Blind Password Registration for Two-Server Password Authenticated Key Exchange and Secret Sharing Protocols

Franziskus Kiefer[1] and Mark Manulis[2]

[1] Mozilla
Berlin, Germany
mail@franziskuskiefer.de
[2] Surrey Center for Cyber Security
Department of Computer Science, University of Surrey, UK
mark@manulis.eu

Abstract. Many organisations enforce policies on the length and formation of passwords to encourage selection of strong passwords and protect their multi-user systems. For Two-Server Password Authenticated Key Exchange (2PAKE) and Two-Server Password Authenticated Secret Sharing (2PASS) protocols, where the password chosen by the client is secretly shared between the two servers, the initial remote registration of policy-compliant passwords represents a major problem because none of the servers is supposed to know the password in clear.
We solve this problem by introducing *Two-Server Blind Password Registration (2BPR)* protocols that can be executed between a client and the two servers as part of the remote registration procedure.
2BPR protocols guarantee that secret shares sent to the servers belong to a password that matches their combined password policy and that the plain password remains hidden from any attacker that is in control of at most one server. We propose a security model for 2BPR protocols capturing the requirements of *policy compliance* for client passwords and their *blindness* against the servers. Our model extends the adversarial setting of 2PAKE/2PASS protocols to the registration phase and hence closes the gap in the formal treatment of such protocols. We construct an efficient 2BPR protocol for ASCII-based password policies, prove its security in the standard model, give a proof of concept implementation, and discuss its performance.

1 Introduction

Password policies set by organisations aim to rule out potentially "weak" passwords and by this contribute to the protection of multi-user systems. In traditional web-based password authentication mechanisms a password policy chosen by the server is typically enforced during the password registration phase — the corresponding compliance check is performed either by the client or on the server side, depending on the available trust assumptions. If the client is not

trusted with the selection of a policy-compliant password, then the compliance check must be performed by the server. The most common approach in this case is to transmit chosen passwords over a secure channel, e.g., TLS channel, to the server that performs the check on the received (plain) password. The drawback of this approach, however, is that the client's password is disclosed to the server. Although this approach represents a common practice nowadays, its main drawback is the necessity to trust the server to process and store the received password in a protected way, e.g., by hashing it. This trust assumption often does not hold in practice as evident from the frequent server-compromise attacks based on which plain password databases have been disclosed [1,2,3,4].

Considering that "password-cracking tools" such as Hashcat [5] and John the Ripper [6] are very efficient, it is safe to assume that leaked password hashes are not safer than un-hashed ones when compromised by an attacker [7,8,9,10]. The notion of threshold and two-server password authenticated key-exchange [11,12] has been proposed where the password is not stored on a single server but split between a number of servers such that leakage of a password database on a non-qualified subset does not reveal the password. The two-server setting is regarded as more practical (in comparison to a more general threshold setting) given that if one server is compromised a notification to change the password can be sent out to the clients. Two-server password authenticated key-exchange protocols (2PAKE) [13,14,15] split the client's password pw into two shares $\mathfrak{s}_1$ and $\mathfrak{s}_2$ such that each share is stored on a distinct server. During the authentication phase both servers collaborate in order to authenticate the client. Yet, no server alone is supposed to learn the plain password. A second, more recent development in two-server (and threshold) password protocols is password authenticated secret sharing (PASS) [16,17,18] where a client stores shares of a (high-entropy) secret key on a number of servers and uses a (low-entropy) password to authenticate the retrieval process.

Registering password shares for 2PAKE/2PASS protocols however makes it impossible for the servers to verify their password policies upon registration unless the password is transferred to each of them in plain. This however, would imply that the client trusts both servers to securely handle its password, which contradicts the purpose and trust relationships of multi-server protocols. The use of two-server password protocols in a remote authentication setting, therefore, requires a suitable password registration procedure in which none of the servers would receive information enabling it (or an attacker in control of the server) to deliberately or inadvertently recover the client's password. This registration procedure must further allow for policy compliance checks to be performed by the servers since secret sharing per se does not protect against "weak" passwords. A trivial approach of sending $\mathfrak{s}_1$ and $\mathfrak{s}_2$ to the corresponding servers over secure channels is not helpful here since it is not clear how the two servers can perform the required compliance check. To alleviate a similar problem in the verifier-based PAKE setting, Kiefer and Manulis [19] introduced the concept of zero-knowledge password policy checks (ZKPPC), where upon registration the client can prove to the server the compliance of its chosen password with re-

spect to the server's policy without disclosing the actual password. In this work, we propose the concept of blind password registration for two-server password protocols and thus show how to realise secure registration of password shares in a way that protects against at most one malicious server (if both servers are malicious, the attacker obviously gets the password), yet allows both servers to check password compliance against their mutual password policy. It bases on techniques introduced in the framework for ZKPPC from [19] but uses a security model for the entire blind setup process and is based in the two-server setting which brings additional challenges. Two-server Blind Password Registration (2BPR) is not vulnerable to offline dictionary attacks as long as one server remains honest. This is in contrast to the single-server setting where an attacker is always able to perform offline dictionary attacks on password verifiers after compromising a server. Our main contribution is the 2BPR security model and the corresponding protocol for secure registration of 2PAKE/2PASS passwords. We show how secure distribution of password shares can be combined with an appropriate policy-compliance proof for the chosen password in a way that does not reveal the password and can still be verified by both servers. Our 2BPR protocol can be used to enforce policies over the alphabet of all 94 printable ASCII characters[3], including typical requirements on password length and character types.

2 Preliminaries

In this section we recall the underlying primitives and concepts that are used in the construction of our two-server blind password registration protocol.

2.1 Commitments

Let $\mathtt{C} = (\mathtt{CSetup}, \mathtt{Com})$ denote a commitment scheme and $C \leftarrow \mathtt{Com}(x; r)$ a commitment on x using randomness r, with $\mathtt{CSetup}$ generating parameters for $\mathtt{C}$. A commitment scheme $\mathtt{C} = (\mathtt{CSetup}, \mathtt{Com})$ is *efficient* if $\mathtt{CSetup}(\lambda)$ and $(C, d) \leftarrow \mathtt{Com}(x; r)$ are computable in polynomial time, *complete* if $\mathtt{Com}(d) = (C, d)$ for $(C, d) \leftarrow \mathtt{Com}(x; r)$, and secure if it is

- Binding: For all PPT adversaries $\mathcal{A}$ there exists a negligible function $\varepsilon_{\mathrm{bi}}(\cdot)$ such that for all $(x, x', r, r', C) \leftarrow A$: $\Pr[x \neq x' \wedge (C, d) = \mathtt{Com}(x; r) \wedge (C, d') = \mathtt{Com}(x'; r')] \leq \varepsilon_{\mathrm{bi}}(\lambda)$,
- Hiding: For all PPT adversaries $\mathcal{A}$ there exists a negligible function $\varepsilon_{\mathrm{hi}}(\cdot)$ such that for all x_0, x_1 with $|x_0| = |x_1|$ and $b \in_R \{0, 1\}, (C, d) \leftarrow \mathtt{Com}(x_b; r)$ and $b' \leftarrow A(C, x_1, x_2)$: $\Pr[b = b'] \leq 1/2 + \varepsilon_{\mathrm{hi}}(\lambda)$.

[3] Note that using other encodings such as UTF-8 is possible but might influence performance due to a different size of possible characters.

Pedersen commitments [20] We use perfectly hiding, computationally binding, homomorphic Pedersen commitments [20] defined as follows. Let $\mathtt{C_P} = (\mathtt{CSetup}, \mathtt{Com})$ with $(g, h, q, \lambda) \leftarrow \mathtt{CSetup}(\lambda)$ and $C \leftarrow \mathtt{Com} = (x; r) = g^x h^r$ denote the Pedersen commitment scheme where g and h are generators of a cyclic group G of prime order q with bit-length in the security parameter λ and the discrete logarithm of h with respect to base g is not known. Pedersen commitments are *additively homomorph*, i.e. for all $(C_i, d_i) \leftarrow \mathtt{Com}(x_i; r_i)$ for $i \in 0, \ldots, m$ it holds that $\prod_{i=0}^{m} C_i = \mathtt{Com}(\sum_{i=0}^{m} x_i; \sum_{i=0}^{m} r_i)$.

Trapdoor commitments In order to build zero-knowledge proofs of knowledge with malicious verifiers we require a trapdoor commitment scheme, which allows a party knowing the correct trapdoor to open a commitment to any value. Fortunately, Pedersen commitments are trapdoor commitments as they can be opened to any element using the discrete logarithm $\log_g(h)$ as trapdoor.

2.2 Zero Knowledge Proofs

A zero-knowledge proof is executed between a prover and a verifier, proving that a word x is in a language L, using a witness w proving so. An interactive protocol ZKP for a language L between prover P and verifier V is a zero knowledge proof if the following holds:

- Completeness: If $x \in L$, V accepts if P holds a witness proving so.
- Soundness: For every malicious prover $P^*(x)$ with $x \in L$ that the probability of making V accept is negligible.
- Zero-Knowledge: If $x \in L$, then there exists an efficient simulator Sim that on input of x is able to generate a view, indistinguishable from the view of a malicious verifier V^*.

A zero-knowledge proof of knowledge ZKPoK is a zero-knowledge proof with the following special soundness definition:

- Soundness: There exists an efficient knowledge extractor Ext that can extract a witness from any malicious prover $P^*(x)$ with $x \in L$ that has non-negligible probability of making V accept.

