Blind Password Registration for Verifier-based PAKE

Franziskus Kiefer
Mozilla Inc.
Berlin, Germany
mail@franziskuskiefer.de

Mark Manulis
Surrey Centre for Cyber Security
Department of Computer Science
University of Surrey, UK
mark@manulis.eu

ABSTRACT

We propose *Blind Password Registration (BPR)*, a new class of cryptographic protocols that is instrumental for secure registration of client passwords at remote servers with additional protection against unwitting password disclosures on the server side that may occur due to the lack of the state-of-the-art password protection mechanisms implemented by the server or due to common server-compromise attacks. The *dictionary attack resistance* property of BPR protocols guarantees that the only information available to the server during and after the execution of the protocol cannot be used to reveal the client password without performing an offline dictionary attack on a password verifier (e.g. salted hash value) that is stored by the server at the end of the protocol. In particular, at no point in time the server is supposed to work with plain passwords. Our BPR model allows servers to enforce *password policies* and the requirement on the client to obey them during the execution of the BPR protocol is covered by the *policy compliance* property.

We construct an efficient BPR protocol in the standard model for ASCII-based password policies using some techniques underlying the recently introduced Zero-Knowledge Password Policy Checks (ZKPPC). However, we do not rely on the full power of costly ZKPPC proofs and in fact show that BPR protocols can be modelled and realised simpler and significantly faster (as supported by our implementation) without using them as a building block. Our BPR protocol can directly be used to replace ZKPPC-based registration procedure for existing VPAKE protocols.

1. INTRODUCTION

Cryptographic password authentication, including password authenticated key exchange (PAKE) protocols and their variants, remains an important research topic since the 1990s [8, 22, 7], driven by the wide use of passwords and continuous damage from password compromise attacks. A rich variety of challenges related to the modelling of PAKE protocols [7, 12, 4, 13, 18], efficient design and security analysis [23, 3, 24, 10], possible deployment in practice [1, 2, 19, 20, 38, 28] along with the development of metrics for password strength evaluation [32, 39, 27, 30] and usability improvements for password authentication [14, 21, 26] have been addressed so far, with only little attention paid to the remote registration of passwords, which is the initial step of any remote password-based protocol and perhaps the most crucial one in terms of security. In many cryptographic password-based protocols this step is often omitted using the assumption that passwords are set up in a secure way and known to the parties prior to the execution of the protocol.

To see the importance of password registration consider the current approach for remote registration of client passwords on the Web: the client establishes a server-authenticated confidential channel (e.g., using TLS) over which the password is sent to the server and then (securely) stored in the password database. In order to offer better protection against compromised password databases servers are supposed to store only the randomised password hash and the random salt that was used. In later sessions the client authenticates itself with the password whereas the server uses stored password hash and salt to perform the check. This concept resembles what is called a Verifier-based PAKE (VPAKE) protocol [9, 18, 11] (aka. asymmetric [7] or augmented PAKE [9]). A large number of break-ins into server databases makes VPAKE a particularly valuable concept for remote password-based authentication since in order to recover the actual client password the attacker must perform a costly dictionary attack on its password hash. However, the increasing number of successful password leaks [33, 15, 31] suggests that many servers do not apply randomised password hashing at all. For this reason, remote password registration should ideally be performed without the need for the client to send its password to the server; in other words, without trusting the server to securely process and store client passwords.

This was recently taken as a motivation in [25] to come up with the concept of *blind* password registration for VPAKE protocols by which the server only learns the randomised password hash and the random salt but not the actual password. The major challenge addressed there was to find a solution for blind registration of password verifiers for VPAKE that can be proven to satisfy the server's *policy* on the format of passwords (e.g. minimal password length, inclusion of characters of different types, etc.) during the registration procedure, yet without disclosing those passwords in clear. This was realised using new Zero-Knowledge

AsiaPKC'16, May 30-29 2016, Xi'an, China

ISBN 978-1-4503-4286-5/16/05...$15.00

DOI: http://dx.doi.org/10.1145/2898420.2898424

Password Policy Checks (ZKPPC) that served in [25] as an intermediate registration step and allowed the client to prove policy-compliance (soundness) of the chosen password without leaking any information about the password (zero-knowledge) to the server. The security of ZKPPC was defined to prevent malicious clients from registering passwords that do not comply with the server's policy and to prevent malicious servers from learning any information about passwords from the ZKPPC protocol execution. The security model from [25] focused only on the ZKPPC proofs and did not model the overall security of the blind password registration procedure, where the client first performs the ZKPPC proof and then discloses to the server (partial) randomness that was used to compute the randomised password hash and that remained secret during the ZKPPC execution. Obviously, the zero-knowledge property of ZKPPC is too strong since it also protects randomness that is disclosed later and so the blind password registration procedure, if modelled as a stand-alone protocol, can possibly be realised using weaker (and more efficient) primitives than ZKPPC proofs. In addition, a concrete ZKPPC construction proposed in [25] for password strings consisting of printable ASCII characters is too inefficient for practical use due to the use of set membership proofs on committed values over sets of size $O(n_{\texttt{max}})$ where the maximum password length $n_{\texttt{max}}$ is a fixed global parameter for the ZKPPC protocol. Therefore, while [25] made the first step towards blind registration of policy-compliant VPAKE passwords, their proposed modelling approach and solutions based on ZKPPC leave space for further improvements.

