for key generation in the BS scheme does not render that scheme impractical either.

The opening procedure Open of all three schemes makes implicit use of the verification algorithm GVrfy and thus has linear execution time. However, in the CG scheme this time will depend on the number of revoked members r since GVrfy is only used to rule out signatures computed by revoked members. The actual identification of the signer in the CG scheme takes constant amount of time and makes this scheme most efficient in this respect. In the two VLR schemes, BS and BCNSW-VLR, identification of the signer takes linear amount of time for GVrfy and depends on the difference $(m - r)$, which is usually larger than r. Based on our performance comparison of GVrfy, provided in the next section, we expect the BCNSW-VLR opening procedure to be more efficient in practice than the same procedure of the BS scheme.

The only non-VLR scheme in our analysis is the CG scheme, which thus brings an additional algorithm UpdM for the group members to update their secret signing keys once the group manager publishes the required update information through the corresponding algorithm Revoke.

We observe that costs of those two algorithms can be seen as negligible, in comparison to the costs of other algorithms provided by the CG scheme, in particular UpdM and Revoke do not utilize any costly cryptographic operations.

The BCNSW-VLR scheme is the only scheme in our analysis, which provides verifiable opening. For this purpose it brings the algorithm Judge, whose execution time is linear in the number of revoked members r, due to the implicit use of the verification procedure GVrfy that rules out signatures of revoked signers. In comparison to this the additional time for verification of an ordinary signature $\bar{\sigma}_i$ remains negligible. The judgement procedure, however, does not have a significant impact on the practical use of the BCNSW-VLR scheme since Judge is a rather rare operation that would usually be performed on a resourceful device.

15.1.2. Performance Comparison for Signature Generation and Verification

In Table 15.1 we compare timings of signing and verification procedures that were estimated using our performance heuristics on both reference platforms (PC and Smartphone). Recall that those heuristics were calculated based on the execution time of the most dominant operations in Chapter 11 and projected to the number of operations that are required in the respective signing and verification algorithms. Additionally, for BS and BCNSW-VLR schemes, Table 15.1 estimates those costs based on two fastest pairing implementation from the literature on the PC [145] and FPGA [187] platforms.

Table 15.1.: Comparison of Signing and Verification Procedures

Group Signature Scheme	Context	GSign	GVrfy
CG	PC, 2048-bit modulus	170.4 ms	$170.4 + 6.3r$ ms
	PC, 3248-bit modulus	794.0 ms	$794.0 + 21r$ ms
	Smartphone, 2048-bit modulus	431.5 ms	$431.5 + 16.0r$ ms
	Smartphone, 3248-bit modulus	1356.7 ms	$1356.7 + 35.4r$ ms
BS	PC	402.4 ms	$691.6 + 91.8r$ ms
	Smartphone	1211.7 ms	$2118.0 + 283.2r$ ms
	PC, [145]	131.4 ms	$150.1 + 1.55r$ ms
	FPGA, [187]	128.7 ms	$144.76 + 0.66r$ ms
BCNSW-VLR	PC	332.0 ms	$508.0 + 91.8r$ ms
	Smartphone	1002.3 ms	$1577.4 + 283.2r$ ms
	PC, [145]	61.0 ms	$56.8 + 1.55r$ ms
	FPGA, [187]	58.3 ms	$52.3 + 0.66r$ ms

r — number of revoked members.

Observe that in all three schemes the signing procedure takes constant amount of time. For our measurements, based on the MNT curve D476971 from the PBC library, we can see that signature generation in the BS and BCNSW-VLR schemes is less efficient than in the CG scheme. For example, performance of CG signature generation on Smartphone for the selected security level is comparable to the generation of BS signatures on PC if the latter is implemented using the mentioned MNT curve. However, using state-of-the-art pairing implementations the signing performance of the BS and BCNSW-VLR schemes can be significantly improved and would outperform the CG scheme.

Verification performance in all three schemes depends on two factors: first, there is a constant amount of time in each group signature scheme, which is needed to check whether the signer is in possession of the secret membership credential for a particular group; second, there is a linearly increasing amount of time, which depends on the number of revoked users and is needed to perform the corresponding revocation checks. We observe that the CG scheme has a more efficient verification procedure than the pairing-based BS and BCNSW-VLR schemes as long as the open-source PBC library is used. However, faster implementation of pairings from the literature would lead to significant performance improvements of those schemes, which would be superior to the performance of the CG scheme.

We remark that our PC platform is not as powerful as the PC platform from [145]. Therefore, underlying operations of the CG scheme, when measured on a platform similar to [145], would result in better timings. We may expect a factor of two in performance increase for the most dominant operations in the RSA setting, in comparison to the measurements on our reference platform. This intuition is based on some additional measurements that we performed on the Xeon E5430 2.66 GHz platform. In this case we may still assume that performance of pairing-based BS and BCNSW-VLR schemes would prevail over that of the CG scheme.

Summary. Our heuristic approach for performance estimation allows us to make the following conclusions: (1) using state-of-the-art implementation of pairings the signing and verification procedures of the BS and BCNSW-VLR would outperform those of the CG scheme; (2) the BCNSW-VLR scheme has better signing and verification performance than the BS scheme; (3) with about one second for signature generation on our Smartphone platform and sometimes even higher verification times all three schemes seem to be on the border of practicality on such platforms; (4) in contrast, state-of-the-art implementations seem to indicate that all three schemes would perform fairly well in practice on a PC platform, and as a consequence on an FPGA platform.

15.1.3. Comparison of Verification Scalability with Revocation Checks

In this section we compare the scalability of verification procedures. Figure 15.1 shows estimated verification performance of the CG, BS, and BCNSW-VLR schemes, as a function of the number of revoked members, based on our own measurements for the BS and BCNSW-VLR schemes using the PBC library.

Although all three schemes have linear increase for verification, the CG scheme has better scalability than the pairing-based BS and BCNSW-VLR schemes. That is, its verification time remains shorter for both modulus sizes (2048 and 3248 bits). It takes less than two seconds to verify CG signatures in the presence of 150 revoked members, whereas verification

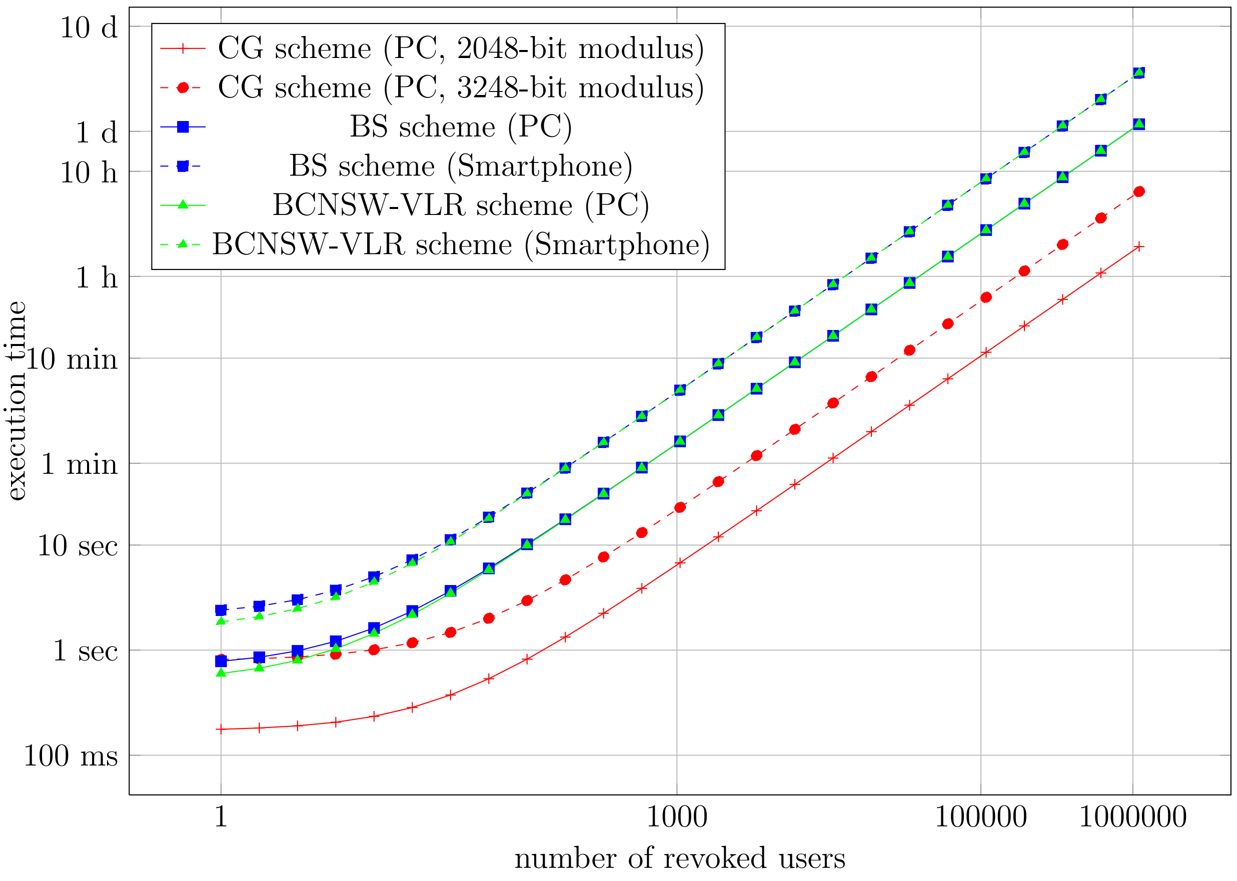


Figure 15.1.: Scalability of Verification Procedure with Revocation Checks (Our Measurements)