We use the following committed Σ-protocol to ensure extractability (ZKPoK) and simulatability when interacting with a malicious verifier [21,22]. Let $P_1(x, w, r)$ and $P_2(x, w, r, c)$ denote the two prover steps of a Σ-protocol and $H : \{0,1\}^* \mapsto \mathbb{Z}_q$ a collision-resistant hash function. A committed Σ-protocol based on Pedersen commitments is then given by the following steps:

- The prover computes $m_1 \leftarrow P_1(x, w, r), \mathtt{Co} \leftarrow \mathtt{Com}(H(x, m_1); r_1) = g^{H(x,m_1)} h^{r_1}$, and sends $\mathtt{Co}$ to the verifier.
- The verifier picks random challenge $\mathtt{Ch} = c$ and returns it to the prover.
- The prover computes $m_2 \leftarrow P_2(x, w, r, c), \mathtt{Rs}_1 \leftarrow \mathtt{Com}(H(m_2); r_2) = g^{H(m_2)} h^{r_2}$, and sends $\mathtt{Rs}$ to the verifier.

– Further, the prover opens the commitments $\mathtt{Co}$ and $\mathtt{Rs}_1$ by sending $\mathtt{Rs}_2 = (x, m_1, m_2, r_1, r_2)$ to the verifier.
– The verifier accepts if both commitments are valid and if the verification of the Σ-protocol (x, m_1, c, m_2) is successful.

We note that in the malicious verifier setting, this type of protocol is a concurrent zero-knowledge proof since its security proof does not require rewinding [21,22]. We observe that all zero-knowledge protocols used in this work are committed Σ-protocols those security relies on the hardness of the discrete logarithm problem in G and the collision resistance property of H.

Passwords We adopt the reversible, structure-preserving encoding scheme from [19] that (uniquely) maps strings of printable ASCII characters to integers. We use pw for the ASCII password string, $c_i = \mathrm{pw}[i]$ for the i-th ASCII character in pw, and integer π for the encoded password string. The encoding proceeds as follows: $\pi \leftarrow \mathtt{PWDtoINT}(\mathrm{pw}) = \sum_{i=0}^{n-1} b^i(\mathtt{ASCII}(c_i) - 32)$ for the password string pw and $\pi_i \leftarrow \mathtt{CHRtoINT}(c_i) = \mathtt{ASCII}(c_i) - 32$ for the i-th *unshifted* ASCII character in pw. Note that n denotes the length of pw and $b \in \mathbb{N}$ is used as shift base. (We refer to [19] for a discussion on the shift base b. Note, however, that shift base related attacks on the password verifier from [19] are not possible in our two-server setting.) The $\mathtt{ASCII}$ function returns the decimal ASCII code of a character.

Remark 1. While password distribution is important for the security of password registration protocols for Verifier-based PAKE [19], the role of password distribution in the two-server setting is different. Since each server stores only a random-looking password share, offline dictionary attacks from an attacker who compromises at most one of the two servers become infeasible. Security of 2BPR protocols defined in this work is therefore independent of client passwords. Note however that the password strength still continues to play an important role for the security of 2PAKE/2PASS protocols, where it influences the probability of successful online dictionary attacks.

Password Sharing We focus on the additive password sharing of client passwords, i.e. $\pi = \mathfrak{s}_0 + \mathfrak{s}_1 \bmod q$ over a prime-order group G_q. Such sharing has been used in various 2PAKE protocols, including [15,23,24,25]. To be used in combination with 2PASS protocols such as [17] one can define the password as g^π and thus adopts the multiplicative sharing $g^\pi = g^{\mathfrak{s}_0} g^{\mathfrak{s}_1}$. Password shares are created as $\mathfrak{s}_0 \in_R \mathbb{Z}_q$ and $\mathfrak{s}_1 = \pi - \mathfrak{s}_0 \bmod q$. We remark that other sharing options such as XOR have been used in literature] [13,14] but are not supported by our 2BPR protocol.

Password Policies We represent password policies as in [19], i.e. a password policy $f = (R, n_{\mathtt{min}})$ consists of a simplified regular expression R that defines

ASCII subsets that must be present in the chosen password string and the minimum length $n_{\texttt{min}}$ of the password string. The expression R is defined over the *four* ASCII subsets $\Sigma = \{d, u, l, s\}$ with digits d, upper case letters u, lower case letters l and symbols s, and gives the minimum frequency of a character from the subset that is necessary to fulfil the policy; for instance, $R = ulld$ means that policy-conform password strings must contain at least one upper case letter, two lower case letters and one digit. In the two-server setting, if each of the servers has its own password policy, i.e. f_0 and f_1, then registered passwords would need to comply with the *mutual password policy* defined as $f = f_0 \cap f_1 = (\max(R_0, R_1), \max(n_{\texttt{min}0}, n_{\texttt{min}1}))$, where $\max(R_0, R_1)$ is the regular expression with the maximum number of characters from each of the subsets u, l, d, s from R_0 and R_1. A mutual policy is fulfilled, i.e. $f(\text{pw}) = \texttt{true}$, iff $f_0(\text{pw}) = \texttt{true}$ and $f_1(\text{pw}) = \texttt{true}$, and not fulfilled, i.e. $f(\text{pw}) = \texttt{false}$, iff $f_0(\text{pw}) = \texttt{false}$ or $f_1(\text{pw}) = \texttt{false}$. We mainly operate on the integer representation π of a password string pw throughout this paper and sometimes write $f(\pi)$, which means $f(\text{pw})$ for $\pi \leftarrow \texttt{PWDtoINT}(\text{pw})$. Further note that a character $c_i \in \text{pw}$ is called *significant* if this character is necessary to fulfil R and we denote the corresponding set $R_j \in R$ as a *significant set* for the policy.

Password Dictionaries A password dictionary $\mathcal{D}_f$, if not specified otherwise, is a set of password strings adhering to a given policy $f = (R, n_{\texttt{min}})$, i.e. their length is limited by $n_{\texttt{min}} \leq |\text{pw}|$ and the required types of characters are identified by R. We denote the size of a dictionary $\mathcal{D}$ by $|\mathcal{D}|$. We omit index f if the policy is clear from the context. We further define dictionary $\mathcal{D}_{f,n}$ holding policy-conform passwords according to f of length n and will use it throughout the paper. In order to be able to use the optimal dictionary $\mathcal{D}_f$, the client would either have to prove correctness of password characters that are not necessary for R without revealing their number (which seems impossible with the approach used in this paper), or use a fixed password length to hide n in it (which is inefficient). (Note that we only consider reasonable dictionaries sizes, i.e. $|\mathcal{D}_{f,n}| > 1$.)

3 Two-Server Blind Password Registration

Two-server Blind Password Registration (2BPR) allows a client to register password shares with two servers for later use in 2PAKE/2PASS protocols and prove that the shares can be combined to a password that complies with the mutual password policy of both servers, without disclosing the password. A 2BPR protocol is executed between client $\mathcal{C}$ and two servers S_0 with password policy f_0 and S_1 with password policy f_1. $\mathcal{C}$ interacts with S_0 and S_1 in order to distribute shares of a freshly chosen password string pw and prove its compliance with the mutual policy, i.e. $f_0(\text{pw}) = \texttt{true}$ and $f_1(\text{pw}) = \texttt{true}$. A 2BPR protocol between an honest client $\mathcal{C}$ and two honest servers S_0 and S_1 is correct if S_0 and S_1 accept their password shares if and only if the client is able to prove the following statement for $f = f_0 \cap f_1$:

$$(\text{pw}, \mathfrak{s}_0, \mathfrak{s}_1) : \ \texttt{PWDtoINT}(\text{pw}) = \mathfrak{s}_0 + \mathfrak{s}_1 \ \wedge \ f(\text{pw}) = \texttt{true}. \tag{1}$$

Note that the 2BPR protocol can be used to register new clients or to register new passwords for existing clients. The following definition formally captures the functionality of 2BPR protocols.

Definition 1 (Two-Server Blind Password Registration). *A 2BPR protocol is executed between a client $\mathcal{C}$ and two servers S_0 and S_1, holding a password policy f_b each, such that the servers, when honest, eventually accept password shares $\mathfrak{s}_b$ of a policy compliant, client chosen password* pw *iff* $f(\text{pw}) = \texttt{true}$ *for* $f = f_0 \cap f_1$, $\texttt{PWDtoINT}(\text{pw}) = \mathfrak{s}_b + \mathfrak{s}_{1-b}$ *and* $b \in \{0, 1\}$.

Definition 1 requires that password shares $\mathfrak{s}_0$ and $\mathfrak{s}_1$ can be combined to the policy-compliant integer password π. The corresponding verification must therefore be part of the 2BPR protocol. Otherwise, the client could register password shares $\mathfrak{s}_0$ and $\mathfrak{s}_1$ that can both be combined to a policy compliant password in the respective proofs with the servers, but combining $\mathfrak{s}_0$ and $\mathfrak{s}_1$ might result in a password that is *not* policy compliant, i.e. $f(\mathfrak{s}_0 + \mathfrak{s}') = \texttt{true}$ and $f(\mathfrak{s}_1 + \mathfrak{s}'') = \texttt{true}$ but $f(\pi) \neq \texttt{true}$. This further ensures that servers hold valid password shares, which is crucial for the security of 2PAKE/2PASS protocols that should be executed later with these password shares. We assume that the protocol is initiated by servers (possibly after the client expresses his interest to register). This allows each server to send its password policy to the client. We further assume that both servers can communicate with each other over an authenticated and confidential channel. This communication can either be done directly between the servers or indirectly using the client to transmit messages.