In this work we take Blind Password Registration (BPR) to a new level: first, we define a security model for stand-alone BPR protocols without resorting to any intermediate building blocks like ZKPPC and model two main requirements — *policy compliance* and *dictionary attack resistance* — that a secure BPR protocol must fulfil (cf. Section 3). In particular, our dictionary attack resistance, which basically says that the attacker must not be able to retrieve the client password significantly faster from the information obtained out of BPR executions than by trying each password individually against the password hash and the salt, is a more natural requirement for BPR protocols than the zero-knowledge requirement formulated for ZKPPC proofs. Second, we present a truly efficient BPR protocol for policy-compliant VPAKE passwords in Section 4 where $n_{\texttt{max}}$ is no longer needed as a parameter and the underlying set membership proofs for committed password characters proceed with respect to sets of size $O(1)$. This improvement comes from the use of an efficient shuffling proof from [16] that is used to link proofs of set membership of single characters to the policy compliance of the combined password without the need to inflate character sets to the maximum password length $n_{\texttt{max}}$. (While more efficient shuffling proofs exist, e.g., [6], they have higher round complexity and their advantage becomes noticeable only for larger sets than those used to represent ASCII-based passwords.) By performing set membership proofs over sets of maximum 94 characters we eliminate the factor $n_{\texttt{max}}$ from the protocol performance but introduce a smaller cost of the shuffling proof in the password length n, which seems unavoidable in this approach if the protocol should hide any relation between the policy and the password other than its compliance. The proposed BPR protocol is compatible with the recent VPAKE protocol from [25] such that both protocols together can be seen as a framework for remote password registration and authentication where the client password remains hidden from the server.

2. PRELIMINARIES

We recall building blocks we use in our construction and give preliminaries. We start with definitions of passwords, dictionaries, and policies from [25]. Note that we incorporate small but substantial modifications to some definitions. We use bold characters $\boldsymbol{C} = \{C_i\}$ to denote vectors of elements.

2.1 Passwords, Dictionaries and Policies

ASCII-based passwords.

The *character mapping* function $\texttt{CHRtoINT} : \Sigma \mapsto \mathbb{Z}_{95}$ maps any of 94 printable ASCII characters $c \in \Sigma$ to an integer in $\mathbb{Z}_{95}$ using its decimal ASCII code $\texttt{ASCII}(c)$, namely $\texttt{CHRtoINT}(c) := \texttt{ASCII}(c) - 32$ for all $33 \leq \texttt{ASCII}(c) \leq 126$. Let $n = |pw|$ denote the password length. The *password mapping* function $\texttt{PWDtoINT} : \Sigma^n \mapsto \mathbb{Z}_{b^n}$ with *shift base* $b \in \mathbb{N}$ maps any password string $pw = (c_0, \ldots, c_{n-1}) \in \Sigma^n$ to an integer in the set $\mathbb{Z}_{b^n}$, namely

$$\texttt{PWDtoINT}(pw) := \sum_{i=0}^{n-1} b^i \texttt{CHRtoINT}(c_i).$$

We specify our protocol using shift base b as a parameter and give concrete values in Section 4.3 where the implementation and performance of the protocol are discussed. We use pw to denote a password string and $\pi \leftarrow \texttt{PWDtoINT}(pw)$ for its integer value, and write $\pi = \sum_{i=0}^{n-1} b^i \pi_i$ for encoded characters $\pi_i \leftarrow \texttt{ASCII}(c) - 32$. Note that the password encoding $\texttt{PWDtoINT}$ is an injective function that maps every password string pw to a corresponding unique integer π. In particular, $\texttt{PWDtoINT}(pw) \neq \texttt{PWDtoINT}(pw')$ for all password strings $pw, pw' \in \{0,1\}^*$ with $pw \neq pw'$.

Password policies.

A *password policy* is defined as a tuple $f = (R, n_{\texttt{min}})$, where R is a policy expression over $\Sigma = \{d, u, l, s\}$, where d denotes digits, u upper case letters, l lower case letters, s symbols, and $n_{\texttt{min}}$ defines the minimum length of a password. Note that, unlike [25], we do not specify an upper limit on the password length. A policy expression R over Σ is a simplified regular expression that only specifies the sets necessary for a string to fulfil the expression. In particular, it specifies the minimum number of occurrences of elements from Σ in the password string, e.g., $R = dl$ requires pw to have at least one digit and one lower case letter, and $R = ssd$ requires pw to have at least two symbols and one digit. We write $f(pw) = \texttt{true}$ to indicate that the policy is satisfied by the password string pw. Further, a character $c_i \in pw$ is called *significant* if it is necessary to fulfil a policy expression R and say the according set $R_j \in R$ is the according *significant set*. For every $R_j \in R$ the first occurrence of a character $c_i \in R_j$ is considered significant. Note that Σ, and thus d, u, l and s, in this work can refer to the set of encoded characters, or the set of ASCII characters, depending on the context.

Dictionaries.

A password dictionary is denoted by $\mathcal{D}$ and contains all possible combinations of ASCII characters. For a more precise security analysis we also define the following subsets: a dictionary containing all policy compliant passwords is denoted by $\mathcal{D}_f = \{pw \in \mathcal{D} : f(pw) = \texttt{true}\}$ and its subsets with passwords of length $l \in \mathbb{N}$ by $\mathcal{D}_{f,l}$, i.e. $\mathcal{D}_{f,l} = \{pw \in \mathcal{D} : f(pw) = \texttt{true} \wedge |pw| = l\}$.

2.2 Password Distributions and Min-Entropy

Intuitively, a password hashing scheme should be considered secure (and a BPR protocol resistant to dictionary attacks) if an attacker can not retrieve the password from its hash more efficiently than by performing a brute-force attack over the dictionary. Therefore, security definitions for password hashing and dictionary attack resistance rely on the notion of min-entropy β.