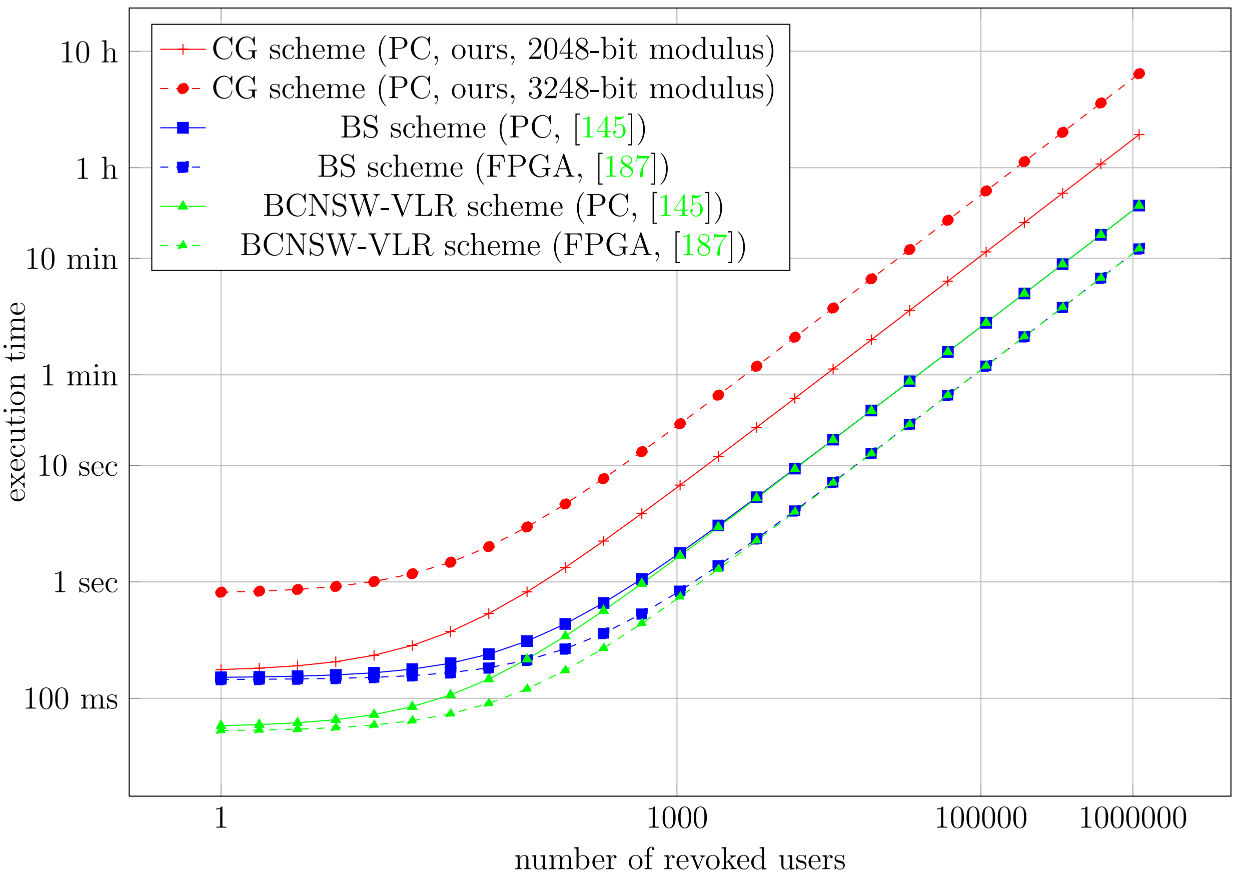


Figure 15.2.: Scalability of Verification Procedure with Revocation Checks (Literature Reports)

of BS and BCNSW-VLR signatures would need about the same amount of time to perform a single revocation check. We observe that these results are due to the use of a non-optimized implementation with the PBC library.

For this reason, in Figure 15.2, we also plot verification times calculated using literature reports for pairing evaluations from Table 11.4. More precisely, we consider 1.55 ms for a pairing evaluation on a PC [145], and 0.66 ms on an FPGA [187]. We observe a significant increase in performance of the BS and BCNSW-VLR schemes, not only in comparison to our PBC-based calculations but also, and more importantly, in comparison to the CG scheme.

For example, within roughly one second it seems feasible to verify BS group signatures with up to 550 revoked members on a PC and up to 1300 revoked members on an FPGA. The BCNSW-VLR scheme shows a slightly better performance for smaller revocation lists, i.e. it would allow verification with up to 600 revoked members on a PC and up to 1440 revoked members on an FPGA within the same time frame. Interestingly, starting with roughly 1000 revoked members both pairing-based schemes show very similar verification performance and scalability. The reason is that the impact of the initial timing difference of their verification procedures, which is about 100 ms on a PC and about 90 ms on an FPGA using calculations based on [145, 187] in Table 15.1, starts to become negligible for sufficiently large revocation lists.

Summary. Our heuristic approach applied for the scalability analysis of verification procedures in Figures 15.1 and 15.2 leads to the following results: (1) using state-of-the-art implementation of pairings the scalability of verification procedures in the BS and BCNSW-VLR would outperform that of the CG scheme, and (2) the BCNSW-VLR scheme shows better scalability for revocation checks than the BS scheme, yet for sufficiently large revocation lists (roughly 1000 members and more) the performance difference between the two pairing-based schemes becomes negligible.

15.1.4. Impact of Scalability on Group Sizes

Our previous scalability analysis addressed the increasing size of revocation lists. In practice, we are interested in total group sizes that can be efficiently handled by the respective schemes. Due to the constant signing costs in all three schemes, larger group sizes do not have any impact on the performance of their signature generation procedures. However, scalability of their verification procedures depends solely on the number of revoked members. Hence, if none of the members gets revoked the increasing group size wouldn't matter either. Yet, this assumption is too strong to be justifiable in practice, where revocation should be taken into account.

We cannot estimate tolerable group sizes without stating additional assumptions on the relationship between the number of revoked members and the total group size. We will thus apply an intuitive argument that in larger groups revocations are more likely to occur. To make our analysis more explicit we will consider two cases: we will assume that at any given point in time the number of revoked members does not exceed either 10% or 25% of the total group size, and perform scalability analysis with regard to four different types of groups: *small groups* with less than 1000 members, *medium groups* ranging between 1000 and 100,000 members, *large groups* ranging between 100,000 and one million members, *very large groups* with more than one million members.

Case 10%**.** Figure 15.3 illustrates scalability of the verification procedure assuming that at each point in time the number of revoked members corresponds to 10% of the total group size. These numbers are based on fast implementations of pairings from the literature and verification timing of the three group signature schemes were calculated using our heuristic approach from the previous parts of the analysis.

We observe that for small groups all three schemes perform reasonably well, in particular their verification time would remain below one second in most cases (the only exception is the CG scheme with 3248 bit modulus). The only scheme that would achieve verification time of at most 200 ms in small groups with up to 1000 members is the BCNSW-VLR scheme. In medium groups, verification performance of BS and BCNSW-VLR schemes is close to each other, it would take them roughly one second to verify one group signature for groups of maximal size between 5500 members on a PC and 13000 members on an FPGA. In medium to large groups with around 100,000 members, verification of BS and BCNSW-VLR signatures will take roughly between 7 and 15 seconds. This seems to be impractical, at least for applications where the signature verification process cannot tolerate significant delays. In larger groups with up to one million members only FPGA implementations would offer verification timings of less

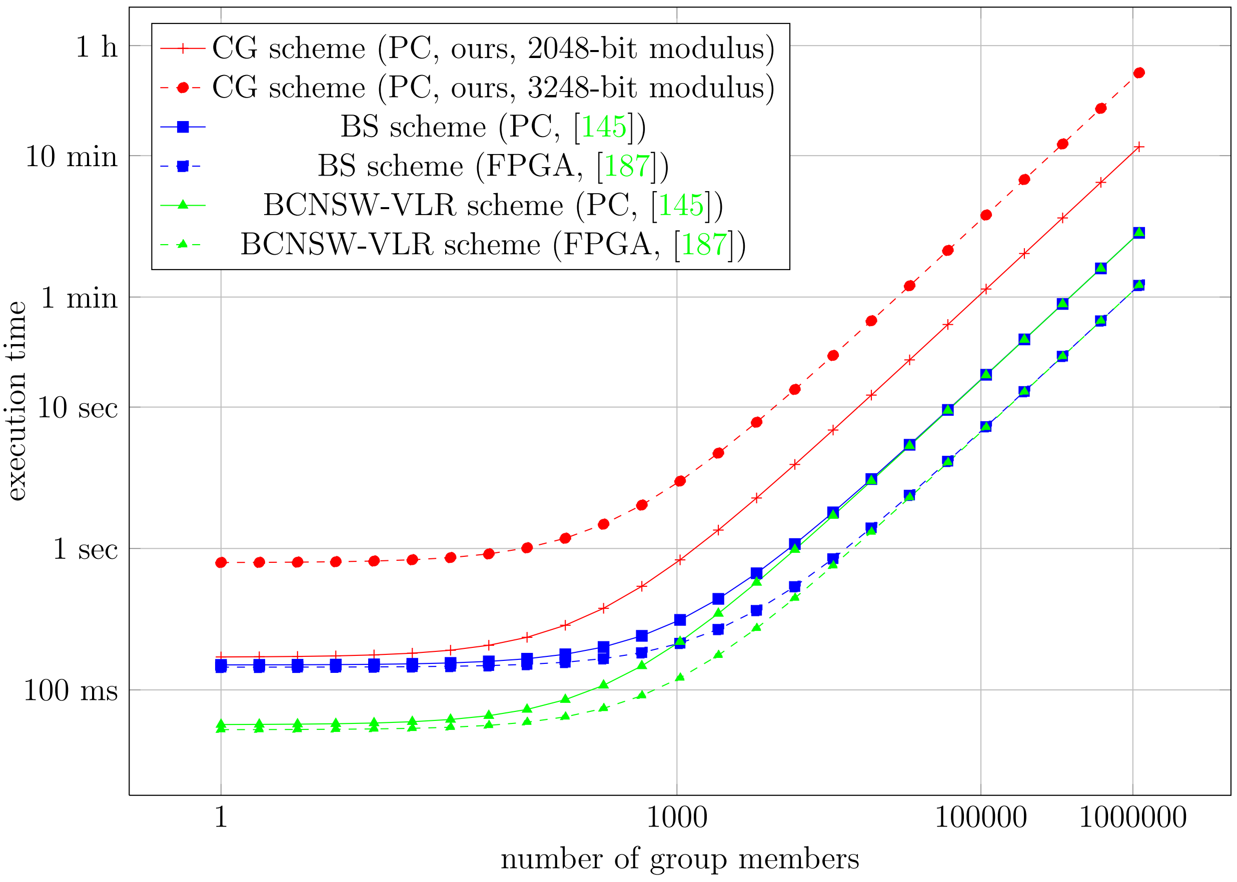


Figure 15.3.: Scalability Impact on Group Sizes (10% revoked members)

than one minute. When it comes to very large groups with more than one million members, the verification process of all three schemes would take several minutes and more.