3.1 Security Model for 2BPR Protocols

2BPR protocols must guarantee that the client knows the sum $\texttt{PWDtoINT}(\text{pw})$ of the password shares $\mathfrak{s}_0$ and $\mathfrak{s}_1$, and that pw fulfils both password policies f_0 and f_1 if both servers accept the registration procedure. We translate Eq. (1) into a game-based security model that captures 2BPR security in form of two security requirements. The first requirement is called *Policy Compliance (PC)* of the registered password. In particular, if both servers are honest while accepting their password shares in the 2BPR protocol, the combination π of the shares represents a password compliant with their mutual policy $f = f_0 \cap f_1$, i.e. $f(\mathfrak{s}_b + \mathfrak{s}_{1-b}) = \texttt{true}$. The second requirement relates to the fact that servers should not learn anything about the registered password and is therefore called *Password Blindness (PB)*, i.e. a malicious server S_b may only learn whether a registered password is compliant with the mutual policy and nothing else. We observe that the blindness property must hold for all possible password policies and all compliant passwords. PB also implies impossibility of mounting an offline dictionary attack after observing 2BPR executions or through gaining access to and controlling at most one of the servers.

Setup and Participants Protocol participants $\mathcal{C}, S_0, S_1$ with $\mathcal{C}$ from the universe of clients and S_0, S_1 from the universe of servers have common inputs, necessary for the execution of the protocol. Instances of protocol participants

$\mathcal{C}$ or S are denoted $\mathcal{C}_i$, $S_{0,i}$ or $S_{1,i}$. Protocol participants without specified role are denoted by P, and S_b and S_{1-b} for unspecified servers. A client can register one password with any pair of servers from the universe. We use $\mathcal{C}$ and S_b as unique identifiers for the client and servers (e.g. $\mathcal{C}$ can be seen as a *username* that will be stored by servers alongside with password shares). We say a client $\mathcal{C}$ registers a password share for $(\mathcal{C}, S_{1-b})$ at server S_b and a password share for $(\mathcal{C}, S_b)$ at server S_{1-b}. There can be only at most one (most recent) password share registered at S_b resp. S_{1-b} for $(\mathcal{C}, S_{1-b})$ resp. $(\mathcal{C}, S_b)$ at any given time. A tuple $(\mathcal{C}, S_{1-b}, \mathfrak{s}_b)$ is stored on server S_b and tuple $(\mathcal{C}, S_b, \mathfrak{s}_{1-b})$ on server S_{1-b} only if the 2BPR protocol is viewed as successful by the servers.

Oracles A PPT adversary $\mathcal{A}$ has access to Setup, Send, $\mathsf{Execute}$ and $\mathsf{Corrupt}$ oracles for interaction with the protocol participants.

- $\mathsf{Setup}(\mathcal{C}, S_0, S_1, \mathrm{pw}')$ creates new instances of all participants and stores identifiers of the other parties to each participant. To this end the client receives the server policies $f_0 \cap f_1 = f$ and either chooses a new policy compliant password $\mathrm{pw} \in \mathcal{D}_f$ if $\mathrm{pw}' = \perp$ or uses $\mathrm{pw} = \mathrm{pw}'$.
- $\mathsf{Execute}(\mathcal{C}, S_0, S_1)$ models a passive attack and executes a 2BPR protocol between new instances of $\mathcal{C}$, S_0 and S_1. It returns the protocol transcript and the internal state of all corrupted parties.
- $\mathsf{Send}_{\mathcal{C}}(\mathcal{C}_i, S_{b,j}, m)$ sends message m, allegedly from client instance $\mathcal{C}_i$, to server instance $S_{b,j}$ for $b \in \{0,1\}$. If $\mathcal{C}_i$ or $S_{b,j}$ does not exist, the oracle aborts. Note that any instance $\mathcal{C}_i$ and $S_{b,j}$ was thus set up with Setup and therefore has an according partner instance $S_{1-b,j}$. If all participants exist, the oracle returns the server's answer m' if there exists any. Necessary inter server communication is performed in $\mathsf{Send}_{\mathcal{C}}$ queries. If $m = \perp$, server $S_{b,j}$ returns its first protocol message if it starts the protocol.
- $\mathsf{Send}_S(S_{b,i}, \mathcal{C}_j, m)$ sends message m, allegedly from server instance $S_{b,i}$ for $b \in \{0,1\}$, to client instance $\mathcal{C}_j$. If $S_{b,i}$ or $\mathcal{C}_j$ does not exist, the oracle aborts. Note that any instance $S_{b,i}$ and $\mathcal{C}_j$ was thus set up with Setup and therefore has an according partner instance $S_{1-b,i}$. If all participants exist, the oracle returns the client's answer m' if there exists any. If $m = \perp$, server $S_{b,i}$ returns its first message if he starts the protocol.
- $\mathsf{Send}_{SS}(S_{b,i}, S_{1-b,j}, m)$ sends message m, from server instance $S_{b,i}$ for $b \in \{0,1\}$, to server instance $S_{1-b,j}$. If $S_{b,i}$ or $S_{1-b,j}$ does not exist, the oracle aborts. Note that any instance $S_{b,i}$ and $S_{1-b,j}$ was thus set up with Setup. If all participants exist, the oracle returns the server's answer m' if there exists any.
- $\mathsf{Corrupt}(S_b)$ allows the adversary to corrupt a server S_b and retrieve its internal state, i.e. stored messages and randomness, and the list of stored password shares $(\mathcal{C}, S_{1-b}, \mathfrak{s}_b)$. S_b is marked *corrupted*.

Note that we allow the adversary to register passwords with servers without requiring existence of a client instance $\mathcal{C}_i$ in a successful registration session. This is because we do not assume authenticated clients, i.e. client identifiers $\mathcal{C}$ are unique but not secret and can therefore be used by the adversary.

Policy Compliance This is a natural security property of 2BPR protocols, requiring that registered client passwords comply with the mutual policy $f(\text{pw}) = \texttt{true}$. The attacker here plays the role of the client trying to register a password pw that is *not* policy compliant at two honest servers.

Definition 2 (Policy Compliance). *Policy compliance of a 2BPR protocol holds if for every PPT adversary $\mathcal{A}$ with access to* Setup *and* $\mathsf{Send}_{\mathcal{C}}$ *oracles the probability that two server instances $S_{b,i}$ and $S_{1-b,j}$ exist after $\mathcal{A}$ stopped that accepted $(\mathcal{C}, S_{1-b}, \mathfrak{s}_b)$, $(\mathcal{C}, S_b, \mathfrak{s}_{1-b})$ respectively, with $f(\mathfrak{s}_b + \mathfrak{s}_{1-b}) = \texttt{false}$ is negligible.*

Password Blindness This property requires that every password, chosen and set-up by an honest client must remain hidden from an adversary who may corrupt at most one of the two servers, thus obtaining the internal state and taking full control over the corrupted server. We model password blindness through a distinguishing experiment where the attacker, after interacting with the oracles, outputs a challenge comprising of two passwords (pw_0 and pw_1), two clients ($\mathcal{C}_0$ and $\mathcal{C}_1$), and a pair of servers (S_0 and S_1). After a random assignment of passwords to the two clients, the adversary interacts with the oracles again and has to decide which client used which password in the 2BPR protocol execution. This is formalised in the following definition.

Definition 3 (Password Blindness). *The password blindness property of a 2BPR protocol Π holds if for every PPT adversary $\mathcal{A}$ there exists a negligible function $\varepsilon(\cdot)$ such that*

$$\mathsf{Adv}^{\text{PB}}_{\Pi,\mathcal{A}} = \left| \Pr[\mathsf{Exp}^{\text{PB}}_{\Pi,\mathcal{A}} = 1] - \frac{1}{2} \right| \leq \varepsilon(\lambda).$$

$\mathsf{Exp}^{\text{PB}}_{\Pi,\mathcal{A}}$:
$(\mathcal{C}_0, \mathcal{C}_1, S_0, S_1, \text{pw}_0, \text{pw}_1) \leftarrow \mathcal{A}_1^{\mathsf{Setup},\mathsf{Send}_S,\mathsf{Send}_{SS},\mathsf{Execute},\mathsf{Corrupt}}$
check $\text{pw}_0, \text{pw}_1 \in \mathcal{D}_{f_0 \cap f_1}$, $|\text{pw}_0| = |\text{pw}_1|$, $\mathcal{C}_0, \mathcal{C}_1 \in \{\mathcal{C}\}$ *and* $S_0, S_1 \in \{S\}$
$b' \leftarrow \mathcal{A}^{\mathsf{Setup}',\mathsf{Send}_S,\mathsf{Send}_{SS},\mathsf{Execute},\mathsf{Corrupt}}(\lambda, \mathcal{D}, \{\mathcal{C}\}, \{S\})$
if S_0 *or* S_1 *is uncorrupted, return* $b = b'$*; otherwise return* 0

where the modified oracle Setup' *(in contrast to* Setup*) picks a random bit* $b \in_R \{0,1\}$ *and uses* pw_b *as a password for client* $\mathcal{C}_0$ *and* pw_{1-b} *for* $\mathcal{C}_1$

4 An Efficient Two-Server BPR Protocol

Before diving into technical details, we give a high-level description of our 2BPR protocol. We assume that client $\mathcal{C}$ selected two servers S_0 and S_1 to register with. We also assume the existence of server-authenticated and confidential channels (e.g. TLS channels [26,27,28]) between $\mathcal{C}$ and each S_b, $b \in \{0,1\}$ as well as between S_0 and S_1. These channels prevent active impersonation of any server S_b,

$b \in \{0,1\}$ and hide the contents of exchanged messages unless the corresponding server is corrupted.

Our 2BPR protocol further assumes a common reference string $\texttt{crs} = (g, h, q)$ containing two generators g and h of a cyclic group of prime order q where $\log_g(h)$ is not known.