The dictionary $\mathcal{D}$, from which the passwords are chosen, has min-entropy β such that efficient sampling of the dictionary allows retrieving the password from its hash with probability in β. Although passwords in cryptographic research are often assumed to be uniformly at random distributed low-entropy secrets, we consider a somewhat more realistic password model. In particular, we use passwords as character strings where the distribution of characters depends on the used character sets ω, character positions and the password string itself. We thus use a definition of password min-entropy commonly used in password security research [37, 27, 30], which captures the difficulty of brute-force attacks on passwords chosen from certain dictionaries. As discussed in [37, 27], this definition can capture many realistic password creation models. Let $\mathbb{D}_\omega$ denote the probability distribution in password pw of characters from a character set $\omega \in \{\Sigma, d, u, l, s\}$. Min-entropy for $pw = (c_0, \ldots, c_{n-1})$ is then defined according to Shannon [36] as

$$\beta_{\mathcal{D}_{f,l}} = -\max_{pw \in \mathcal{D}_{f,l}} \sum_{i=0}^{n-1} [\mathbb{D}_\Sigma(c_i) \lg(\mathbb{D}_\Sigma(c_i))].$$

Note that definitions for min-entropy of $\mathcal{D}$ and $\mathcal{D}_f$ are equivalent to the definition for $\mathcal{D}_{f,l}$. While this may be surprising at first glance, one has to consider that while the policy restricts the character space, it does not restrict the positions where these characters appear, i.e. an adversary cannot exclude any characters at any position. (Note that we exclude the special case where every character in a password is significant and the policy expression R does not use all four available character sets.)

2.3 Building Blocks

In this section we define building blocks and give definitions that are used in the remainder of this work. Due to space limitations more well known definitions can be found in the full version of this work.

2.3.1 Commitments

Pedersen commitments [35].

The Pedersen commitment scheme [35] is perfectly hiding and additively homomorphic. Its $\texttt{CSetup}(\lambda)$ algorithm outputs (q, g, h, λ), where g and h are generators of a cyclic group G of prime order q of length λ and the discrete logarithm of h with respect to g is unknown. $\texttt{Com}(x; r)$ for $x, r \in \mathbb{Z}_q^*$ outputs commitment $C = g^x h^r$ and decommitment $d = (x, r)$. $\texttt{Open}(C, d)$ returns x iff $C = g^x h^r$.

2.3.2 Password Hashing

A *password hashing scheme* $\mathcal{H}$ consists of the following five algorithms:

- $\texttt{PSetup}(\lambda)$ generates password hashing parameters $\texttt{p}_\texttt{P}$. These parameters contain implicit descriptions of random salt spaces $\mathbb{S}_P$ and $\mathbb{S}_H$.
- $\texttt{PPHSalt}(\texttt{p}_\texttt{P})$ generates a random pre-hash salt $s_P \in_R \mathbb{S}_P$.
- $\texttt{PPreHash}(\texttt{p}_\texttt{P}, pw, s_P)$ outputs the pre-hash value P.
- $\texttt{PHSalt}(\texttt{p}_\texttt{P})$ generates a random hash salt $s_H \in_R \mathbb{S}_H$.
- $\texttt{PHash}(\texttt{p}_\texttt{P}, P, s_P, s_H)$ outputs the hash value H.

We write $H \leftarrow \texttt{Hash}_\texttt{P}(pw, r)$ to denote $H \leftarrow \texttt{PHash}(\texttt{p}_\texttt{P}, P, s_P, s_H)$ with $P \leftarrow \texttt{PPreHash}(\texttt{p}_\texttt{P}, pw, s_P)$, where $r = (s_P, s_H)$ combines the randomness used in $\texttt{PHash}$ and $\texttt{PPreHash}$. See Appendix [11, 25] for security definitions of password hashing.

Password hashing from Pedersen commitments [25].

We use the following algebraic password-hashing scheme from [25]: $\texttt{PSetup}(\lambda)$ generates $\texttt{p}_\texttt{P} = (q, g, h, \lambda)$ where g, h are independent generators of a cyclic group G of prime order q of length λ. $\texttt{PPHSalt}(\texttt{p}_\texttt{P})$ generates a pre-hash salt $s_P \in_R \mathbb{Z}_q^*$. $\texttt{PPreHash}(\texttt{p}_\texttt{P}, \pi, s_P)$ outputs the pre-hash value $P = g^{s_P \pi}$. $\texttt{PHSalt}(\texttt{p}_\texttt{P})$ generates a hash salt $s_H \in_R \mathbb{Z}_q^*$. $\texttt{PHash}(\texttt{p}_\texttt{P}, P, s_P, s_H)$ outputs hash value $H = (H_1, H_2) = (g^{s_P}, P h^{s_H})$.

3. BLIND PASSWORD REGISTRATION

Blind Password Registration (BPR) allows a user to register a password verifier at a VPAKE server and prove that it contains a password that complies with the server's policy without disclosing the password. A BPR protocol is thus executed between a client C and a server S, both holding the server's password policy f. (The policy can be exchanged before the actual protocol with other general information about the registration.) After choosing a policy compliant password pw, C engages in a protocol with S to prove policy compliance of pw, i.e. $f(pw) = \texttt{true}$, and sends a password verifier to the server, which can later be used in VPAKE protocols. VPAKE protocols that can be used with verifiers set-up with our protocol are discussed in [25]. Blind password registration is formally defined as follows.