Case 25%**.** Figure 15.4 performs a similar analysis, but assuming 25% of revoked group members. We observe that this threshold might be too high for practical purposes, yet is sufficient to obtain some intuition on how the percentage of revoked group members impacts the scalability. Also due to the higher fraction of revoked members, in comparison to the previous case, we observe the expected general increase of the verification effort.

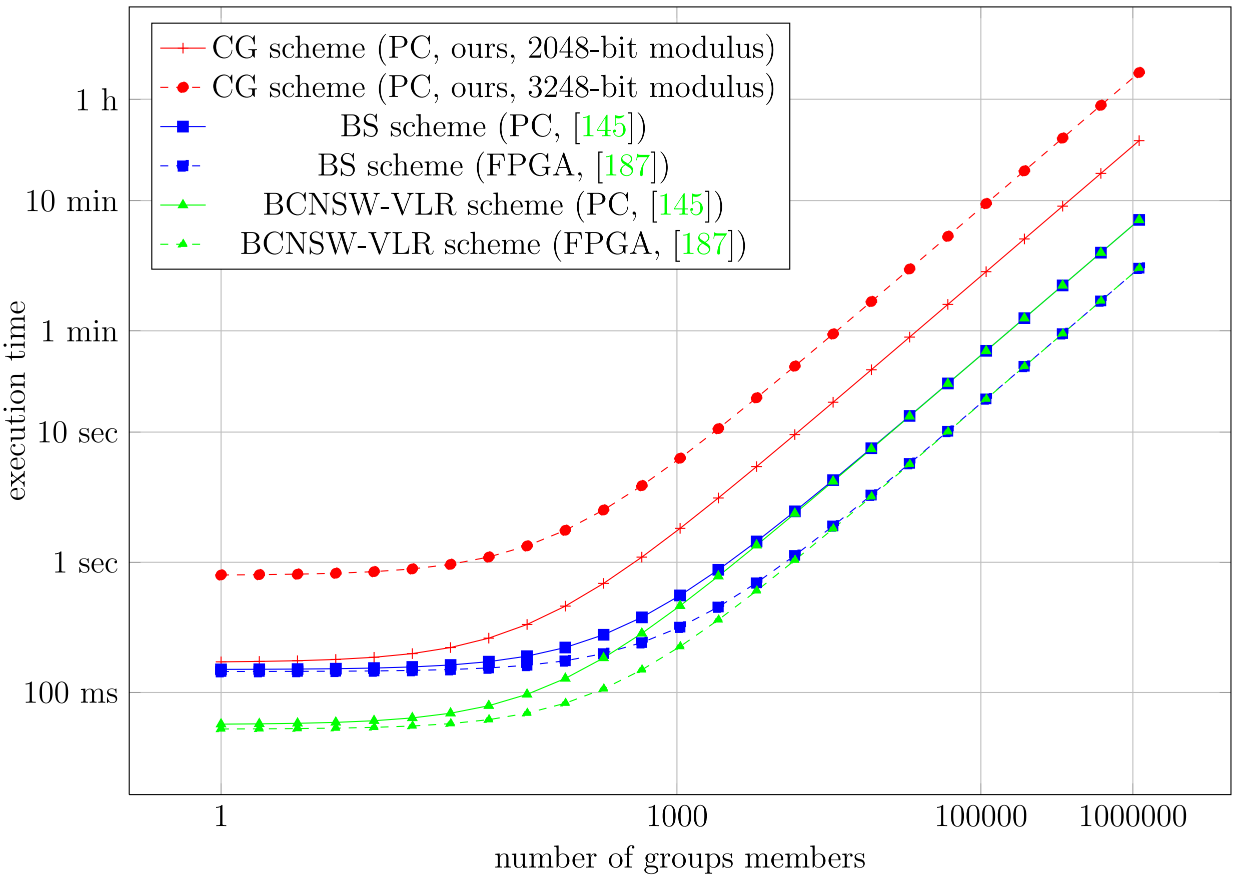


Figure 15.4.: Scalability Impact on Group Sizes (25% revoked members)

In small groups with up to 1,000 members, the pairing-based BS and BCNSW-VLR schemes perform reasonably well, offering verification times of up to one second. The CG scheme is slightly worse in this respect, i.e. in the range of a second CG would be able to perform verification for groups of up to 500 members assuming 2048-bit modulus, or for groups of up to 100 members with 3248-bit modulus. Verification time of BS and BCNSW-VLR signatures seems to exceed one second for 2000 members on a PC and 5000 group members on an FPGA. In medium to large groups with around 100,000 members, all three schemes are with at least a few seconds for a single signature verification at the borderline of practicability – or, in the case of the CG scheme, already far beyond, with corresponding timings of up to 10 minutes.

Needless to say, practical applications in large or very large groups, with up to one million or even more members, become time consuming with all three schemes, requiring several minutes or even hours to verify a single group signature.

Summary. Our heuristic approach, applied in the underlying analysis of verification procedures in Figures 15.3 and 15.4, leads to the following results: (1) using state-of-the-art implementation of pairings, it seems that the BS and BCNSW-VLR schemes can handle larger groups than the CG scheme; (2) in small groups with up to 1000 members the BCNSW-VLR scheme shows better performance and should be preferred over the BS scheme; (3) in groups with more than 1000 members both pairing-based schemes show similar performance.

15.2. Space Requirements for Secret and Public Parameters

15.2.1. Space Requirements for Secret Parameters

This section compares the amount of space, which would be required by each group signature scheme to store the most significant secret parameters used in its algorithms. These space requirements are calculated based on the estimated space complexity from Table 10.4 using concrete parameter lengths, as discussed in Chapter 10.

Table 15.2 summarizes the required amount of space for the group manager's secret key *gmsk* (while also counting storage of secret information that is disclosed during the revocation procedure) and for the individual signing keys $\boldsymbol{gsk}[i]$.[1]

Table 15.2.: Space Requirements for Secret Parameters

Group Signature Scheme		*gmsk* with $\boldsymbol{reg}$ or $\boldsymbol{grt}$	$\boldsymbol{gsk}[i]$
CG	2048-bit modulus	$2.41 + 5.15(m - r)$ Kb	6.89 Kb
	3248-bit modulus	$3.58 + 7.49(m - r)$ Kb	10.41 Kb
BS		$0.35(m - r)$ Kb	0.68 Kb
BCNSW-VLR		$0.65 + 2.40(m - r)$ Kb	1.38 Kb

m — number of group members; r — number of revoked members.

All three schemes require the group manager to store secret information, whose length depends linearly on the number of unrevoked members, i.e. the difference between the total current group size m and the number of currently revoked members r. In the BS scheme, which is static, the initially required space will only decrease with each revocation event; whereas in the dynamic CG and BCNSW-VLR schemes it may grow, depending on the population progress

[1]Note that for the BCNSW-VLR scheme we do not count sizes of ordinary signatures $\bar{\sigma}_i$ which are produced by member i during the admission procedure and are stored by the group manager as part of $\boldsymbol{reg}[i]$ since the signature scheme is left unspecified.

in the group. We observe that the amount of secret information that should be stored on the group manager's side in the CG scheme is significantly higher than in both pairing-based constructions. A direct comparison of BS and BCNSW-VLR schemes, shows that the BS scheme is more economical in this respect.

In small groups, with up to 1000 members, these differences are rather unnoticeable, i.e. BS would require up to 44 KB space, in comparison to about 300 KB for BCNSW-VLR and about 645 KB resp. 937 KB for the CG scheme. In medium groups with up to 100,000 members these differences would become somewhat noticeable, yet still fairly practical, with space requirements of up to 4.3 MB for BS, 29.3 MB for BCNSW-VLR, and 62.9 MB resp. 91.5 MB for the CG scheme. In large groups with up to one million members the corresponding space for secret information on the group manager's side would grow to at most 42.8 MB for BS, 293.0 MB for BCNSW-VLR, and 628.7 MB resp. 914.3 MB for the CG scheme.

In contrast, individual secret signing keys $\boldsymbol{gsk}[i]$ remain of constant length in all three schemes. The pairing-based BS and BCNSW-VLR schemes clearly outperform the CG scheme also in this parameter. In particular, their individual secret signing keys of lengths 0.68 Kb resp. 1.38 Kb can possibly be stored in trusted hardware modules; for example unique endorsement keys that are hard-coded inside modern TPM chips, as specified by TCG (`www.trustedcomputinggroup.org`), have sizes of 2 Kb.

Summary. The amount of secret information that has to be stored on the group manager's side is linear in all three schemes. Yet, in practice, where the role of the group manager is typically performed on devices with high computational and storage resources, the impact of this linear increase remains rather small. All three schemes seem to offer an acceptable overhead with regard to these space requirements, even for very large groups, whereby the BS scheme is the most economical one. As for the secret signing keys, we observe that both pairing-based BS and BCNSW-VLR constructions have very short secret signing keys that could even be encoded into hardware, for higher protection. Nonetheless, also CG scheme has acceptable storage costs for individual signing keys on most computing platforms (incl. mobile devices).

15.2.2. Space Requirements for Public Parameters

This section compares space requirements for most significant public parameters of the three group signature schemes. The amount of space is calculated using the storage costs from Table 10.4, by considering concrete lengths from Chapter 10. In Table 15.3 we summarize the amount of space for the group public key gpk, the size of revocation lists RL (applies to BS and BCNSW-VLR schemes) or update information $\boldsymbol{upd}$ that has to be made public to allow unrevoked group members to update their secret signing keys (applies to CG scheme), and the length of the output group signature σ.

All three schemes have group public keys of constant size, whereby the size of gpk in pairing-based BS and BCNSW-VLR schemes is at least five times smaller than in the CG scheme. However, up to 33 Kb large group public keys in the CG scheme seem still suitable for most practical applications.

Similarly, all three constructions output constant-size group signatures. The shortest group signatures (of only 1.7 Kb length) can be produced with the BS scheme, followed by the

Table 15.3.: Space Requirements for Public Parameters

Group Signature Scheme		gpk	$RL/\boldsymbol{upd}$	σ
CG	2048-bit modulus	20.41 Kb	$1.15r$ Kb	14.79 Kb
	3248-bit modulus	32.13 Kb	$1.15r$ Kb	21.82 Kb
BS		2.75 Kb	$0.35r$ Kb	2.32 Kb
BCNSW-VLR		3.79 Kb	$1.04r$ Kb	1.7 Kb

r — number of revoked members.