At the beginning of the registration phase the client $\mathcal{C}$ commits to the integer representation π of the chosen password string pw and sends this commitment together with a password share $\mathfrak{s}_b$ to the corresponding server S_b, $b \in \{0,1\}$, along with auxiliary information that is needed to perform the policy compliance proof. For the latter, the client needs to prove the knowledge of π in the commitment such that $\pi = \mathfrak{s}_0 + \mathfrak{s}_1$ and that it fulfills both policies f_1 and f_2. Thus, servers S_0 and S_1 eventually register the new client, accept and store the client's password share, iff each S_b holds $\mathfrak{s}_b$ such that $\mathfrak{s}_0 + \mathfrak{s}_1 = \pi$ for $\pi \leftarrow \texttt{PWDtoINT}(\text{pw})$ and $f(\text{pw}) = \texttt{true}$ for $f = f_0 \cap f_1$.

4.1 Protocol Overview

In Figure 1 we give an overview of the 2BPR protocol involving a client $\mathcal{C}$ and two servers S_b, $b \in \{0,1\}$. The protocol proceeds in three phases. In the first phase *(client preparation)* the client chooses $\text{pw} \in_R \mathcal{D}_f$, encodes it to π, computes shares $\mathfrak{s}_0$ and $\mathfrak{s}_1$, and computes commitments $\mathfrak{C}_0, \mathfrak{C}_1, \mathfrak{D}_0, \mathfrak{D}_1$ to the shares and the password. In the second phase *(password registration)* $\mathcal{C}$ interacts with each server S_b, $b \in \{0,1\}$ over a server-authenticated and confidential channel. $\mathcal{C}$ computes a commitment C_i for each encoded character $\pi_i \leftarrow \texttt{CHRtoINT}(c_i)$, $c_i \in \text{pw}$, and a second commitment C'_i as a re-randomised version of C_i. The set $\boldsymbol{C}'$ containing the re-randomised commitments C'_i, is then shuffled and used to prove through the Proof of Membership (**PoM**) protocol that each character committed to in $C'_i \in \boldsymbol{C}'$ is a member of some character set $\omega_{\phi(i)}$, chosen according to policy f. Note that **PoM** must be performed over the *shuffled* set $\boldsymbol{C}'$ of commitments as the server would otherwise learn the type (i.e. lower/upper case, digit, or symbol) of each password character. To further prove that transmitted commitments $\boldsymbol{C}, \mathfrak{C}_b$, and $\mathfrak{D}_b$ are correct, namely that the product of commitments in $\boldsymbol{C}$ commits to the password pw, $\mathfrak{C}_b$ contains the correct share $\mathfrak{s}_b$, and $\mathfrak{D}_b$ contains pw, client and server execute the Proof of Correctness (**PoC**) protocol. Finally, the client proves to each server that set $\boldsymbol{C}'$ is a shuffle of set $\boldsymbol{C}$ by executing the Proof of Shuffle (**PoS**) protocol. This proof is necessary to finally convince both servers that (1) the characters committed to in $\boldsymbol{C}'$ are the same as the characters in the commitments in $\boldsymbol{C}$, which can be combined to password pw (as follows from the **PoC** protocol) and (2) each commitment C_i is for a character $c_i \in \text{pw}$ from some set ω_i, chosen according to policy f (as follows from the **PoM** protocol). For all three committed Σ-protocols (**PoM**, **PoC**, **PoS**) we use variables as defined in Section 2. If each server S_b, $b \in \{0,1\}$ successfully verifies all three committed Σ-protocols and the length of the committed password pw is policy-conform, then both servers proceed with the last phase. In the third phase *(share verification)* the two servers S_0 and S_1 interact with each other over a mutually-authenticated and confidential channel. Each

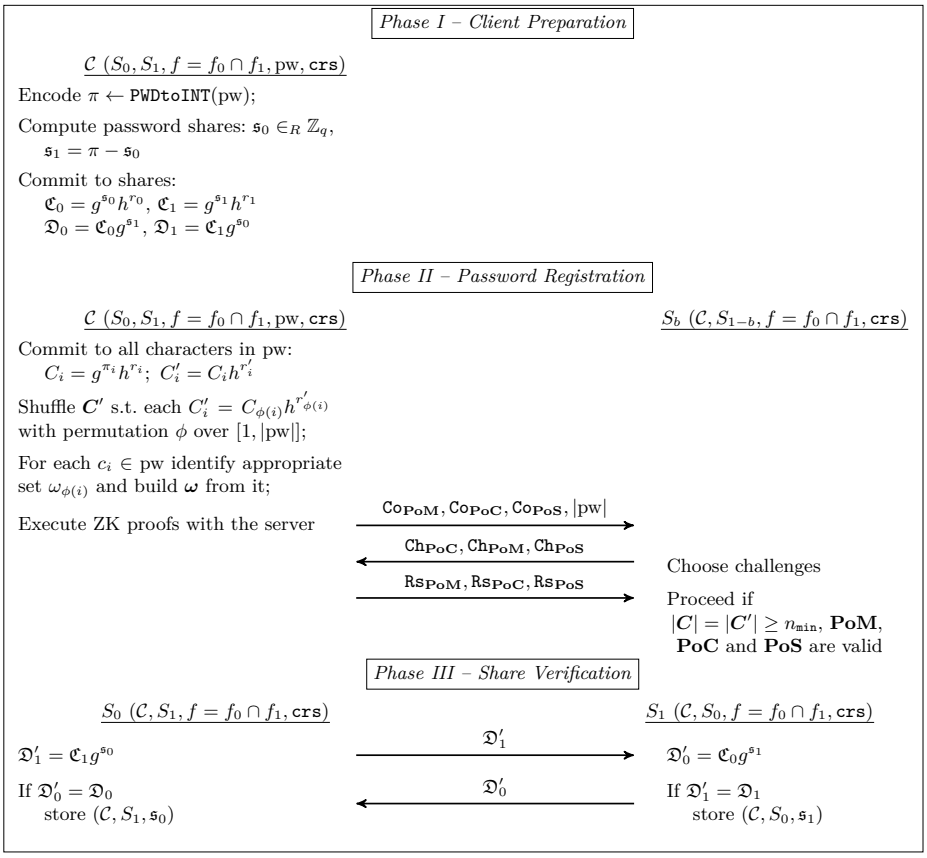

Fig. 1: Two-Server BPR Protocol — A High-Level Overview

$\boldsymbol{\omega}$ contains character sets of $c_{\phi(i)}$ ordered according to permutation ϕ used in **PoM**

S_b computes its verification value $\mathfrak{D}'_{1-b}$ and sends it to S_{1-b}. Upon receiving $\mathfrak{D}'_b$, S_b checks it against $\mathfrak{D}_b$ to verify that the client used the same password with both servers in the second phase, i.e. that $\mathfrak{s}_b + \mathfrak{s}_{1-b} = \pi$. If this verification is successful, S_b stores the client's password share $(\mathcal{C}, S_{1-b}, \mathfrak{s}_b)$ and considers $\mathcal{C}$ as being registered.

4.2 Two-Server BPR Specification

In the following we give a detailed description of the 2BPR protocol. To this end we describe the three proofs **PoC**, **PoM** and **PoS** detailing on their computations. We describe the interaction between client $\mathcal{C}$ and server S_b and therefore only consider one policy f_b. Note that $\mathcal{C}$ and each server S_b perform the same protocol. If both servers accept, the password fulfils the policy $f = f_b \cap f_{1-b}$.

We first describe the client's pre-computations such as password encoding and sharing before giving a detailed description of the proofs. The protocol

operates on a group G of prime-order q with generator g. Further, let $h, f_i \in_R G$ for $i \in [-4, m]$ denote random group elements such that their discrete logarithm with respect to g is unknown. Public parameters of the protocol are defined as $(q, g, h, \boldsymbol{f})$ with $\boldsymbol{f} = \{f_i\}$ where m is at least $n = |\text{pw}|$. In practice m can be chosen big enough, e.g., 100, in order to process all reasonable passwords. Note that we use the range $i \in [0, n-1]$ for characters pw$[i]$, but $[1, x]$ for most other ranges.

Phase I – Client Preparation We assume that password policies f_0 and f_1 are known by the client. This can be achieved by distributing them beforehand with other set-up parameters. The client chooses a password pw $\in_R \mathcal{D}_f$ from the dictionary and encodes it $\pi \leftarrow \texttt{PWDtoINT}(\text{pw})$. The password is shared by choosing a random $\mathfrak{s}_b \in_R \mathbb{Z}_q$ and computing $\mathfrak{s}_{1-b} = \pi - \mathfrak{s}_b$. The client then commits to both password shares $\mathfrak{C}_b = g^{\mathfrak{s}_b} h^{r_b}$ and $\mathfrak{C}_{1-b} = g^{\mathfrak{s}_{1-b}} h^{r_{1-b}}$ with $r_b, r_{1-b} \in_R \mathbb{Z}_q$ and computes commitments to the entire password π with the same randomness, i.e. $\mathfrak{D}_b = \mathfrak{C}_b g^{\mathfrak{s}_{1-b}}$ and $\mathfrak{D}_{1-b} = \mathfrak{C}_{1-b} g^{\mathfrak{s}_b}$. For the following proofs the client further encodes every character $c_i \in$ pw as $\pi_i \leftarrow \texttt{CHRtoINT}(c_i)$.