Definition 1 (Blind Password Registration) *A BPR protocol is executed between a client C and a server S with server's password policy f as a common input. At the end of the protocol the server eventually outputs the password verifier v_C for a policy compliant, client chosen password pw.*

3.1 Security of Blind Password Registration

We consider two security properties for BPR protocols to capture the requirement that the server *learns nothing about the password in the verifier* and that the password verifier v_C belongs to a *policy compliant* password. The first security notion regarding the server, called *Dictionary Attack*

Resistance (DAR), considers a passive attack in which the adversary must not be able to retrieve the password from the password verifier faster than with a brute-force attack over the used dictionary. The second security notion regarding the client, called *Policy Compliance (PC)*, considers an active attack where the adversary plays the role of the client and tries to register a non-compliant password at a server. We propose a game-based security model for BPR protocols over dictionaries $\mathcal{D}_{f,n}$. Recall that a policy dictionary $\mathcal{D}_f$ contains all passwords pw with $f(pw) = \texttt{true}$ and a dictionary $\mathcal{D}_{f,n}$ contains all passwords pw with $f(pw) = \texttt{true}$ and $|pw| = n$. We work in the semi-honest server model where the client can be malicious, but the server is honest in its execution. Note that security of BPR protocols can only be assessed with respect to the used password hashing scheme $\texttt{Hash}_\texttt{P}$ since the attackers ability to recover the password from a compromised server depends on the pre-image resistance of the hashing scheme.

Participants and Parameters.

A BPR protocol is executed between a client C from a universe of clients $\mathcal{C}$ and a server S_f chosen from the universe of servers $\mathcal{S}$. The universe of servers $\mathcal{S}$ contains servers S_f such that for every policy f, there exists a server S_f. Note that we usually omit f and write S instead. Both protocol participants have common inputs, necessary for the execution of the protocol, and the password policy f. Instances of protocol participants C or S are denoted C_i or S_i. Protocol participants without specified role are denoted by P. A client can only register one password with one server, but can register passwords at an arbitrary number of servers. Further, a server only allows a single registration from a client such that any attempt to register a password with a server that already stores a verifier from this client is rejected by the server. The client C is unique and is used as identifier on the server, i.e. as *username* to store alongside the password verifier v_C for later VPAKE executions. An entry (C, v_C) is only stored on the server if the BPR protocol is successful. To interact with protocol participants, the adversary has access to an $\mathsf{Execute}$ and a Send oracle.

- $\mathsf{Execute}(C, S)$ models a passive attack and executes a BPR protocol between new instances of C and S. If there exists a verifier v_C for client C on server S, the oracle aborts. Otherwise, it returns the protocol transcript and the internal state of the server S.
- $\mathsf{Send}(C_i, S_i, m)$ models an active attack and sends message m, allegedly from client instance C_i, to server instance S_i (a new server instance with a unique index i is created if it does not exist yet). If there exists a verifier v_C for client C on server S, the oracle aborts. Otherwise, it returns the server's answer m' if there exists any.

Note that we allow the adversary to register passwords with servers such that we do not require the existence of a client C after a successful registration of (C, v_C) on a server (client identities C are unique but not secret and can therefore be used by the adversary).

Policy compliance is the first natural security property of BPR protocols, requiring that a password set up with a BPR protocol is compliant with the server's policy f. The attacker here plays the role of the client and tries to register a password pw on a server that is *not* policy compliant.

Definition 2 (Policy Compliance) *Let $\mathcal{A}$ denote a PPT adversary with access to* $\mathsf{Execute}$ *and* Send *oracles. The probability that a server instance S_i exists after $\mathcal{A}$ terminated that accepted (C, v_C) with $v_C = \texttt{Hash}_\texttt{P}(pw; r)$ and $f(pw) = \texttt{false}$ is negligible in λ.*

To model the second security property, *Dictionary Attack Resistance* (DAR), we define another oracle, which models the offline dictionary attack on the password verifier v_C. DAR models server compromise and requires that it is impossible for an attacker to recover the client's password from the password verifier v_C in a more efficient way than traversing the used dictionary. Note that it is always possible for an attacker to brute-force a password verifier such that the defined definition of DAR is the strongest possible notion.

- $\texttt{Finalise}(C, S, pw)$ takes a client, server pair (C, S) and a password pw as input, and returns 1 iff there exists a server instance S_i that accepted (C, v_C) with $v_C = (H_1, H_2, s_H)$ and $(H_1, H_2) \leftarrow \texttt{Hash}_\texttt{P}(pw; r)$ and S_i is a passive session, i.e. no Send was queried for (C, S) on session S_i. Otherwise, return 0.

The adversary in the DAR experiment outputs a (C, S, pw) triple after interacting with the $\mathsf{Execute}$ and Send oracle. This triple is handed over to $\texttt{Finalise}$ such that the experiment is successful if and only if $\texttt{Finalise}$ returns 1, i.e. the adversary is able to compute the password pw from a password verifier v_C stored on server S. Since this is always possible, we have to restrict the time the adversary is allowed to take to compute the correct password, i.e. he must not be more efficient in computing the password than performing a brute force attack. We formalise the notion of dictionary attack resistance in the following definition.

Definition 3 (Dictionary Attack Resistance) *A BPR protocol using password hashing scheme $\mathcal{H}$ is* dictionary attack resistant *if for all PPT adversaries $\mathcal{A}$ running in time t (excl. time for oracle computations) and all dictionaries $\mathcal{D}_f$ there exists a negligible function $\varepsilon(\cdot)$ such that:*

$$\Pr[(C, S, pw) \leftarrow \mathcal{A}^{\mathsf{Execute},\mathsf{Send}}(\lambda);\ \texttt{Finalise}(C, S, pw) = 1]$$

$$\leq \frac{2^{-\beta_{\mathcal{D}_{f,|pw|}}} \cdot t}{t_{\texttt{PPreHash}}} + \varepsilon(\lambda),$$

with $t_{\texttt{PPreHash}}$ being the running time of $\texttt{PPreHash}$.