BCNSW-VLR scheme, whose signatures are longer by roughly 635 bits. The CG scheme outputs signatures that are longer by several factors, i.e. between 14.79 Kb and 21.82 Kb, depending on the setting. Nonetheless, also these lengths seem to be practical for many applications.

In contrast, the amount of published information that handles revocation increases linearly with the number of revoked members in all three constructions, as also illustrated in Figure 15.5. While the difference between the CG and BCNSW-VLR schemes is not very noticeable (about 112 bits per revoked user), the BS scheme with only about 360 bits per entry is the most economical one. Interestingly, such difference between the two pairing-based schemes originates from the fact that – although the entries in the revocation lists in both cases consist only of a single element – in the BS scheme this entry is an element from $\mathbb{G}_1$, and thus 360 bits long, whereas in the BCNSW-VLR scheme it belongs to $\mathbb{G}_2$, requiring 1064 bits.

Here we remark the significant difference to revocation in classical PKIs (cf. Section 1.1.2), where revocation is performed by publishing serial numbers of the revoked PKI certificates. These numbers are of non-cryptographic nature and their lengths are negligible in comparison to the amount of information that has to be published in a group signature scheme. The reason for such considerably larger overhead for revocation in group signatures is due to the fundamental requirement of unlinkability: while PKI certificates are linkable in nature, group signatures (of unrevoked signers) must remain unlinkable, therefore revocation information belonging to a particular group member must be of cryptographic nature and should not be leaked by any public parameters of the scheme or produced group signatures until that group member is revoked.

In order to illustrate practical impact of the storage costs for RL and $\boldsymbol{upd}$ on the total size of the group we give one further example. We use the same assumption as in the scalability of the verification procedure from Section 15.1.4, namely that at any point in time the number of revoked members does not exceed 10% of the total group size: In small groups with less than 1000 members the revocation information would require up to 4.4 KB for BS, 13 KB for BCNSW-VLR, and 14.4 KB for the CG scheme. In medium groups with up to $100,000$ members these space requirements would increase to 437.5 KB for BS, 1.3 MB for BCNSW-VLR, and 1.4 MB for the CG scheme. In large groups with up to one million members they would further grow to 4.3 MB for BS, 12.7 MB for BCNSW-VLR, and up to 14.1 MB for the CG scheme, and in very large groups with e.g. 10 million members, the space overhead would vary between 42.7 MB for BS and 140.4 MB for the CG scheme.

If we consider now as an example a download rate of 1.8 Mbit/s in a mobile setting, the

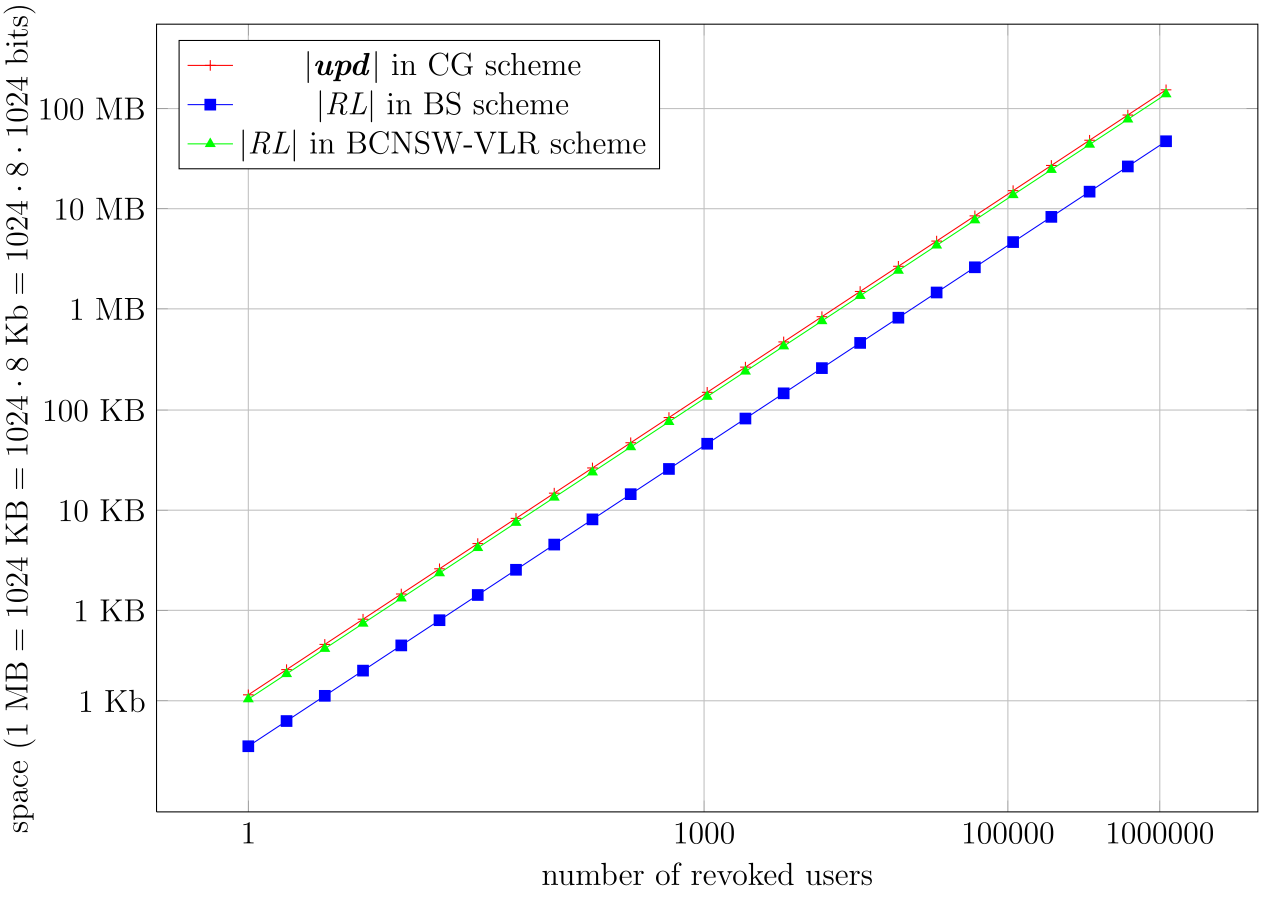


Figure 15.5.: Scalability of Published Revocation Information

download of the complete revocation information with any of the three schemes would require few milliseconds in small groups, several seconds for medium to large groups, and up to one minute and more in large or even very large groups with one million members or more. Considering fast broadband connection with a download rate of 16 Mbit/s, the required time for a complete download can be reduced to a fraction of a second even for group sizes of up to 100,000 members and to a few seconds or single minutes in large resp. very large groups.

Summary. The required time for a complete download of the public revocation information in the BS scheme is only one third of the time needed in the CG and BCNSW-VLR schemes. While this is certainly an interesting point when it comes to the decision, which of the three schemes shall be used, it seems that with respect to the space requirements and considering modern technologies in networking power and storage capacity, all three schemes would perform reasonably well in most practical applications.

Here we discussed the simplest approach where users retrieve the entire revocation information, i.e. by downloading it completely in one step. We notice, however, that there are many other techniques, e.g. those applied in the PKI setting, that would even further reduce the resulting overhead. For example, one could think of differential updates of the revocation information, i.e. users would have to only fetch new revocation information that they haven't downloaded yet. Another way to decrease the overhead is to "reset" the group upon fixed time intervals, in which case all previously published revocation information will become irrelevant. This approach, however, would introduce additional communication between the current group members and the group manager for the update of their group membership credentials. Our analysis demonstrates that even without such methods the overhead in all three schemes seems not critical for practical purposes.

15.3. Concluding Discussion

Our direct comparison of the three group signature schemes in this chapter shows that the pairing-based BS and BCNSW-VLR schemes have better performance than the CG scheme, especially when it comes to the most frequently executed signature generation and verification algorithms. The two pairing-based constructions can also handle larger group sizes than the CG scheme, as becomes obvious from our scalability analysis of the verification procedure. In small groups with up to 1000 members the BCNSW-VLR scheme seems to offer a better performance, while in larger groups the performance of BS and BCNSW-VLR is widely similar.

Nonetheless, we observe significant loss of performance, when it comes to groups of medium size, in the range of 100,000 members and more. The expected time for revocation checks in such groups seems to rule out practical deployment of those schemes, even if we put restrictions on the amount of revoked members in comparison to the total size of the group (e.g. our examples with 10% and 25% revoked members) and consider state-of-the-art implementations in pairing-based cryptography. For example, for groups with about 5500 members and more on a commodity PC platform and about 13000 members and more if cryptographic operations are implemented in hardware the required time of more than one second for revocation checks would already be impractical for many applications. In general, we observe that revocation remains the major bottleneck of modern group signature schemes and that further research is urgently needed to design schemes offering better scalability with regard to revocation.

On the other hand, there might be applications, where revocation is not of prime importance. For example, when the group signature scheme is deployed only for the purpose of anonymity and the actual accountability of member's actions, including later revocation of signing rights in case of misuse, is not required. In such scenarios revocation checks could be safely omitted from the corresponding verification procedures, whose execution times would then become constant. All three group signature schemes would then offer fairly practical performance, even in very large groups with several millions of members, whereby the pairing-based schemes would still be preferable in practice due to their shorter lengths for the main parameters such as group public keys, individual secret signing keys, and output group signatures.

Due to its support for dynamic groups, the verifiable opening property, and the slightly better performance the BCNSW-VLR scheme should probably be preferred over the BS scheme. With regard to security we observe that both schemes offer insider anonymity; while this is not the strongest anonymity notion, it is sufficient for most applications. Moreover, an individual secret signing key $\boldsymbol{gsk}[i]$ in the BCNSW-VLR scheme is about 1067 bits and could potentially be stored in a tamper-resistant cryptographic hardware chip, which would significantly minimize the possibility of its leakage. Both schemes offer full non-frameability — however, in the (static) BS scheme the group manager's secret key must be erased after the generation of individual secret keys. If the group manager is malicious and doesn't erase its key then framing attacks against BS users become possible. On the other hand, potential advantage of the BS scheme is that it offers full traceability, has a somewhat simpler design in comparison to the BCNSW-VLR scheme, and that it can be used in a distributed mode with two distinct authorities. Therefore, this scheme is preferable for applications where the management of the group must be distributed amongst two non-colluding authorities.