Phase II – Password Registration The client iterates over all encoded characters π_i to perform the following operations: commit to π_i by computing $C_i = g^{\pi_i} h^{r_i}, C_i' = C_i h^{r_i'}$ for $r_i, r_i' \in_R \mathbb{Z}_q^*$; choose a random permutation $\phi(i)$ over $[1, n]$ to shuffle C_i'; if π_i is *significant* for any $R_j \in R$, set $\omega_{\phi(i)} \leftarrow R_j$, otherwise $\omega_{\phi(i)} \leftarrow \Sigma$ (all ASCII characters). Let $l_i \in \mathbb{N}$ denote the index in $\omega_{\phi(i)}$ such that $c_i = \omega_{\phi(i)}[l_i]$. Values $(C_i, C_i', \omega_{\phi(i)}, \phi(i), l_i, \pi_i, r_i, r_i')$ are used in the following zero-knowledge proofs. The client combines previously computed values $\boldsymbol{C} = \{C_i\}$. Shuffled commitments $C'_{\phi(i)}$ and sets $\omega_{\phi(i)}$ are combined according to the shuffled index $\phi(i)$, i.e. $\boldsymbol{C'} = \{C'_{\phi(i)}\}$ and $\boldsymbol{\omega} = \{\omega_{\phi(i)}\}$. Once these computations are finished $\mathcal{C}$ and S_b proceed with the protocol. In the following we describe the three proofs **PoM**, **PoC** and **PoS** and define their messages.

Proof of Correctness (PoC) This proof links the password shares, sent to each server, to the proof of policy compliance and shows knowledge of the other password share. We define the proof of correctness for an encoded password π, which proves that share $\mathfrak{s}_b$ can be combined with a second share $\mathfrak{s}_{1-b}$ such that $\pi = \mathfrak{s}_b + \mathfrak{s}_{1-b}$ and that the received commitments to password characters c_i can be combined to a commitment to that same password π. **PoC** is defined as a committed zero-knowledge proof between $\mathcal{C}$ and S_b for the statement

$$\mathsf{ZKP}\{(\pi, r_{1-b}, r_b, r_{Cb}) : \mathfrak{C}_{1-b} g^{\mathfrak{s}_b} = g^{\pi} h^{r_{1-b}} \wedge \prod_{i=0}^{n-1} C_i^{b^i} = g^{\pi} h^{r_{Cb}} \wedge \mathfrak{D}_b = g^{\pi} h^{r_b}\}.$$

$C_i = g^{\pi_i} h^{r_i}$ are character commitments from the set-up stage and $r_{Cb} = \sum_{i=0}^{n-1} b^i \cdot r_i$ is the combined randomness from the character commitments C_i. $\mathfrak{C}_{1-b} = g^{\mathfrak{s}_{1-b}} h^{r_{1-b}}$, $\mathfrak{D}_b = \mathfrak{C}_b g^{\mathfrak{s}_{1-b}}$, and $\mathfrak{C}_b = g^{\mathfrak{s}_b} h^{r_b}$ are the share and password commitments from the client preparation phase. This incorporates the link of the password

commitment to the product of the commitments to the single characters with the proof of knowledge of the combined password $\pi = \mathfrak{s}_b + \mathfrak{s}_{1-b}$. The messages for **PoC** are computed as follows:

1. The client chooses random k_π, $k_{\rho b}$, $k_{\rho(1-b)}, k_{\rho C} \in_R \mathbb{Z}_q$, computes $t_{C(1-b)} = g^{k_\pi} h^{k_{\rho(1-b)}}$, $t_C = g^{k_\pi} h^{k_{\rho C}}$ and $t_{Db} = g^{k_\pi} h^{k_{\rho b}}$. The first message with $r_{\mathtt{Com}_{\mathbf{PoC}}} \in_R \mathbb{Z}_q$ is then given by commitment

$$\mathtt{Com}_{\mathbf{PoC}} = g^{H(\mathfrak{C}_{1-b} g^{\mathfrak{s}_b}, \{C_i\}, \mathfrak{D}_b, t_{C(1-b)}, t_C, t_{Db})} h^{r_{\mathtt{Com}_{\mathbf{PoC}}}}.$$

2. After receiving $\mathtt{Com}_{\mathbf{PoC}}$ from the client the server chooses a random challenge $\mathtt{Ch}_{\mathbf{PoC},b} \in_R \mathbb{Z}_q$ and sends it back to the client.
3. After receiving the challenge $\mathtt{Ch}_{\mathbf{PoC},b}$, the client computes $s_\pi = k_\pi + \mathtt{Ch}_{\mathbf{PoC},b}\pi$, $s_{\rho(1-b)} = k_{\rho(1-b)} + \mathtt{Ch}_{\mathbf{PoC},b} r_{1-b}$, $s_{\rho C} = k_{\rho C} + \mathtt{Ch}_{\mathbf{PoC},b} \sum_{i=0}^{n-1} b^i r_i$ and $s_{\rho b} = k_{\rho b} + \mathtt{Ch}_{\mathbf{PoC},b} r_b$ before computing the next message with $r_{\mathtt{Rs}_{\mathbf{PoC}}} \in_R \mathbb{Z}_q$

$$\mathtt{Rs}_{\mathbf{PoC}}1 = g^{H(s_\pi, s_{\rho(1-b)}, s_{\rho C}, s_{\rho b})} h^{r_{\mathtt{Rs}_{\mathbf{PoC}}}}.$$

4. Eventually the client sets the decommitment message

$$\mathtt{Rs}_{\mathbf{PoC}}2 = (\mathfrak{s}_b, \mathfrak{C}_{1-b}, \{C_i\}, \mathfrak{D}_b, t_{C(1-b)}, t_C, t_{Db}, s_\pi, s_{\rho(1-b)}, s_{\rho C}, s_{\rho b}, r_{\mathtt{Com}_{\mathbf{PoC}}}, r_{\mathtt{Rs}_{\mathbf{PoC}}}).$$

$\mathtt{Rs}_{\mathbf{PoC}}1$ and $\mathtt{Rs}_{\mathbf{PoC}}2$ form together client's response message $\mathtt{Rs}_{\mathbf{PoC}}$. The server verifies the proof by checking the following:

- $\mathtt{Com}_{\mathbf{PoC}} \stackrel{?}{=} g^{H(\mathfrak{C}_{1-b} g^{\mathfrak{s}_b}, \{C_i\}, \mathfrak{D}_b, t_{C(1-b)}, t_C, t_{Db})} h^{r_{\mathtt{Com}_{\mathbf{PoC}}}}$
- $\mathtt{Rs}_{\mathbf{PoC}}1 \stackrel{?}{=} g^{H(s_\pi, s_{\rho(1-b)}, s_{\rho C}, s_{\rho b})} h^{r_{\mathtt{Rs}_{\mathbf{PoC}}}}$; $g^{s_\pi} h^{s_{\rho(1-b)}} \stackrel{?}{=} t_{C(1-b)} (\mathfrak{C}_{1-b} g^{\mathfrak{s}_b})^{\mathtt{Ch}_{\mathbf{PoC},b}}$
- $g^{s_\pi} h^{s_{\rho C}} \stackrel{?}{=} t_C (\prod_{i=0}^{n-1} C_i^{b^i})^{\mathtt{Ch}_{\mathbf{PoC},b}}$; $g^{s_\pi} h^{s_{\rho b}} \stackrel{?}{=} t_{Db} \mathfrak{D}_b^{\mathtt{Ch}_{\mathbf{PoC},b}}$

Proof of Membership (PoM) The proof of membership **PoM** proves for every password character $c_{\phi(i)} \in \mathrm{pw}$ that its integer value $\pi_{\phi(i)} \in \omega_{\phi(i)}$ using the shuffled commitments $C'_{\phi(i)}$, i.e.

$$\mathsf{ZKP}\{\{\pi_i, r_i\}_{i \in [0,n-1]} : \; C'_{\phi(i)} = g^{\pi_i} h^{r_i} \;\wedge\; \pi_{\phi(i)} \in \omega_{\phi(i)}\}.$$

This proof consists of the following steps:

1. To prove that every $C'_{\phi(i)}$ commits to a value in the according set $\omega_{\phi(i)}$ the client computes the following values for the first move of the proof:

- $\forall \pi_j \in \omega_{\phi(i)} \wedge \pi_j \neq \pi_{\phi(i)}$: $s_j \in_R \mathbb{Z}_q^*, c_j \in_R \mathbb{Z}_q^*$ and $t_j = g^{\pi_j} h^{s_j} (C'_{\phi(i)} / g^{\pi_j})^{c_j}$
- $k_{\rho_i} \in_R \mathbb{Z}_q^*$; $t_{l_{\phi(i)}} = g^{\pi_i} h^{k_{\rho_i}}$

Values $(\boldsymbol{t}_{\phi(i)}, \boldsymbol{s}_{\phi(i)}, \boldsymbol{c}_{\phi(i)}, k_{\rho_i})$, with $\boldsymbol{t}_{\phi(i)} = \{t_j, t_{l_{\phi(i)}}\}$, $\boldsymbol{s}_{\phi(i)} = \{s_j\}$, and $\boldsymbol{c}_{\phi(i)} = \{c_j\}$ are stored for future use. Note that $t_{l_{\phi(i)}}$ has to be added at the correct position $l_{\phi(i)}$ in $\boldsymbol{t}_{\phi(i)}$. A commitment $\mathtt{Co}_{\mathbf{PoM}} = g^{H(\boldsymbol{\omega}, \boldsymbol{C'}, \boldsymbol{t}_{\phi(i)})} h^{r_{\mathtt{Co}_{\mathbf{PoM}}}}$ with $r_{\mathtt{Co}_{\mathbf{PoM}}} \in_R \mathbb{Z}_q$ is computed as output with $\boldsymbol{\omega} = \{\omega_{\phi(i)}\}$.