Note that t used in the above definition measures time that is spend by $\mathcal{A}$ on the actual computation of pw. This time can be estimated as $t = t_{\mathcal{A}} - t_{q,E} - t_{q,S}$, where $t_{\mathcal{A}}$ is the overall running time of $\mathcal{A}$, $t_{q,E}$ is the time for processing q_E $\mathsf{Execute}$ queries, and $t_{q,S}$ is the time for processing q_S Send queries.

Our definition of dictionary attack resistance seems a reasonable compromise between the desired security and efficiency for BPR protocols. Nonetheless, it is possible to change the balance between security and efficiency by aiming at a stronger form of dictionary attack resistance that would further hide the password length, or at a weaker form of dictionary attack resistance that would disclose the sets of significant characters to the adversary. We discuss both variants in the following.

3.1.1 A Note on Relation to ZKPPC

Our model defines a complete blind password registration procedure for VPAKE protocols, in contrast to [25] that defined only zero-knowledge password policy checks (ZKPPC) and used them as a building block for VPAKE registration procedure, without modeling the latter. We observe that according to our security definitions of BPR protocols, ZKPPC proofs do not necessarily lead to secure BPR constructions. As mentioned in the previous paragraph the generic blind password registration procedure based on ZKPPC from [25] leaks positions and sets of significant characters. While this is tolerable in [25] where the actual time needed to retrieve the password from the verifier is not restricted, in our model this protocol would not satisfy dictionary attack resistance because an attacker would be able to retrieve passwords from verifiers significantly faster than required by Definition 3. Furthermore, a zero-knowledge property in the context of password verifiers seems an unnecessarily strong requirement since offline dictionary attacks can always be performed on the server side. By dropping the zero-knowledge requirement and focusing on the entire registration process we thus obtain a more realistic security model and are able to construct more efficient BPR protocols. Note that Definition 3 models the intrinsic VPAKE requirement that a server, holding a password hash and used random salt, must not be able to recover the password faster than by bruteforcing the dictionary. While this requirement also applies to ZKPPC-based BPR protocols it was not explicitly modeled in [25].

4. AN EFFICIENT BPR PROTOCOL IN THE STANDARD MODEL

A high-level overview of our BPR protocol is given in Figure 1. The client starts the registration procedure by choosing an ASCII-based password $pw \in_R \mathcal{D}_f$ of length n, which is then mapped to an integer $\pi \leftarrow \texttt{PWDtoINT}(pw)$. The client maps each password character $c_i \in pw$ to an integer $\pi_i \leftarrow \texttt{CHRtoINT}(c_i)$ and computes Pedersen commitments C_i and C'_i for π_i whereby each C'_i is a re-randomised version of C_i. The client builds vector $\boldsymbol{C} = \{C_i\}$, shuffles commitments in $\boldsymbol{C'} = \{C'_i\}$, and proves that $\boldsymbol{C'}$ contains commitments to ASCII characters, including those that are significant to fulfil the password policy f. This proof is performed using an appropriate *proof of membership* **PoM**. The client also computes the randomised password hash (H_1, H_2) using π, sends (H_1, H_2) with the hash salt s_H to the server, and proves that π used to compute (H_1, H_2) is the same as in the product of shifted commitments $C_i^{b^i}$. This proof is performed using an appropriate *proof of equivalence* **PoE**. The product of $C_i^{b^i}$ used in the verification can be computed by the server using shift base b and the received commitments $C_i \in \boldsymbol{C}$. Finally, the client proves to the server that $\boldsymbol{C'}$ is a shuffle of $\boldsymbol{C}$ using an appropriate *proof of shuffle* **PoS**. The purpose of this proof is to link the proof that pw contains ASCII characters and fulfils policy f (**PoM**) with the password hash (H_1, H_2) of π (**PoE**) without leaking positions and ASCII subsets of characters that are significant for f (as discussed in Section 3.1). The server, after successful verification of all proofs, stores the client's password verifier $v_C = (H_1, H_2, s_H)$ in its protected password database and terminates the registration protocol. For remote registration of the password verifier we assume that the BPR protocol is executed over a server-authenticated secure channel in order to protect transmission of v_C; otherwise an eavesdropping adversary would be able to recover the password by bruteforcing the dictionary. For example, our BPR protocol can be executed on top of the TLS channel established between the client and the server (cf. [29] for the technique on how to securely bind password-based protocols in the application layer to the TLS channel).

4.1 Detailed Protocol Specification

While the high-level idea of the protocol is intuitive, the actual specification becomes somewhat technical. Note that the three proofs **PoM**, **PoE** and **PoS** can be performed in parallel. Further note that all sets are *ordered* in the following and set operations are assumed to use elements from the correct positions. We first describe local pre-computation steps of the client such as password encoding and hashing before giving a detailed specification of the proofs. The protocol uses a cyclic group G of prime order q with generator g. Let $h, f_i \in_R G$ for $i \in [-4, m]$ where m is at least $|pw|$ denote random group elements whose discrete logarithms with respect to g are assumed to be unknown. In practice, m can be chosen sufficiently large in order to accommodate all reasonable password lengths. The public parameters of the protocol are $(q, g, h, \boldsymbol{f})$ with $\boldsymbol{f} = \{f_i\}$. We let $n = |pw|$ and count indices $i \in [0, n-1]$ when dealing with password characters from pw, whereas for the indices of other sets we mostly use the interval $[1, x]$, $x \in \mathbb{N}$. Note that index ranges change frequently in the description of the protocol.