Bibliography

[1] M. Abdalla, J. H. An, M. Bellare, and C. Namprempre, *From Identification to Signatures via the Fiat-Shamir Transform: Minimizing Assumptions for Security and Forward-Security*, in EUROCRYPT 2002, vol. 2332 of LNCS, Springer, 2002, pp. 418–433. 23

[2] M. Abdalla and B. Warinschi, *On the Minimal Assumptions of Group Signature Schemes*, in ICICS 2004, vol. 3269 of LNCS, Springer, 2004, pp. 1–13. 97

[3] M. Abe, *A Secure Three-Move Blind Signature Scheme for Polynomially Many Signatures*, in EUROCRYPT 2001, vol. 2045 of LNCS, Springer, 2001, pp. 136–151. 40

[4] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo, *Structure-Preserving Signatures and Commitments to Group Elements*, in CRYPTO 2010, vol. 6223 of LNCS, Springer, 2010, pp. 209–236. 40

[5] M. Abe and M. Ohkubo, *A Framework for Universally Composable Non-committing Blind Signatures*, in ASIACRYPT 2009, vol. 5912 of LNCS, Springer, 2009, pp. 435–450. 40

[6] M. Abe, M. Ohkubo, and K. Suzuki, *1-out-of-n Signatures from a Variety of Keys*, in ASIACRYPT 2002, vol. 2501 of LNCS, Springer, 2002, pp. 639–645. 41

[7] T. Acar, K. Lauter, M. Naehrig, and D. Shumow, *Affine Pairings on ARM*. Cryptology ePrint Archive, Report 2011/243, 2011. http://eprint.iacr.org/2011/243. 204, 205

[8] M. Ajtai, *Generating Hard Instances of Lattice Problems (Extended Abstract)*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC 1996), ACM, 1996, pp. 99–108. 36

[9] *Android NDK*. Available at http://developer.android.com/sdk/ndk/. 198

[10] ANSSI, *Mecanismes cryptographiques - Regles et recommandations*, 2010. Available at http://www.ssi.gouv.fr/IMG/pdf/RGS_B_1.pdf. 194

[11] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik, *A Practical and Provably Secure Coalition-Resistant Group Signature Scheme*, in CRYPTO 2000, vol. 1880 of LNCS, Springer, 2000, pp. 255–270. 31, 101, 102, 105, 113

[12] G. Ateniese and B. de Medeiros, *Efficient Group Signatures without Trapdoors*, in ASIACRYPT 2003, vol. 2894 of LNCS, Springer, 2003, pp. 246–268. 131, 133, 135, 137, 172, 175, 180, 181, 185

[13] ——, *A Provably Secure Nyberg-Rueppel Signature Variant with Applications*. Cryptology ePrint Archive, Report 2004/093, 2004. http://eprint.iacr.org/2004/093. 135, 139

[14] G. Ateniese, J. Kirsch, and M. Blanton, *Secret Handshakes with Dynamic and Fuzzy Matching*, in Network and Distributed System Security Symposium (NDSS 2007), The Internet Society, 2007. 39

[15] G. Ateniese, D. X. Song, and G. Tsudik, *Quasi-Efficient Revocation in Group Signatures*, in International Conference on Financial Cryptography and Data Security (FC 2002), vol. 2357 of LNCS, Springer, 2002, pp. 183–197. 101, 106

[16] G. Ateniese and G. Tsudik, *Group Signatures á la carte*, in ACM-SIAM Symposium on Discrete Algorithms (SODA 1999), ACM, 1999, pp. 848–849. 101, 102, 172, 175, 180, 181, 185

[17] ——, *Some Open Issues and New Directions in Group Signatures*, in International Conference on Financial Cryptography and Data Security (FC 1999), vol. 1648 of LNCS, Springer, 1999, pp. 196–211. 31, 102

[18] D. Balfanz, G. Durfee, N. Shankar, D. K. Smetters, J. Staddon, and H.-C. Wong, *Secret Handshakes from Pairing-Based Key Agreements*, in IEEE Symposium on Security and Privacy 2003, IEEE CS, 2003, pp. 180–196. 39

[19] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, *Elementary Gates for Quantum Computation*, Phys. Rev. A, 52 (1995), pp. 3457–3467. 36

[20] N. Barić and B. Pfitzmann, *Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees*, in EUROCRYPT 1997, vol. 1233 of LNCS, Springer, 1997, pp. 480–494. 107

[21] M. Bellare and S. Duan, *Partial Signatures and their Applications*. Cryptology ePrint Archive, Report 2009/336, 2009. http://eprint.iacr.org/2009/336. 37

[22] M. Bellare, D. Micciancio, and B. Warinschi, *Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions*, in EUROCRYPT 2003, vol. 2656 of LNCS, Springer, 2003, pp. 614–629. 32, 34, 45, 47, 48, 51, 93, 94, 96, 101, 121, 144, 148

[23] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, *The one-more-rsa-inversion problems and the security of chaum's blind signature scheme*, J. Cryptology, 16 (2003), pp. 185–215. 40

[24] M. Bellare and P. Rogaway, *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*, in ACM Conference on Computer and Communications Security (ACM CCS 1993), ACM, 1993, pp. 62–73. 83

[25] M. Bellare, H. Shi, and C. Zhang, *Foundations of Group Signatures: The Case of Dynamic Groups*, in CT-RSA 2005, vol. 3376 of LNCS, Springer, 2005, pp. 136–153. 73, 93, 95, 96, 97, 99, 101, 141

[26] J. C. Benaloh and M. de Mare, *One-Way Accumulators: A Decentralized Alternative to Digital Sinatures (Extended Abstract)*, in EUROCRYPT 1993, vol. 765 of LNCS, Springer, 1993, pp. 274–285. 107

[27] A. Bender, J. Katz, and R. Morselli, *Ring Signatures: Stronger Definitions, and Constructions Without Random Oracles*, in TCC 2006, vol. 3876 of LNCS, Springer, 2006, pp. 60–79. 41

[28] V. Benjumea, S. Choi, J. Lopez, and M. Yung, *Fair Traceable Multi-Group Signatures*, in Financial Cryptography and Data Security (FC 08), vol. 5143 of LNCS, Springer, 2008, pp. 231–246. 42

[29] D. J. Bernstein, J. Buchmann, and E. Dahmen, *Post-Quantum Cryptography*, Springer, 2009. 36

[30] P. Bichsel, J. Camenisch, T. Gross, and V. Shoup, *Anonymous Credentials on a Standard Java Card*, in ACM Conference on Computer and Communications Security (ACM CCS 2009), ACM, 2009, pp. 600–610. 38

[31] P. Bichsel, J. Camenisch, G. Neven, N. P. Smart, and B. Warinschi, *Get Shorty via Group Signatures without Encryption*, in International Conference on Security and Cryptography for Networks (SCN 2010), vol. 6280 of LNCS, Springer, 2010, pp. 381–398. 37, 58, 83, 141, 149, 152, 155, 168, 170, 172, 175, 180, 181, 185, 191, 192, 193, 196, 197, 198, 201, 227, 239

[32] BNetzA and BSI, *Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung*, Preprint at http://www.bundesnetzagentur.de/media/, (2011). 194, 195, 207, 208

[33] A. Boldyreva, *Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme*, in Public Key Cryptography (PKC 2003), vol. 2567 of LNCS, Springer, 2003, pp. 31–46. 40

[34] D. Boneh and X. Boyen, *Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles*, in EUROCRYPT 2004, vol. 3027 of LNCS, Springer, 2004, pp. 223–238. 82

[35] ——, *Short Signatures Without Random Oracles*, in EUROCRYPT 2004, vol. 3027 of LNCS, Springer, 2004, pp. 56–73. 81

[36] D. Boneh, X. Boyen, and H. Shacham, *Short Group Signatures*, in CRYPTO 2004, vol. 3152 of LNCS, Springer, 2004, pp. 41–55. 49, 82, 141, 142, 144, 145, 172, 175, 180, 181, 185

[37] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, *Aggregate and Verifiably Encrypted Signatures from Bilinear Maps*, in EUROCRYPT 2003, vol. 2656 of LNCS, Springer, 2003, pp. 416–432. 41

[38] D. Boneh and H. Shacham, *Group Signatures with Verifier-Local Revocation*, in ACM Conference on Computer and Communications Security (ACM CCS 2004), ACM, 2004, pp. 168–177. 48, 155, 159, 161, 162, 163, 164, 165, 167, 172, 175, 180, 181, 185, 191, 192, 193, 196, 197, 198, 201, 219, 239

[39] X. Boyen and B. Waters, *Compact Group Signatures Without Random Oracles*, in EUROCRYPT 2006, vol. 4004 of LNCS, Springer, 2006, pp. 427–444. 141

[40] ——, *Full-Domain Subgroup Hiding and Constant-Size Group Signatures*, in Public Key Cryptography (PKC 2007), vol. 4450 of LNCS, Springer, 2007, pp. 1–15. 141, 165

[41] R. Bradshaw, J. Holt, and K. Seamons, *Concealing Complex Policies with Hidden Credentials*, in ACM Conference on Computer and Communications Security (CCS 2004), ACM, 2004, pp. 146–157. 39

[42] E. Bresson and J. Stern, *Efficient Revocation in Group Signatures*, in Public Key Cryptography (PKC 2001), vol. 1992 of LNCS, Springer, 2001, pp. 190–206. 41, 101, 106

[43] E. Brickell, L. Chen, and J. Li, *A New Direct Anonymous Attestation Scheme from Bilinear Maps*, in International Conference on Trusted Computing - Challenges and Applications (TRUST 2008), vol. 4968 of LNCS, Springer, 2008, pp. 166–178. 41

[44] E. F. Brickell, J. Camenisch, and L. Chen, *Direct Anonymous Attestation*, in ACM Conference on Computer and Communications Security (ACM CCS 2004), ACM, 2004, pp. 132–145. 23, 41

[45] J. Camenisch, *Efficient and Generalized Group Signatures*, in EUROCRYPT 1997, vol. 1233 of LNCS, Springer, 1997, pp. 465–479. 92