2. The server stores received values, checks them for group membership, and chooses a random challenge $\mathtt{Ch}_{\mathbf{PoM}} = c \in_R \mathbb{Z}_q^*$.
3. After receiving the challenge c from the server, the client computes the following verification values for all commitments $C'_{\phi(i)}$ (note that s_j and c_j for all $j \neq l_{\phi(i)}$ are chosen already):

$$c_{l_{\phi(i)}} = c \oplus \bigoplus_{j=1, j \neq l_{\phi(i)}}^{|\omega_{\phi(i)}|} c_j; \quad s_{l_{\phi(i)}} = k_{\rho_{\phi(i)}} - c_{l_{\phi(i)}}(r_i + r'_{\phi(i)}),$$

where i is the index of $C'_{\phi(i)}$ before shuffling. The client then combines $\boldsymbol{s} = \{\boldsymbol{s}_{\phi(i)} \cup \{s_{l_{\phi(i)}}\}\}$ and $\boldsymbol{c} = \{\boldsymbol{c}_{\phi(i)} \cup \{c_{l_{\phi(i)}}\}\}$. Note again that the set union has to consider the position of $l_{\phi(i)}$ to add the values at the correct position. A commitment $\mathtt{Rs}_{\mathbf{PoM}1} = g^{H(\boldsymbol{s},\boldsymbol{c})} h^{r_{\mathtt{Rs}_{\mathbf{PoM}}}}$ with $r_{\mathtt{Rs}_{\mathbf{PoM}}} \in_R \mathbb{Z}_q$ is computed as output.
4. Eventually the client sets the decommitment message with $\boldsymbol{t} = \{\boldsymbol{t}_{\phi(i)}\}$, $\boldsymbol{\omega} = \{\omega_{\phi(i)}\}$, $\boldsymbol{r}_{\mathtt{Co}_{\mathbf{PoM}}} = \{r_{\mathtt{Co}_{\mathbf{PoM}}i}\}$, $\boldsymbol{r}_{\mathtt{Rs}_{\mathbf{PoM}}} = \{r_{\mathtt{Rs}_{\mathbf{PoM}}i}\}$, and $\boldsymbol{C}' = \{C'_{\phi(i)}\}$ to

$$\mathtt{Rs}_{\mathbf{PoM}2} = (\boldsymbol{\omega}, \boldsymbol{C}', \boldsymbol{t}, \boldsymbol{s}, \boldsymbol{c}, \boldsymbol{r}_{\mathtt{Co}_{\mathbf{PoM}}}, \boldsymbol{r}_{\mathtt{Rs}_{\mathbf{PoM}}}).$$

$\mathtt{Rs}_{\mathbf{PoM}1}$ and $\mathtt{Rs}_{\mathbf{PoM}2}$ form together $\mathtt{Rs}_{\mathbf{PoM}}$. To verify the proof, i.e. to verify that every commitment $C'_{\phi(i)}$ in $\boldsymbol{C}'$ commits to a character c_i from either a subset of Σ if significant or Σ if not, the server verifies the following for every set $\omega_{\phi(i)} \in \boldsymbol{\omega}$ with $i \in [1, n]$ and $x = \phi(i)$:

- Let $c_j \in \boldsymbol{c}_i$ for $\boldsymbol{c}_i \in \boldsymbol{c}$ and verify $c \stackrel{?}{=} \bigoplus_{j=1}^{|\omega_i|} c_j$
- Let $\pi_j \in \omega_{\phi(i)}$, $\boldsymbol{s}_i \in \boldsymbol{s}$, $\boldsymbol{t}_i \in \boldsymbol{t}$, and $\boldsymbol{c}_i \in \boldsymbol{c}$, and verify $\boldsymbol{t}_i[j] \stackrel{?}{=} g^{\pi_j} h^{\boldsymbol{s}_i[j]} (C'_i / g^{\pi_j})^{\boldsymbol{c}_i[j]}$ for all $j \in [1, |\omega_{\phi(i)}|]$

The server further verifies commitments $\mathtt{Co}_{\mathbf{PoM}} \stackrel{?}{=} g^{H(\boldsymbol{\omega}, \boldsymbol{C}', \boldsymbol{t})} h^{r_{\mathtt{Co}_{\mathbf{PoM}}}}$ and $\mathtt{Rs}_{\mathbf{PoM}} \stackrel{?}{=} g^{H(\boldsymbol{s},\boldsymbol{c})} h^{r_{\mathtt{Rs}_{\mathbf{PoM}}}}$. The verification of the proof is successful iff all equations above are true *and* $\boldsymbol{\omega}$ contains all significant characters for f_b.

Proof of Shuffle (PoS) The proof of correct shuffling **PoS** is based on the proofs from [29,30]. In the following we specify the proof to work with Pedersen commitments instead of ElGamal ciphertexts. Note that indices for commitments C and C' run from 1 to n and index ranges in the following change frequently.

1. In the first move, the client (prover) builds a permutation matrix and commits to it. First he chooses random $A'_j \in_R \mathbb{Z}_q^*$ for $j \in [-4, n]$. Let A_{ij} denote a matrix with $i \in [-4, n]$ and $j \in [0, n]$, i.e. of size $(n+5) \times (n+1)$, such that a $n \times n$ sub-matrix of A_{ij} is the permutation matrix (built from permutation ϕ). Further, let ϕ^{-1} be the inverse shuffling function. This allows us to write the shuffle as $C'_i = \prod_{j=0}^{n} C_j^{A_{ji}} = C_{\kappa_i} h^{r'_{\kappa_i}}$ with $C_0 = h$ and $\kappa_i = \phi^{-1}(i)$ for $i \in [1, n]$. The matrix A_{ij} is defined with $A_{w0} \in_R \mathbb{Z}_q^*, A_{-1v} \in_R \mathbb{Z}_q^*$ and $A_{0v} = r'_{\phi(v)}$ for

$w \in [-4, n]$ and $v \in [1, n]$. The remaining values in A_{ij} are computed as follows for $v \in [1, n]$:

$$- \; A_{-2v} = \sum_{j=1}^{n} 3A_{j0}^2 A_{jv}; \quad A_{-3v} = \sum_{j=1}^{n} 3A_{j0}A_{jv}; \quad A_{-4v} = \sum_{j=1}^{n} 2A_{j0}A_{jv}$$

After generating A_{ij} the client commits to it in $(C'_0, \tilde{f}, \boldsymbol{f'}, w, \tilde{w})$ for $\boldsymbol{f'} = \{f'_v\}$ with $v \in [0, n]$:

$$- \; f'_v = \prod_{j=-4}^{n} f_j^{A_{jv}}; \quad \tilde{f} = \prod_{j=-4}^{n} f_j^{A'_j}; \quad \tilde{w} = \sum_{j=1}^{n} A_{j0}^2 - A_{-40}$$

$$- \; C'_0 = g^{\sum_{j=1}^{n} \pi_j A_{j0}} h^{A_{00} + \sum_{j=1}^{n} r_j A_{j0}}; \quad w = \sum_{j=1}^{n} A_{j0}^3 - A_{-20} - A'_{-3} \qquad (2)$$

Note that $C'_0 = \prod_{j=0}^{n} C_j^{A_{j0}} = h^{A_{00}} \prod_{j=1}^{n} C_j^{A_{j0}}$, but Eq. 2 saves $n-1$ exponentiations. The output is then created as $\mathtt{Co_{PoS}} = g^{H(\{C_i\}, \{C'_{\phi(i)}\}, C'_0, \tilde{f}, \boldsymbol{f'}, w, \tilde{w})} h^{r_{\mathtt{Co_{PoS}}}}$ with $r_{\mathtt{Co_{PoS}}} \in_R \mathbb{Z}_q$.

2. When receiving $\mathtt{Co_{PoS}}$ the server chooses $\boldsymbol{c} = \{c_v\}$ with $c_v \in_R \mathbb{Z}_q^*$ for $v \in [1, n]$ and sets $\mathtt{Ch_{PoS}} = \boldsymbol{c}$.

3. After receiving challenges $\boldsymbol{c}$ from the server, the client computes the following verification values $(\boldsymbol{s}, \boldsymbol{s'})$ for $\boldsymbol{s} = \{s_v\}$ and $\boldsymbol{s'} = \{s'_v\}$ with $v \in [-4, n]$ and $c_0 = 1$:

$$s_v = \sum_{j=0}^{n} A_{vj} c_j; \qquad\qquad s'_v = A'_v + \sum_{j=1}^{n} A_{vj} c_j^2$$

The client sets $\mathtt{Rs_{PoS}1} = g^{H(\boldsymbol{s}, \boldsymbol{s'})} h^{r_{\mathtt{Rs_{PoS}}}}$ with $r_{\mathtt{Rs_{PoS}}} \in_R \mathbb{Z}_q$.

4. Eventually, the client sends the decommitment message to the server

$$\mathtt{Rs_{PoS}2} = (C'_0, \tilde{f}, \boldsymbol{f'}, w, \tilde{w}, \boldsymbol{s}, \boldsymbol{s'}, r_{\mathtt{Co_{PoS}}}, r_{\mathtt{Rs_{PoS}}}).$$

Note that $\{C_i\}$ and $\{C'_{\phi(i)}\}$ are omitted here as they are part of $\mathtt{Rs_{PoC}2}$, $\mathtt{Rs_{PoM}2}$ respectively, already. If this proof is used stand-alone, those values have to be added to $\mathtt{Rs_{PoS}2}$.