4.1.1 Pre-Computations

The client chooses a password string $pw = (c_0, \ldots, c_{n-1}) \in_R \mathcal{D}_f$ compliant with the policy f, encodes it $\pi \leftarrow \texttt{PWDtoINT}(pw)$ using the appropriate shift base b, and iterates over all password character positions $i \in [0, n-1]$ to perform the following computations:

- encode the character as $\pi_i \leftarrow \texttt{CHRtoINT}(c_i)$
- commit to π_i by computing Pedersen commitments $C_i = g^{\pi_i} h^{r_i}, C'_i = C_i h^{r'}$ for $r_i, r'_i \in_R \mathbb{Z}_q^*$
- choose a unique random index $k_i \in_R [1, n]$ to shuffle each $C'_i \leftarrow C'_{k_i}$
- if π_i is *significant* for any $R_j \in R$, set $\omega_{k_i} \leftarrow R_j$, otherwise $\omega_{k_i} \leftarrow \Sigma$
- let $l_i \in \mathbb{N}$ denote the index in ω_{k_i} such that $c_i = \omega_{k_i}[l_i]$

Note that values $(C_i, C'_i, \omega_{k_i}, k_i, l_i, \pi_i, r_i, r'_i)$ will be used in the proofs of knowledge. The client then generates random salts $s_P, s_H \in_R \mathbb{Z}_p$ for the password hashing scheme and computes the password verifier $v_C = (H_1, H_2, s_H)$ where $(H_1, H_2) \leftarrow (g^{s_P}, H_1^{\tau} h^{s_H})$. Further, the client combines previously computed values $\boldsymbol{C} = \{C_i\}$. The shuffled commitments and sets ω_{k_i} are combined in specific order according to the chosen index k_i, i.e. $\boldsymbol{C'} = \{C'_{k_i}\}$ and $\boldsymbol{\omega} = \{\omega_{k_i}\}$. With these values the client can start the computation of the three proofs **PoM**, **PoE** and **PoS**. In the following we describe these three proofs and define their messages. Note that we do not mention standard checks such as checks for group membership in our description.

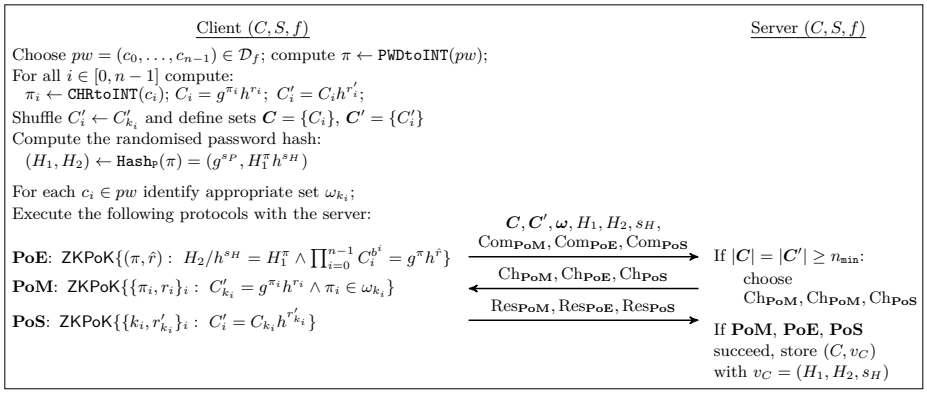

Figure 1: Our BPR Protocol — A High-Level Overview

4.1.2 *Proof of Membership* (PoM)

This protocol proves that every password character $c_{k_i} \in \omega_{k_i}$ using the shuffled set of commitments $\boldsymbol{C'}$, i.e.

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

(we use $\pi_i \in \omega_{k_i}$ as a short form of $c_i \in \omega_{k_i}$ with $\pi_i \leftarrow \texttt{CHRtoINT}(c_i)$ here). Note that $C'_{k_i} \in \boldsymbol{C'}$.

1. To prove that every C'_{k_i} commits to a value in the corresponding set ω_{k_i} the client computes the following values for the first move of the proof:

$$\forall \pi_j \in \omega_{k_i} \wedge \pi_j \neq \pi_i : s_j \in_R \mathbb{Z}_q^*, c_j \in_R \mathbb{Z}_q^*$$
$$t_j = g^{\pi_j} h^{s_j} (C'_{k_i} / g^{\pi_j})^{c_j}$$

$$k_{\rho_i} \in_R \mathbb{Z}_q^*; \ t_{l_{k_i}} = g^{\pi_i} h^{k_{\rho_i}}$$

Values $(\boldsymbol{t}_{k_i}, \boldsymbol{s}_{k_i}, \boldsymbol{c}_{k_i}, k_{\rho_i})$, with $\boldsymbol{t}_{k_i} = \{t_j, t_{l_{k_i}}\}$, $\boldsymbol{s}_{k_i} = \{s_j\}$, and $\boldsymbol{c}_{k_i} = \{c_j\}$ are stored for future use.[1] After computing the proof for every C'_{k_i} the client sets the message $\mathrm{Com}_{\mathbf{PoM}} = \boldsymbol{t} = \{\boldsymbol{t}_{k_i}\}$.

2. The server stores received values, checks them for group membership, chooses a random challenge $c \in_R \mathbb{Z}_q^*$ and sets $\mathrm{Ch}_{\mathbf{PoM}} = c$.

3. After receiving the challenge c from the server, the client computes the following verification values for all commitments C'_{k_i} (note that s_j and c_j for all $j \neq l_{k_i}$ are chosen already):

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

where i is the index of C'_{k_i} before shuffling. The client then combines $\boldsymbol{s} = \{\boldsymbol{s}_{k_i} \cup \{s_{l_{k_i}}\}\}$ and $\boldsymbol{c} = \{\boldsymbol{c}_{k_i} \cup \{c_{l_{k_i}}\}\}$.[2] The response message $\mathrm{Res}_{\mathbf{PoM}}$ is then set to $(\boldsymbol{s}, \boldsymbol{c})$.