[46] ——, *Efficient Anonymous Fingerprinting with Group Signatures*, in ASIACRYPT 2000, vol. 1976 of LNCS, Springer, 2000, pp. 415–428. 23

[47] ——, *Anonymous Credentials: Opportunities and Challenges*, in IFIP TC-11 21st International Information Security Conference (SEC 2006), vol. 201 of IFIP, Springer, 2006, p. 460. 38

[48] J. Camenisch and T. Gross, *Efficient Attributes for Anonymous Credentials*, in ACM Conference on Computer and Communications Security, ACM, 2008, pp. 345–356. 38

[49] J. Camenisch and J. Groth, *Group Signatures: Better Efficiency and New Theoretical Aspects*, in International Conference on Security in Communication Networks (SCN 2004), vol. 3352 of LNCS, Springer, 2004, pp. 120–133. 101, 118, 119, 122, 172, 175, 180, 181, 185, 191, 192, 193, 195, 198, 201, 207, 210, 239

[50] J. Camenisch and E. V. Herreweghen, *Design and Implementation of the Idemix Anonymous Credential System*, in ACM Conference on Computer and Communications Security (ACM CCS 2002), ACM, 2002, pp. 21–30. 23, 38

[51] J. Camenisch, M. Kohlweiss, and C. Soriente, *An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials*, in International Conference on Practice and Theory in Public Key Cryptography (PKC 2009), vol. 5443 of LNCS, Springer, 2009, pp. 481–500. 38

[52] ——, *Solving Revocation with Efficient Update of Anonymous Credentials*, in International Conference of Security and Cryptography in Networks (SCN 2010), vol. 6280 of LNCS, Springer, 2010, pp. 454–471. 38

[53] J. Camenisch, M. Koprowski, and B. Warinschi, *Efficient Blind Signatures Without Random Oracles*, in SCN 2004, vol. 3352 of LNCS, Springer, 2004, pp. 134–148. 40

[54] J. Camenisch and A. Lysyanskaya, *An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation*, in EUROCRYPT 2001, vol. 2045 of LNCS, Springer, 2001, pp. 93–118. 38

[55] ——, *A Signature Scheme with Efficient Protocols*, in International Conference on Security in Communication Networks (SCN 2002), vol. 2576 of LNCS, Springer, 2002, pp. 268–289. 119, 120, 207, 209

[56] ——, *Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials*, in CRYPTO 2002, vol. 2442 of LNCS, Springer, 2002, pp. 61–76. 38, 101, 106, 107, 109, 110, 113, 122

[57] ——, *Signature Schemes and Anonymous Credentials from Bilinear Maps*, in CRYPTO 2004, vol. 3152 of LNCS, Springer, 2004, pp. 56–72. 38, 141, 145, 146, 147, 148, 149, 169, 172, 175, 180, 181, 185, 227

[58] J. Camenisch and M. Michels, *A Group Signature Scheme with Improved Efficiency*, in ASIACRYPT 1998, vol. 1514 of LNCS, Springer, 1998, pp. 160–174. 101, 102

[59] ——, *Proving in Zero-Knowledge that a Number Is the Product of Two Safe Primes*, in EUROCRYPT 1999, vol. 1592 of LNCS, Springer, 1999, pp. 107–122. 103, 113

[60] ——, *Separability and Efficiency for Generic Group Signature Schemes*, in CRYPTO 1999, vol. 1666 of LNCS, Springer, 1999, pp. 413–430. 93

[61] J. Camenisch and M. Stadler, *Efficient Group Signature Schemes for Large Groups*, in CRYPTO 1997, vol. 1294 of LNCS, Springer, 1997, pp. 410–424. 101, 102, 106

[62] S. Canard, B. Schoenmakers, M. Stam, and J. Traoré, *List Signature Schemes*, Discrete Applied Mathematics, 154 (2006), pp. 189–201. 23

[63] S. Canard and J. Traoré, *On Fair E-cash Systems Based on Group Signature Schemes*, in Australasian Conference on Information Security and Privacy (ACISP 2003), vol. 2727 of LNCS, Springer, 2003, pp. 237–248. 23

[64] C. Castelluccia, S. Jarecki, and G. Tsudik, *Secret Handshakes from CA-Oblivious Encryption*, in ASIACRYPT 2004, vol. 3329 of LNCS, Springer, 2004, pp. 293–307. 39

[65] D. Catalano, R. Gennaro, N. Howgrave-Graham, and P. Q. Nguyen, *Paillier's Cryptosystem Revisited*, in ACM Conference on Computer and Communications Security (ACM CCS 2001), ACM, 2001, pp. 206–214. 115, 117, 118, 176

[66] N. Chandran, J. Groth, and A. Sahai, *Ring Signatures of Sub-linear Size Without Random Oracles*, in ICALP 2007, vol. 4596 of LNCS, Springer, 2007, pp. 423–434. 41, 42

[67] M. Chase and A. Lysyanskaya, *On Signatures of Knowledge*, in CRYPTO 2006, vol. 4117 of LNCS, Springer, 2006, pp. 78–96. 92

[68] D. Chaum, *Blind Signatures for Untraceable Payments*, in CRYPTO 1982, 1982, pp. 199–203. 40

[69] ——, *Security Without Identification: Transaction Systems to Make Big Brother Obsolete*, Commununications of the ACM, 28 (1985), pp. 1030–1044. 38

[70] D. Chaum and J.-H. Evertse, *A secure and privacy-protecting protocol for transmitting personal information between organizations*, in CRYPTO 1986, vol. 263 of LNCS, Springer, 1986, pp. 118–167. 38

[71] D. Chaum and E. van Heyst, *Group Signatures*, in EUROCRYPT 1991, vol. 547 of LNCS, Springer, 1991, pp. 257–265. 21, 30, 31, 32, 101

[72] L. Chen, *A DAA Scheme Requiring Less TPM Resources*, in International Conference on Information Security and Cryptology (Inscrypt 2009), vol. 6151 of LNCS, Springer, 2009, pp. 350–365. 41

[73] L. Chen and T. P. Pedersen, *New Group Signature Schemes*, in EUROCRYPT 1994, vol. 950 of LNCS, Springer, 1994, pp. 171–181. 32, 131

[74] S. Choi, K. Park, and M. Yung, *Short Traceable Signatures Based on Bilinear Pairings*, in IWSEC 2006, vol. 4266 of LNCS, Springer, 2006, pp. 88–103. 42

[75] S. Chow, *Real Traceable Signatures*, in Selected Areas in Cryptography (SAC 2009), vol. 5867 of LNCS, Springer, 2009, pp. 92–107. 42

[76] S. S. M. Chow, V. K. Wei, J. K. Liu, and T. H. Yuen, *Ring signatures without random oracles*, in ACM Symposium on Information, Computer and Communications Security (ASIACCS 2006), ACM, 2006, pp. 297–302. 41

[77] R. Cramer and V. Shoup, *A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack*, in CRYPTO 1998, vol. 1462 of LNCS, Springer, 1998, pp. 13–25. 141, 146, 147, 148

[78] L. Dallot and D. Vergnaud, *Provably secure code-based threshold ring signatures*, in Cryptography and Coding, vol. 5921 of LNCS, Springer, 2009, pp. 222–235. 41

[79] I. Damgård, *Payment Systems and Credential Mechanisms with Provable Security Against Abuse by Individuals*, in CRYPTO 1988, vol. 403 of LNCS, Springer, 1988, pp. 328–335. 38

[80] I. Damgård and E. Fujisaki, *A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order*, in ASIACRYPT 2002, vol. 2501 of LNCS, Springer, 2002, pp. 125–142. 115, 117

[81] C. Delerablée and D. Pointcheval, *Dynamic Fully Anonymous Short Group Signatures*, in VIETCRYPT 2006, vol. 4341 of LNCS, Springer, 2006, pp. 193–210. 141

[82] D. Deutsch, *Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer*, Proc. of the Royal Society of London. A. Mathematical and Physical Sciences, 400 (1985), pp. 97–117. 36

[83] X. Ding, G. Tsudik, and S. Xu, *Leak-Free Group Signatures with Immediate Revocation*, in International Conference on Distributed Computing Systems (ICDCS 2004), IEEE CS, 2004, pp. 608–615. 30, 43

[84] ——, *Leak-Free Mediated Group Signatures*, Journal of Computer Security, 17 (2009), pp. 489–514. 30, 43

[85] Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup, *Anonymous Identification in Ad Hoc Groups*, in EUROCRYPT 2004, vol. 3027 of LNCS, Springer, 2004, pp. 609–626. 41

[86] S. Duquesne and N. Guillermin, *A FPGA pairing implementation using the Residue Number System*. Cryptology ePrint Archive, Report 2011/176, 2011. http://eprint.iacr.org/2011/176. 204, 205

[87] ECRYPT II, *Yearly Report on Algorithms and Keysizes (2009-2010)*. Available at http://www.ecrypt.eu.org/. 194, 195, 207, 208

[88] T. El Gamal, *A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms*, IEEE Transactions on Information Theory, 31 (1985), pp. 469–472. 80, 104

[89] A. Fiat and A. Shamir, *How to Prove Yourself: Practical Solutions to Identification and Signature Problems*, in CRYPTO 1986, vol. 263 of LNCS, Springer, 1986, pp. 186–194. 23, 90, 92

[90] M. Fischlin, *Round-Optimal Composable Blind Signatures in the Common Reference String Model*, in CRYPTO 2006, vol. 4117 of LNCS, Springer, 2006, pp. 60–77. 40

[91] ——, *Anonymous Signatures Made Easy*, in Public Key Cryptography (PKC 2007), vol. 3958 of LNCS, 2007, pp. 31–42. 37

[92] D. Freeman, M. Scott, and E. Teske, *A Taxonomy of Pairing-Friendly Elliptic Curves*, Journal of Cryptology, 23 (2010), pp. 224–280. 194, 196, 197

[93] J. Furukawa and H. Imai, *An Efficient Group Signature Scheme from Bilinear Maps*, in Australasian Conference on Information Security and Privacy (ACISP 2005), vol. 3574 of LNCS, Springer, 2005, pp. 455–467. 141

[94] J. Furukawa and S. Yonezawa, *Group Signatures with Separate and Distributed Authorities*, in International Conference on Security in Communication Networks (SCN 2004), vol. 3352 of LNCS, Springer, 2004, pp. 77–90. 131, 136, 138, 140, 172, 175, 180, 181, 185