$\mathtt{Rs_{PoS}1}$ and $\mathtt{Rs_{PoS}2}$ form together $\mathtt{Rs_{PoS}}$. The server verifies now that the correctness of the commitments $\mathtt{Co_{PoS}} \stackrel{?}{=} g^{H(\{C_i\}, \{C'_{\phi(i)}\}, C'_0, \tilde{f}, \boldsymbol{f'}, w, \tilde{w})} h^{r_{\mathtt{Co_{PoS}}}}$ and $\mathtt{Rs_{PoS}1} \stackrel{?}{=} g^{H(\boldsymbol{s}, \boldsymbol{s'})} h^{r_{\mathtt{Rs_{PoS}}}}$, and that the following equations hold for a randomly chosen $\alpha \in_R \mathbb{Z}_q^*$ and $C_0 = h$:

$$- \; \prod_{v=-4}^{n} f_v^{s_v + \alpha s'_v} \stackrel{?}{=} f'_0 \tilde{f}^{\alpha} \prod_{j=1}^{n} f_j'^{c_j + \alpha c_j^2}; \quad \prod_{v=0}^{n} C_v^{s_v} \stackrel{?}{=} \prod_{j=0}^{n} C_j'^{c_j};$$

$$- \; \sum_{j=1}^{n} (s_j^3 - c_j^3) \stackrel{?}{=} s_{-2} + s'_{-3} + w; \quad \sum_{j=1}^{n} (s_j^2 - c_j^2) \stackrel{?}{=} s_{-4} + \tilde{w}$$

The server accepts the proof if and only if all those verifications succeed. This concludes the proof of correct shuffling.

Phase III – Share verification To verify that the client used the same password pw and shares $\mathfrak{s}_0, \mathfrak{s}_1$ with both servers S_0 and S_1, the servers compute the commitment $\mathfrak{D}'_b$ from the share commitment $\mathfrak{C}_b$ and their share $\mathfrak{s}_{1-b}$, and exchange it. Comparing $\mathfrak{D}'_b$ with the value $\mathfrak{D}_b$ received from the client, the server verifies share correctness. This concludes the 2BPR protocol and each server S_b stores $(\mathcal{C}, S_{1-b}, \mathfrak{s}_b)$ if all checks were successful.

4.3 Security Analysis

We show that our 2BPR protocol is secure in the model from Section 3.1 and thus offers policy compliance and password blindness. For space limitations we include only the proofs of Theorems 1 and 2 note that **PoM** and **PoC** protocols are standard concurrent ZK proofs and **PoS** is a slightly modified concurrent ZK proof from [29,30].

Lemma 1. *The* **PoC** *protocol from Section 4.2 is a concurrent zero-knowledge proof if the discrete logarithm problem in the used group G is hard and $H : \{0,1\}^* \mapsto \mathbb{Z}_q$ is a collision resistant hash function.*

Lemma 2. *The* **PoM** *protocol from Section 4.2 is a concurrent zero-knowledge proof if the discrete logarithm problem in the used group G is hard and $H : \{0,1\}^* \mapsto \mathbb{Z}_q$ is a collision resistant hash function.*

Lemma 3 ([29,30]). *The* **PoS** *protocol from Section 4.2 is a concurrent zero-knowledge proof of knowledge of shuffling ϕ if the discrete logarithm problem in the used group G is hard and $H : \{0,1\}^* \mapsto \mathbb{Z}_q$ is a collision resistant hash function.*

Theorem 1. *If G is a DL-hard group of prime-order q with generators g and h, and H a collision resistant hash function, the construction in Figure 1 provides policy compliance according to Definition 2.*

Proof. We show how to build a successful attacker on the soundness of **PoC**, **PoM** and **PoS** using a successful attacker against policy compliance who has access to Setup and $\mathsf{Send}_{\mathcal{C}}$ oracles.

$\mathbf{Game_0}$: This game corresponds to the correct execution of the protocol.
$\mathbf{Game_1}$: In this game we change how $\mathsf{Send}_{\mathcal{C}}(\mathcal{C}_i, S_{b,j}, m)$ queries are answered. If m is parsed as $(\mathtt{Co}_{\mathbf{PoM}}, \mathtt{Co}_{\mathbf{PoC}}, \mathtt{Co}_{\mathbf{PoS}})$ the $\mathtt{Co}_{\mathbf{PoM}}$ is used by the challenger as output to the **PoM** verifier who returns challenge $\mathtt{Ch}_{\mathbf{PoM}}$ which is then returned in response to the $\mathsf{Send}_{\mathcal{C}}$ query (other challenges are generated at random). If m is parsed as $(\mathtt{Rs}_{\mathbf{PoM}1}, \mathtt{Rs}_{\mathbf{PoC}1}, \mathtt{Rs}_{\mathbf{PoS}1})$ or $(\mathtt{Rs}_{\mathbf{PoM}2}, \mathtt{Rs}_{\mathbf{PoC}2}, \mathtt{Rs}_{\mathbf{PoS}2})$ and the first $\mathsf{Send}_{\mathcal{C}}$ query from that session was forwarded to the verifier then $\mathtt{Rs}_{\mathbf{PoM}1}$, $\mathtt{Rs}_{\mathbf{PoM}2}$ respectively, is used as output to the **PoM** verifier. It is easy to see that the challenger breaks soundness of **PoM** if the adversary uses a password $\mathsf{pw} \notin \mathcal{D}_f$ and **PoM** verifies successfully. We can therefore assume for the remaining games that $\mathsf{pw} \in \mathcal{D}_f$.

Game$_2$: In this game we introduce another change to the processing of $\mathsf{Send}_{\mathcal{C}}(\mathcal{C}_i, S_{b,j}, m)$ queries. If m is parsed as $(\mathtt{Co}_{\mathbf{PoM}}, \mathtt{Co}_{\mathbf{PoC}}, \mathtt{Co}_{\mathbf{PoS}})$ then $\mathtt{Co}_{\mathbf{PoC}}$ is used by the challenger as output to the **PoC** verifier who returns challenge $\mathtt{Ch}_{\mathbf{PoC}}$ that is then used as response to the $\mathsf{Send}_{\mathcal{C}}$ query (other challenges are generated at random). If m is parsed as $(\mathtt{Rs}_{\mathbf{PoM}1}, \mathtt{Rs}_{\mathbf{PoC}1}, \mathtt{Rs}_{\mathbf{PoS}1})$ or $(\mathtt{Rs}_{\mathbf{PoM}2}, \mathtt{Rs}_{\mathbf{PoC}2}, \mathtt{Rs}_{\mathbf{PoS}2})$ and the first $\mathsf{Send}_{\mathcal{C}}$ query from that session was forwarded to the verifier then $\mathtt{Rs}_{\mathbf{PoC}1}$, $\mathtt{Rs}_{\mathbf{PoC}2}$ respectively, is used as output to the **PoC** verifier. It is easy to see that the challenger breaks soundness of **PoC** if $\mathfrak{s}_0 + \mathfrak{s}_1 \neq \pi$, i.e. the password share $\mathfrak{s}_b$ can not be used with a second share $\mathfrak{s}_{1-b}$ to rebuild the password π committed to in $\boldsymbol{C}$, i.e. $\sum_i b^i \pi_i \neq \pi$. Observe further that the second share $\mathfrak{s}_{1-b}$ has to be stored on server S_{1-b}, i.e. the attacker has not performed the set-up with S_b and S_{1-b} with shares that do not combine to the same encoded password π. Otherwise we can break the binding property of Pedersen commitments. In particular, the attacker has to generate commitments $\mathfrak{C}_0, \mathfrak{C}_1, \mathfrak{D}_0$ and $\mathfrak{D}_1$ such that $\mathfrak{C}_0 g^{\mathfrak{s}_1} = \mathfrak{D}_0$ or $\mathfrak{C}_1 g^{\mathfrak{s}_0} = \mathfrak{D}_1$. We can therefore for the remaining games that the password share $\mathfrak{s}_b$ received by server S_b can be combined with the second share $\mathfrak{s}_{1-b}$ of server S_{1-b} to an encoded password π with according character commitments C_i.
Game$_3$: In this game we change once more how $\mathsf{Send}_{\mathcal{C}}(\mathcal{C}_i, S_{b,j}, m)$ queries are answered. If m is parsed as $(\mathtt{Co}_{\mathbf{PoM}}, \mathtt{Co}_{\mathbf{PoC}}, \mathtt{Co}_{\mathbf{PoS}})$ then $\mathtt{Co}_{\mathbf{PoS}}$ is used by the challenger as output to the **PoS** verifier who returns challenge $\mathtt{Ch}_{\mathbf{PoS}}$ that is then out in response to the $\mathsf{Send}_{\mathcal{C}}$ query (other challenges are generated at random). If m from the adversary is parsed as $(\mathtt{Rs}_{\mathbf{PoM}1}, \mathtt{Rs}_{\mathbf{PoC}1}, \mathtt{Rs}_{\mathbf{PoS}1})$ or $(\mathtt{Rs}_{\mathbf{PoM}2}, \mathtt{Rs}_{\mathbf{PoC}2}, \mathtt{Rs}_{\mathbf{PoS}2})$ and the first $\mathsf{Send}_{\mathcal{C}}$ query from that session was forwarded to the verifier then $\mathtt{Rs}_{\mathbf{PoS}1}$, $\mathtt{Rs}_{\mathbf{PoS}2}$ respectively, is used as the output to the **PoS** verifier. In this case if the attacker is rewindable, the challenger can act as a knowledge extractor for **PoS**. In particular, we can extract shuffling function ϕ and re-randomiser $\{r'_i\}$ to break soundness of **PoS**. This implies that $\boldsymbol{C}'$ is a correct shuffle of $\boldsymbol{C}$. We conclude the proof by observing that the password shares stored on both servers can be combined to a policy compliant password.

Theorem 2. *If G is a DL-hard group of prime-order q with generators g and h, and H a collision resistant hash function, the construction in Figure 1 provides password blindness according to Definition 3.*

Proof. We prove this theorem through a sequence of games. In the last game simulated interactions between servers and clients are simulated and password independent, thus requiring the attacker to perform a random guess of the bit b.