4. To verify the proof, i.e. to verify that every commitment C'_{k_i} in $\boldsymbol{C'}$ commits to a character c_i from either a subset R_j of Σ if significant or Σ if not, the server verifies the following for every set $\omega_i \in \boldsymbol{\omega}$ with $i \in [1, n]$:

- 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 $\pi_j \in \omega_i$, $\boldsymbol{s}_i \in \boldsymbol{s}$, $\boldsymbol{t}_i \in \boldsymbol{t}$, 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 $j \in [1, |\omega_i|]$

The verification of the proof is successful iff all equations above are true *and* $\boldsymbol{\omega}$ contains all significant characters for f.

4.1.3 *Proof of Equivalence* (PoE)

This protocol proves that the password hash H_2 contains the same encoded password π as the product of the shifted commitments $\prod_{i=0}^{n-1} C_i^{b^i}$ and that the client knows the discrete logarithm s_P of H_1 to base g, in particular:

$$\mathsf{ZKPoK}\{(\pi, \hat{r}) : \ H_2/h^{s_H} = H_1^{\pi} \wedge \prod_{i=0}^{n-1} C_i^{b^i} = g^{\pi} h^{\hat{r}}\}.$$

1. The first client message $\mathrm{Com}_{\mathbf{PoE}}$ is set to (t_{s_P}, t_H, t_{C^*}) for $t_{s_P} = g^{k_{s_P}}$; $t_H = H_1^{k_\pi}$; $t_{C^*} = g^{k_\pi} h^{k_{r^*}}$ with $k_{s_P}, k_\pi, k_{r^*} \in_R \mathbb{Z}_q^*$.

2. The server stores received values and sets $\mathrm{Ch}_{\mathbf{PoE}} = c$.

3. After receiving challenge c from the server, the client computes the following verification values

$$s_{s_P} = k_{s_P} - c s_P; \ s_\pi = k_\pi - c\pi; \ s_{r^*} = k_{r^*} - c \sum_{i=0}^{n-1} b^i r_i$$

and sets $\mathrm{Res}_{\mathbf{PoE}} = (s_{s_P}, s_\pi, s_{r^*})$ as response.

4. To verify the proof, i.e. the product of shifted commitments $C_i^{b_i}$ for $C_i \in \boldsymbol{C}$ contains the same password π as the password hash H_2, the server verifies the following:

$$t_{s_P} \stackrel{?}{=} g^{s_{s_P}} H_1^c; \ t_H \stackrel{?}{=} H_1^{s_\pi} (H_2/h^{s_H})^c;$$

$$t_{C^*} \stackrel{?}{=} g^{s_\pi} h^{s_{r^*}} \left(\prod_{i=0}^{n-1} C_i^{b^i} \right)^c.$$

The server accepts the proof iff those verifications succeed.

4.1.4 *Proof of Shuffle* (PoS)

Let ϕ denote a function such that $\phi(i) = k_i$ shuffles the set $\boldsymbol{C}$ to $\boldsymbol{C'}$. We use the efficient proof for correct shuffling for

[1] Note that $t_{l_{k_i}}$ has to be added at the correct position (l_{k_i}) in $\boldsymbol{t}_{k_i}$.

[2] Note again that the set union has to consider the position of l_{k_i} to add the values at the correct position.

ElGamal ciphertexts from [16], which is an optimised version of [17], and adapt it to Pedersen commitments, which translates

$$\mathsf{ZKPoK}\{(\{k_i, r'_{k_i}\}_i) : \ C'_i = C_{k_i} h^{r'_{k_i}}\}$$

into

$$\mathsf{ZKPoK}\{A_{ji} : \ C'_i = h^{A_{0i}} \cdot \prod_{v=1}^{n} C_v^{A_{vi}}\}$$

for permutation matrix A_{ji}.

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_{ji} denote a matrix with $j \in [-4, n]$ and $i \in [0, n]$, i.e. of size $n+5 \times n+1$, such that a $n \times n$ sub-matrix of A_{ji} is the permutation matrix. Further, let ϕ denote the permutation function that, on input index i, returns the index k_i of the according shuffled element and ϕ^{-1} its inverse. 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_{ji} is defined with $A_{w0} \in_R \mathbb{Z}_q^*, A_{-1v} \in_R \mathbb{Z}_q^*$ and $A_{0v} = r'_{\phi(v)}$ with $w \in [-4, n]$ and $v \in [1, n]$. The remaining values in A_{ji} are computed as follows for $v \in [1, n]$:

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

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

The client then commits to A_{ji} and sets his output $\mathrm{Com}_{\mathbf{PoS}} = (C'_0, \tilde{f}, \boldsymbol{f}', w, \tilde{w})$ for $\boldsymbol{f}' = \{f'_v\}$ with $v \in [0, n]$.

$$f'_v = \prod_{j=-4}^{n} f_j^{A_{jv}}; \ \tilde{f} = \prod_{j=-4}^{n} f_j^{A'_j}$$

$$C'_0 = g^{\sum_{j=1}^{n} \pi_j A_{j0}} h^{A_{00} + \sum_{j=1}^{n} r_j A_{j0}}$$

$$w = \sum_{j=1}^{n} A_{j0}^3 - A_{-20} - A'_{-3}; \ \tilde{w} = \sum_{j=1}^{n} A_{j0}^2 - A_{-40}$$

Note that C'_0 has the form $\prod_{j=0}^{n} C_j^{A_{j0}} = h^{A_{00}} \prod_{j=1}^{n} C_j^{A_{j0}}$, but our way of computing it saves $n-1$ exponentiations.