[95] S. D. Galbraith, K. G. Paterson, and N. P. Smart, *Pairings for Cryptographers*, Discrete Applied Mathematics, 156 (2008), pp. 3113–3121. 81

[96] D. Galindo, B. Libert, M. Fischlin, G. Fuchsbauer, A. Lehmann, M. Manulis, and D. Schröder, *Public-Key Encryption with Non-interactive Opening: New Constructions and Stronger Definitions*, in AFRICACRYPT 2010, vol. 6055 of LNCS, Springer, May 2010, pp. 333–350. 96

[97] C.-z. Gao, Z.-a. Yao, and L. Li, *A Ring Signature Scheme Based on the Nyberg-Rueppel Signature Scheme*, in ACNS 2003, vol. 2846 of LNCS, Springer, 2003, pp. 169–175. 41

[98] S. Garg, V. Rao, A. Sahai, D. Schröder, and D. Unruh, *Round Optimal Blind Signatures*, in CRYPTO 2011, vol. 6841 of Lecture Notes in Computer Science, Springer, 2011, pp. 630–648. 40

[99] *GCC, the GNU Compiler Collection.* Available at http://gcc.gnu.org/. 197, 198

[100] H. Ge and S. Tate, *Traceable Signature: Better Efficiency and Beyond*, in Computational Science and Its Applications (ICCSA 2006), vol. 3982 of LNCS, Springer, 2006, pp. 327–337. 42

[101] *GMP — The GNU Multiple Precision Arithmetic Library.* Available at http://gmplib.org/. 198, 203

[102] D. M. Gordon, *A Survey of Fast Exponentiation Methods*, Journal of Algorithms, 27 (1998), pp. 129–146. 202, 204

[103] S. D. Gordon, J. Katz, and V. Vaikuntanathan, *A Group Signature Scheme from Lattice Assumptions*, in ASIACRYPT 2010, vol. 6477 of LNCS, Springer, 2010, pp. 395–412. 36

[104] R. Granger, D. Page, and N. P. Smart, *High Security Pairing-Based Cryptography Revisited*, in 7th International Symposium on Algorithmic Number Theory (ANTS-VII), vol. 4076 of LNCS, Springer, 2006, pp. 480–494. 194

[105] J. Groth, *Fully Anonymous Group Signatures Without Random Oracles*, in ASIACRYPT 2007, vol. 4833 of LNCS, Springer, 2007, pp. 164–180. 141

[106] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer, 2004. 204

[107] C. Hazay, J. Katz, C.-Y. Koo, and Y. Lindell, *Concurrently-Secure Blind Signatures Without Random Oracles or Setup Assumptions*, in Theory of Cryptography Conference (TCC 2007), vol. 4392 of LNCS, Springer, 2007, pp. 323–341. 40

[108] J. Herranz and G. Sáez, *Forking Lemmas for Ring Signature Schemes*, in INDOCRYPT 2003, vol. 2904 of LNCS, Springer, 2003, pp. 266–279. 41

[109] S. Jarecki, J. Kim, and G. Tsudik, *Group Secret Handshakes or Affiliation-Hiding Authenticated Group Key Agreement*, in CT-RSA 2007, vol. 4377 of LNCS, Springer, 2007, pp. 287–308. 39

[110] ——, *Beyond Secret Handshakes: Affiliation-Hiding Authenticated Key Exchange*, in CT-RSA 2008, vol. 4964 of LNCS, Springer, 2008, pp. 352–369. 39

[111] S. Jarecki and X. Liu, *Unlinkable Secret Handshakes and Key-Private Group Key Management Schemes*, in 5th Intl. Conference on Applied Cryptography and Network Security (ACNS 2007), vol. 4521 of LNCS, Springer, 2007, pp. 270–287. 39

[112] ——, *Affiliation-Hiding Envelope and Authentication Schemes with Efficient Support for Multiple Credentials*, in ICALP (2), vol. 5126 of LNCS, Springer, 2008, pp. 715–726. 39

[113] ——, *Private Mutual Authentication and Conditional Oblivious Transfer*, in CRYPTO 2009, vol. 5677 of LNCS, Springer, 2009, pp. 90–107. 39

[114] A. Juels, M. Luby, and R. Ostrovsky, *Security of Blind Digital Signatures*, in CRYPTO 1997, vol. 1294 of LNCS, Springer, 1997, pp. 150–164. 40

[115] J. Katz, *Digital Signatures*, Springer, 2010. 84

[116] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, Chapman & Hall/CRC, 2007. 84

[117] A. Kiayias, Y. Tsiounis, and M. Yung, *Traceable Signatures*, in EUROCRYPT 2004, vol. 3027 of LNCS, Springer, 2004, pp. 571–589. 42, 43

[118] A. Kiayias and M. Yung, *Extracting Group Signatures from Traitor Tracing Schemes*, in EUROCRYPT 2003, vol. 2656 of LNCS, Springer, 2003, pp. 630–648. 23

[119] ——, *Efficient Secure Group Signatures with Dynamic Joins and Keeping Anonymity Against Group Managers*, in Mycrypt 2005, vol. 3715 of LNCS, Springer, 2005, pp. 151–170. 101, 125, 126, 172, 175, 180, 181, 185

[120] ——, *Group Signatures with Efficient Concurrent Join*, in EUROCRYPT 2005, vol. 3494 of LNCS, Springer, 2005, pp. 198–214. 141

[121] ——, *Secure Scalable Group Signature with Dynamic Joins and Separable Authorities*, IJSN, 1 (2006), pp. 24–45. 101, 125, 126

[122] A. Kiayias and H.-S. Zhou, *Concurrent Blind Signatures Without Random Oracles*, in International Conference on Security and Cryptography for Networks (SCN 2006), vol. 4116 of LNCS, Springer, 2006, pp. 49–62. 40

[123] ——, *Equivocal Blind Signatures and Adaptive UC-Security*, in Theory of Cryptography Conference (TCC 2008), vol. 4948 of LNCS, Springer, 2008, pp. 340–355. 40

[124] J. Kilian and E. Petrank, *Identity Escrow*, in CRYPTO 1998, vol. 1462 of LNCS, Springer, 1998, pp. 169–185. 22

[125] K.-C. Lee, H.-A. Wen, and T. Hwang, *Convertible ring signature*, IEE Proceedings Communications, 152 (2005), pp. 411–414. 41

[126] M.-F. Lee, N. P. Smart, and B. Warinschi, *The Fiat-Shamir Transform for Group and Ring Signature Schemes*, in International Conference on Security and Cryptography for Networks (SCN 2010), vol. 6280 of LNCS, Springer, 2010, pp. 363–380. 22, 23

[127] A. K. Lenstra, *Key Lengths*, in Handbook of Information Security, H. Bidgoli, ed., Wiley, 2004, pp. 617–635. 194

[128] A. K. Lenstra and E. R. Verheul, *Selecting Cryptographic Key Sizes*, Journal of Cryptology, 14 (2001), pp. 255–293. 194

[129] A. Leung, L. Chen, and C. J. Mitchell, *On a Possible Privacy Flaw in Direct Anonymous Attestation (DAA)*, in International Conference on Trusted Computing - Challenges and Applications (TRUST 2008), vol. 4968 of LNCS, Springer, 2008, pp. 179–190. 41

[130] X. Li, D. Zheng, K. Chen, and J. Li, *Democratic Group Signatures with Collective Traceability*, Computers & Electrical Engineering, 35 (2009), pp. 664–672. 29

[131] B. Libert and D. Vergnaud, *Group Signatures with Verifier-Local Revocation and Backward Unlinkability in the Standard Model*, in International Conference on Cryptology and Network Security (CANS 2009), vol. 5888 of LNCS, Springer, 2009, pp. 498–517. 155, 165

[132] J. Liu, V. Wei, and D. Wong, *Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups*, in Australasian Conference on Information Security and Privacy (ACISP 2004), vol. 3108 of LNCS, Springer, 2004, pp. 325–335. 41

[133] A. Lysyanskaya, *Signature Schemes and Applications to Cryptographic Protocol Design*, PhD thesis, Massachusetts Institute of Technology, 2002. AAI0804606. 119, 120, 207

[134] A. Lysyanskaya and Z. Ramzan, *Group Blind Digital Signatures: A Scalable Solution to Electronic Cash*, in Financial Cryptography and Data Security (FC 1998), vol. 1465 of LNCS, Springer, 1998, pp. 184–197. 29, 40

[135] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf, *Pseudonym Systems*, in Selected Areas in Cryptography (SAC 1999), LNCS, Springer, 1999, pp. 184–199. 38, 82

[136] G. Maitland and C. Boyd, *Fair Electronic Cash Based on a Group Signature Scheme*, in International Conference on Information and Communications Security (ICICS 2001), vol. 2229 of LNCS, Springer, 2001, pp. 461–465. 23

[137] M. Manulis, *Democratic Group Signatures: On an Example of Joint Ventures*, in ACM Symposium on Information, Computer and Communications Security, (ASIACCS 2006), ACM, 2006, p. 365. Available at http://eprint.iacr.org/2005/446.pdf. 29

[138] M. Manulis, B. Pinkas, and B. Poettering, *Privacy-Preserving Group Discovery with Linear Complexity*, in International Conference on Applied Cryptography and Network Security (ACNS 2010), vol. 6123 of LNCS, Springer, 2010, pp. 420–437. 39

[139] M. Manulis and B. Poettering, *Affiliation-Hiding Authentication with Minimal Bandwidth Consumption*, in IFIP WG 11.2 International Workshop on Information Security Theory and Practice (WISTP 2011), vol. 6633 of LNCS, Springer, 2011, pp. 85–99. 39

[140] ——, *Practical Affiliation-Hiding Authentication from Improved Polynomial Interpolation*, in ACM Symposium on Information, Computer and Communications Security (ASIACCS 2011), ACM, 2011, pp. 286–295. 39

[141] M. Manulis, B. Poettering, and G. Tsudik, *Affiliation-Hiding Key Exchange with Untrusted Group Authorities*, in International Conference on Applied Cryptography and Network Security (ACNS 2010), vol. 6123 of LNCS, Springer, 2010, pp. 402–419. 39

[142] ——, *Taming Big Brother Ambitions: More Privacy for Secret Handshakes*, in 10th Privacy Enhancing Technologies Symposium (PETS 2010), vol. 6205 of LNCS, Springer, 2010, pp. 149–165. 39