Game$_0$: This is the correct execution of the protocol.
Game$_1$: The challenger computes $\mathtt{crs}$ with the knowledge of the trapdoor $\tau = \log_g(h)$.
Game$_2$: The challenger simulates the proofs **PoC**, **PoM** and **PoS** and messages exchanged between the servers as part of the $\mathsf{Execute}$ oracle but stores two correct shares on the servers to allow consistency if servers become corrupted. Since at least one server must remain uncorrupted the probability difference between both games is negligible due to the zero-knowledge property of the proofs.

Game$_3$: This game modifies Send_S and Send_{SS} responses if the second participating server is uncorrupted by simulating zero-knowledge proofs and answering Send_{SS} queries using $\mathfrak{D}'_b = \mathfrak{D}_b$. To guarantee consistency in case of corruptions the challenger still stores appropriate shares. The probability difference between both games is negligible due to the zero-knowledge property of the proofs. Since all proofs are password-independent and Pedersen commitments offer unconditional hiding the attacker can only win by guessing b.

5 Performance and Use with 2PAKE/2PASS protocols

An unoptimised prototype of the 2BPR protocol from Section 4 was implemented over the NIST P-192 elliptic curve [31] in Python using the Charm framework [32] to estimate the performance. The tests (completed on a laptop with an Intel Core Duo P8600 at 2.40GHz for both client and server) underline the claim that the protocol is practical. For instance, for a password of length 10 and policies $(dl, 5)$ and $(ds, 7)$ computations take 1.4 seconds on the client and 0.68 seconds on each server. The overall computing time for a password of length 10 was 2.76 sec and increased to 6.34 seconds for a password of length 20. Also note that the execution can be parallelised if the client performs the proofs with S_0 and S_1 at the same time. The source code is available from `https://goo.gl/XfIZtn`.

Application to existing 2PAKE/2PASS protocols Our 2BPR protocol can be used to register passwords for 2PAKE and 2PASS protocols that adopt additive password sharing in $\mathbb{Z}_q$ or multiplicative sharing in G. This includes 2PAKE protocols from [15,25] for which no password registration procedures were addressed. Integration of 2BPR into 2PASS protocols is more involved since password registration is considered to be part of the 2PASS protocol during the secret sharing phase. 2PASS protocols in general can be divided in two stages: password and secret registration/sharing and secret reconstruction. While the approach from [16] and subsequent works [33,34,18] do not actually share the password and could therefore use other means to verify policy compliance of a passwords used, the UC-secure 2PASS protocol from [17] uses multiplicative password sharing in G. In order to use our 2BPR protocol withing the setup procedure of [17] we can redefine the encoded password to g^{π} with $\pi \leftarrow \mathtt{PWDtoINT}(\mathrm{pw})$ such that shares are computed as $g^{\pi} = g^{\mathfrak{s}_0} g^{\mathfrak{s}_1}$. The first message (step 1) from the setup protocol in [17] can piggyback the first 2BPR protocol message. The subsequent three messages between the client and each server are performed between step 1 and step 2, while the inter-server communication can be piggybacked on step 2 and step 3. In addition to checking correctness of shares in the setup of [17] the servers can now verify the 2BPR proofs to check policy compliance. This would adds three flows to the setup protocol of [17].

6 Conclusion

In this work we introduced the notion of two-server blind password registration (2BPR), which is a solution for secure registration of policy-compliant, user-selected passwords for 2PAKE/2PASS protocols where each server is supposed to learn only its own share of the password and whether the combined password is conform with his password policy. Our efficient 2BPR protocol can be used to register 2PAKE/2PASS passwords satisfying server-chosen policies over the alphabet of all 94 printable ASCII characters.

References

1. Reuters, "Trove of Adobe user data found on Web after breach: security firm," http://goo.gl/IC4lu8, 2014, Accessed: 01/04/2015.
2. Nik Cubrilovic, "RockYou Hack: From Bad To Worse," http://goo.gl/AF5ZDM, 2014, Accessed: 01/04/2015.
3. Thomson Reuters, "Microsoft India store down after hackers take user data," http://goo.gl/T7puD1, 2014, Accessed: 01/04/2015.
4. Dan Goodin, "Hack of Cupid Media dating website exposes 42 million plaintext passwords," http://goo.gl/ImLE1C, 2014, Accessed: 01/04/2015.
5. hashcat, "hashcat - advanced password recovery," http://hashcat.net/, 2014, Accessed: 01/04/2015.
6. Openwall, "John the Ripper password cracker," http://www.openwall.com/john/, 2014, Accessed: 01/04/2015.
7. J. Ma, W. Yang, M. Luo, and N. Li, "A Study of Probabilistic Password Models," in *IEEE S&P*, 2014, pp. 689–704.
8. M. Dürmuth and T. Kranz, "On Password Guessing with GPUs and FPGAs," in *PASSWORDS'14*, 2014, pp. 19–38.
9. M. Dell'Amico, P. Michiardi, and Y. Roudier, "Password Strength: An Empirical Analysis," in *INFOCOM*. IEEE, 2010, pp. 983–991.
10. J. Bonneau, "The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords," in *IEEE S & P*. IEEE Computer Society, 2012, pp. 538–552.
11. W. Ford and B. S. K. Jr., "Server-assisted generation of a strong secret from a password," in *WETICE*. IEEE, 2000, pp. 176–180.
12. P. D. MacKenzie, T. Shrimpton, and M. Jakobsson, "Threshold password-authenticated key exchange," in *CRYPTO'02*, ser. LNCS, vol. 2442. Springer, 2002, pp. 385–400.
13. J. G. Brainard, A. Juels, B. Kaliski, and M. Szydlo, "A New Two-Server Approach for Authentication with Short Secrets," in *USENIX Security Symposium*. USENIX Association, 2003.
14. M. Szydlo and B. S. K. Jr., "Proofs for Two-Server Password Authentication," in *CT-RSA'05*, ser. LNCS, vol. 3376. Springer, 2005, pp. 227–244.
15. J. Katz, P. MacKenzie, G. Taban, and V. Gligor, "Two-server password-only authenticated key exchange," in *ACNS'05*, ser. LNCS, vol. 3531. Springer, 2005, pp. 1–16.
16. A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu, "Password-protected secret sharing," in *CCS'11*. ACM, 2011, pp. 433–444.

17. J. Camenisch, A. Lysyanskaya, and G. Neven, "Practical yet universally composable two-server password-authenticated secret sharing," in *CCS'12*. ACM, 2012, pp. 525–536.
18. S. Jarecki, A. Kiayias, and H. Krawczyk, "Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model," in *ASIACRYPT'14*, ser. LNCS, vol. 8874. Springer, 2014, pp. 233–253. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-45608-8_13
19. F. Kiefer and M. Manulis, "Zero-Knowledge Password Policy Checks and Verifier-Based PAKE," in *ESORICS'14*, ser. LNCS, vol. 8713. Springer, 2014, pp. 295–312.
20. T. P. Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," in *CRYPTO'91*, ser. LNCS, vol. 576. Springer, 1991, pp. 129–140.
21. I. Damgård, "Efficient concurrent zero-knowledge in the auxiliary string model," in *EUROCRYPT'00*, ser. LNCS, vol. 1807. Springer, 2000, pp. 418–430.
22. S. Jarecki and A. Lysyanskaya, "Adaptively secure threshold cryptography: Introducing concurrency, removing erasures," in *EUROCRYPT'00*, ser. LNCS, vol. 1807. Springer, 2000, pp. 221–242.
23. Y. Yang, R. H. Deng, and F. Bao, "A Practical Password-Based Two-Server Authentication and Key Exchange System," *IEEE Trans. Dependable Sec. Comput.*, vol. 3, no. 2, pp. 105–114, 2006.
24. H. Jin, D. Wong, and Y. Xu, "An efficient password-only two-server authenticated key exchange system," *Information and Communications Security*, pp. 44–56, 2007.
25. F. Kiefer and M. Manulis, "Distributed Smooth Projective Hashing and Its Application to Two-Server Password Authenticated Key Exchange," in *ACNS'14*, ser. LNCS, vol. 8479. Springer, 2014, pp. 199–216.
26. T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), Internet Engineering Task Force, Aug. 2008, updated by RFCs 5746, 5878, 6176, 7465. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt
27. T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the Security of TLS-DHE in the Standard Model," in *CRYPTO'12*, ser. LNCS, vol. 7417. Springer, 2012, pp. 273–293.
28. H. Krawczyk, K. G. Paterson, and H. Wee, "On the Security of the TLS Protocol: A Systematic Analysis," in *CRYPTO'13*, ser. LNCS, vol. 8042. Springer, 2013, pp. 429–448.
29. J. Furukawa and K. Sako, "An Efficient Scheme for Proving a Shuffle," in *CRYPTO'01*, ser. LNCS, vol. 2139. Springer, 2001, pp. 368–387.
30. J. Furukawa, "Efficient and Verifiable Shuffling and Shuffle-Decryption," *IEICE Transactions*, vol. 88-A, no. 1, pp. 172–188, 2005.
31. NIST, "National Institute of Standards and Technology. Recommended elliptic curves for federal government use," http://goo.gl/M1q10h, 1999.
32. J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013. [Online]. Available: http://dx.doi.org/10.1007/s13389-013-0057-3
33. I. Pryvalov and A. Kate, "Introducing Fault Tolerance into Threshold Password-Authenticated Key Exchange," Cryptology ePrint Archive, Report 2014/247, 2014. [Online]. Available: http://eprint.iacr.org/2014/247
34. J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven, "Memento: How to reconstruct your secrets from a single password in a hostile environment," in *CRYPTO'14*, ser. LNCS, vol. 8617. Springer, 2014, pp. 256–275.