[143] M. Manulis, A.-R. Sadeghi, and J. Schwenk, *Linkable Democratic Group Signatures*, in 2nd Information Security Practice and Experience Conference (ISPEC 2006), vol. 3903 of LNCS, Springer, 2006, pp. 187–201. 29

[144] A. Miyaji, M. Nakabayashi, and S. Takano, *New Explicit Conditions of Elliptic Curve Traces for FR-Reduction*, IEICE TRANSACTIONS on Fundamentals A, E84-A (2001), pp. 1234–1243. 203

[145] M. Naehrig, R. Niederhagen, and P. Schwabe, *New Software Speed Records for Cryptographic Pairings*, in LATINCRYPT, 2010, pp. 109–123. 205, 223, 224, 234, 235, 240, 241, 243, 245, 246

[146] T. Nakanishi and N. Funabiki, *Verifier-Local Revocation Group Signature Schemes with Backward Unlinkability from Bilinear Maps*, in ASIACRYPT 2005, vol. 3788 of LNCS, Springer, 2005, pp. 533–548. 155, 157, 161, 165, 166, 167, 168, 172, 175, 180, 181, 185

[147] ——, *A Short Verifier-Local Revocation Group Signature Scheme with Backward Unlinkability*, in International Workshop on Security (IWSEC 2006), vol. 4266 of LNCS, Springer, 2006, pp. 17–32. 155

[148] T. Nakanishi, F. Kubooka, N. Hamada, and N. Funabiki, *Group Signature Schemes with Membership Revocation for Large Groups*, in Australasian Conference on Information Security and Privacy (ACISP 2005), vol. 3574 of LNCS, Springer, 2005, pp. 443–454. 101

[149] T. Nakanishi and Y. Sugiyama, *A Group Signature Scheme with Efficient Membership Revocation for Reasonable Groups*, in Australasian Conference on Information Security and Privacy (ACISP 2004), vol. 3108 of LNCS, Springer, 2004, pp. 336–347. 101

[150] K. Q. Nguyen and J. Traoré, *An Online Public Auction Protocol Protecting Bidder Privacy*, in Australasian Conference on Information Security and Privacy (ACISP 2000), vol. 1841 of LNCS, Springer, 2000, pp. 427–442. 23

[151] L. Nguyen and R. Safavi-Naini, *Efficient and Provably Secure Trapdoor-Free Group Signature Schemes from Bilinear Pairings*, in ASIACRYPT 2004, vol. 3329 of LNCS, Springer, 2004, pp. 372–386. 141

[152] NIST, *Recommendation for Key Management*. Available at http://csrc.nist.gov/groups/ST/toolkit/key_management.html. 194

[153] NSA, *Fact Sheet Suite B Cryptography*, 2010. Available at http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml. 194

[154] K. Nyberg and R. Rueppel, *Message Recovery for Signature Schemes Based on the Discrete Logarithm Problem*, in EUROCRYPT 1994, vol. 950 of LNCS, Springer, 1995, pp. 182–193. 133, 137

[155] T. Okamoto, *Efficient Blind and Partially Blind Signatures Without Random Oracles*, in Theory of Cryptography Conference (TCC 2006), vol. 3876 of LNCS, Springer, 2006, pp. 80–99. 40

[156] P. Paillier, *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*, in EUROCRYPT 1999, vol. 1592 of LNCS, Springer, 1999, pp. 223–238. 115

[157] *PBC — The Pairing-Based Cryptography Library*. Available at http://crypto.stanford.edu/pbc/. 198, 203

[158] T. P. Pedersen, *A Threshold Cryptosystem without a Trusted Party.*, in EUROCRYPT 1991, vol. 547 of LNCS, Springer, 1991, pp. 522–526. 140

[159] ——, *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*, in CRYPTO 1991, vol. 576 of LNCS, Springer, 1991, pp. 129–140. 86, 104, 137, 139

[160] H. Petersen, *How to Convert any Digital Signature Scheme into a Group Signature Scheme*, in Security Protocols Workshop 1997, vol. 1361 of LNCS, Springer, 1997, pp. 177–190. 131

[161] D. Pointcheval and J. Stern, *Security Arguments for Digital Signatures and Blind Signatures*, Journal of Cryptology, 13 (2000), pp. 361–396. 40

[162] R. Rivest, A. Shamir, and Y. Tauman, *How to Leak a Secret*, in ASIACRYPT 2001, vol. 2248 of LNCS, Springer, 2001, pp. 552–565. 41

[163] J. Rompel, *One-Way Functions are Necessary and Sufficient for Secure Signatures*, in 22nd Annual ACM Symposium on Theory of Computing (STOC 1990), ACM, 1990, pp. 387–394. 97

[164] M. Rückert, *Lattice-Based Blind Signatures*, in ASIACRYPT 2010, vol. 6477 of LNCS, Springer, 2010, pp. 413–430. 40

[165] A. Sahai, *Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security*, in 40th Annual Symposium on Foundations of Computer Science (FOCS 1999), IEEE, 1999, pp. 543–553. 96, 100

[166] V. Saraswat and A. Yun, *Anonymous Signatures Revisited*, in International Conference on Provable Security (ProvSec 2009), vol. 5848 of LNCS, Springer, 2009, pp. 140–153. 37

[167] C. Schnorr and M. Jakobsson, *Security of Signed ElGamal Encryption*, in ASIACRYPT 2000, vol. 1976 of LNCS, Springer, 2000, pp. 73–89. 139

[168] M. Scott, N. Costigan, and W. Abdulwahab, *Implementing Cryptographic Pairings on Smartcards*, in CHES, 2006, pp. 134–147. 204, 205

[169] P. W. Shor, *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM Journal on Computing, 26 (1997), pp. 1484–1509. 36

[170] V. Shoup, *Lower Bounds for Discrete Logarithms and Related Problems*, in EUROCRYPT 1997, vol. 1233 of LNCS, Springer, 1997, pp. 256–266. 138, 139

[171] V. Shoup, *A Computational Introduction to Number Theory and Algebra*, Cambridge University Press, 2008. Available at http://www.shoup.net/ntb/. 79, 109, 114

[172] R. D. Silverman, *Exposing the Mythical MIPS Year*, IEEE Computer, 32 (1999). 194

[173] B. Smyth, M. Ryan, and L. Chen, *Direct Anonymous Attestation (DAA): Ensuring Privacy with Corrupt Administrators*, in 4th European Workshop Security and Privacy in Ad-hoc and Sensor Networks (ESAS 2007), vol. 4572 of LNCS, Springer, 2007, pp. 218–231. 41

[174] J. Traoré, *Group Signatures and Their Relevance to Privacy-Protecting Off-Line Electronic Cash Systems*, in Australasian Conference on Information Security and Privacy (ACISP 1999), vol. 1587 of LNCS, Springer, 1999, pp. 228–243. 23

[175] Trusted Computing Group, *Trusted Computing Platform Alliance (TCPA) main specification*. http://www.trustedcomputinggroup.org. 40

[176] P. Tsang and V. Wei, *Short Linkable Ring Signatures for E-Voting, E-Cash and Attestation*, in ISPEC 2005, vol. 3439 of LNCS, Springer, 2005, pp. 48–60. 41

[177] P. Tsang, V. Wei, T. Chan, M. Au, J. Liu, and D. Wong, *Separable Linkable Threshold Ring Signatures*, in INDOCRYPT 2004, vol. 3348 of LNCS, Springer, 2005, pp. 337–376. 41

[178] G. Tsudik and S. Xu, *Accumulating Composites and Improved Group Signing*, in ASIACRYPT 2003, vol. 2894 of LNCS, Springer, 2003, pp. 269–286. 101, 111, 112, 113, 116, 117, 172, 175, 180, 181, 185

[179] ——, *A Flexible Framework for Secret Handshakes*, in 6th Intl. Workshop on Privacy Enhancing Technologies (PET 2006), vol. 4258 of LNCS, Springer, 2006, pp. 295–315. 39

[180] D. Vergnaud, *RSA-Based Secret Handshakes*, in Intl. Workshop on Coding and Cryptography (WCC 2005), vol. 3969 of LNCS, Springer, 2005, pp. 252–274. 39

[181] L. von Ahn, A. Bortz, N. Hopper, and K. O'Neill, *Selectively traceable anonymity*, in Privacy Enhancing Technologies, vol. 4258 of LNCS, Springer, 2006, pp. 208–222. 42

[182] V. Wei, *Tracing-by-Linking Group Signatures*, in Information Security, vol. 3650 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2005, pp. 149–163. 42

[183] D. Wong, K. Fung, J. Liu, and V. Wei, *On the RS-Code Construction of Ring Signature Schemes and a Threshold Setting of RST*, in ICICS 2003, vol. 2836 of LNCS, Springer, 2003, pp. 34–46. 41

[184] S. Xu and M. Yung, *k-Anonymous Secret Handshakes with Reusable Credentials*, in ACM Conference on Computer and Communications Security (ACM CCS 2004), ACM, 2004, pp. 158–167. 39

[185] X. xue Li, H. feng Qian, and J. hua Li, *Democratic Group Signatures with Threshold Traceability*, Journal of Shanghai Jiaotong University (Science), 14 (2009), pp. 98–101. Available at http://eprint.iacr.org/2008/112.pdf. 29

[186] G. Yang, D. S. Wong, X. Deng, and H. Wang, *Anonymous Signature Schemes*, in Public Key Cryptography (PKC 2006), vol. 3958 of LNCS, Springer, 2006, pp. 347–363. 37

[187] G. X. Yao, J. Fan, R. C. Cheung, and I. Verbauwhede, *A High Speed Pairing Coprocessor Using RNS and Lazy Reduction*. Cryptology ePrint Archive, Report 2011/258, 2011. http://eprint.iacr.org/2011/258. 204, 205, 223, 224, 234, 235, 240, 243, 245, 246

[188] C. Zhang, Y. Liu, and D. He, *A New Verifiable Ring Signature Scheme Based on Nyberg-Rueppel Scheme*, in Signal Processing, 2006 8th International Conference on, vol. 4, IEEE, 2006, pp. 16–20. 41

[189] R. Zhang and H. Imai, *Strong Anonymous Signatures*, in International Conference on Information Security and Cryptology (Inscrypt 2008), vol. 5487 of LNCS, Springer, 2009, pp. 60–71. 37