2. The server chooses $\boldsymbol{c} = \{c_v\}$ with $c_v \in_R \mathbb{Z}_q^*$ for $v \in [1, n]$ and sets $\mathrm{Ch}_{\mathbf{PoS}} = \boldsymbol{c}$.

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

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

4. Finally, the server checks the following equations for a randomly chosen $\alpha \in_R \mathbb{Z}_q^*$ and $C_0 = h$:

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

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

The server accepts the **PoS** proof if all verifications succeed.

4.2 Security Analysis

The security of our BPR protocol is established by the following theorem using Lemma 1.

Theorem 1 (BPR Security) *The protocol from Section 4.1 is BPR-secure, i.e. policy compliant and dictionary attack resistant, if the discrete logarithm problem is hard in the used group G.*

To prove Theorem 1, we start with the security of the adopted shuffling approach and prove that **PoS** is a zero-knowledge proof of knowledge for the shuffle of $\boldsymbol{C}$ to $\boldsymbol{C}'$. The following Lemma 1 is proven in [16] for ElGamal ciphertexts.

Lemma 1 (PoS is a ZKPoK) *The* **PoS** *protocol from Section 4.1 is an honest verifier zero-knowledge proof of knowledge of the following statement if the discrete logarithm problem in the used group is hard,*

$$\mathsf{ZKPoK}\{A_{ji} : \ C'_i = h^{A_{0i}} \cdot \prod_{v=1}^{n} C_v^{A_{vi}}\},$$

where a $n \times n$ sub-matrix of A_{ji} is the used permutation matrix and A_{0i} the used re-randomiser.

Proof of Theorem 1. **Policy Compliance** To prove policy compliance of the construction we first show that the three proofs in the protocol are sound. This allows us to argue that every attacker winning the policy compliance experiment allows us to build an attacker against one of the three proofs in the protocol.

Claim 1 (PoE Soundness) **PoE** *is sound, i.e. for every client using $H_2 = H_1^{\pi} h^{s_H}$ and $\prod_{i=0}^{n-1} C_i^{b^i} = g^{\pi'} h^{\hat{r}}$ with $\pi \neq \pi'$ and $\hat{r} = \sum_{i=1}^{n} r_i$ the probability that the server accepts* **PoE** *is negligible.*

Proof. Soundness of **PoE** holds if the probability that

$$t_H \stackrel{?}{=} H_1^{s_\pi} (H_2 / h^{s_H})^c \text{ and } t_{C^*} \stackrel{?}{=} g^{s_\pi} h^{s_{r^*}} \left(\prod_{i=0}^{n-1} C_i^{b^i} \right)^c$$

holds for $H_2 = H_1^{\pi} h^{s_H}$ and $\prod_{i=0}^{n-1} C_i^{b^i} = g^{\pi'} h^{\hat{r}}$ with $\pi \neq \pi'$ and $\hat{r} = \sum_{i=1}^{n} r_i$ is negligible. To show that this holds we assume w.l.o.g. that s_π and s_{r^*} are chosen such that the second equation holds. In particular $s_\pi = x - c\pi'$ for an appropriate value of x. However, we see now that for the first equation to hold, the adversary would have to compute $t_H = g^{s_P s_\pi} (H_2 h^{-s_H})^c$, which implies computing $y = \log_g(s_P x - s_P c \pi' + \beta c)$ for some β. By assumption $\beta \neq s_P \pi'$ such that computation of y is impossible under the discrete logarithm assumption. Note that the additional proof for knowledge of s_P ensures that the client knows the discrete logarithm to basis g of H_1, which allows us to define the equation t_H as $H_1^{s_\pi} (H_2 h^{-s_H})^c = g^{s_P s_\pi} (H_2 h^{-s_H})^c$. □

Soundness of **PoS** is proven in the full version. Note that we require that **PoS** is a proof of knowledge for re-randomiser

r' and permutation matrix. Given soundness of the three proofs it is easy to construct a reduction from the policy compliance adversary to the soundness properties of the proofs. Let $\mathcal{A}$ denote a policy compliance attacker that has non-negligible probability to register a non-compliant password $pw \notin \mathcal{D}_f$. We construct a successful attacker $\mathcal{B}$ on the soundness of **PoM** by simulating Execute queries honestly for $\mathcal{A}$. Send queries are all simulated honestly, except for one session, in which $\mathcal{B}$ outputs the first part of **PoM** in a random Send query as its first message. This Send query returns the challenge that $\mathcal{B}$ receives. The second part of **PoM** in the second Send query of this session is output by $\mathcal{B}$, which results in a success probability of $\mathsf{Succ}_{\mathcal{A}}/q$, where q is the number of active sessions invoked by $\mathcal{A}$.

Knowing that the set membership **PoM** is sound we show how to construct a successful extractor $\mathcal{B}'$ on the permutation **PoS** using a successful attacker $\mathcal{A}$ on the policy compliance experiment. To this end $\mathcal{B}'$ simulates all Execute oracles honestly. Send queries are simulated honestly, except for one session, in which $\mathcal{B}'$ stores the first part of **PoS** in a random Send query and responds with $n+1$ linearly independent challenges $\boldsymbol{c}$ for **PoS** and a random challenge c for **PoM** and **PoE**. Gathering the $n+1$ messages in the second Send query from $\mathcal{A}$ on that session, $\mathcal{B}'$ can extract r'_i and the permutation matrix. Building an attacker on the soundness of **PoE** using a successful attacker $\mathcal{A}$ on the policy compliance experiment is similar to building $\mathcal{B}$ on **PoM**. Considering security of **PoM** and **PoE** and the soundness of **PoS** policy compliance follows.

Dictionary Attack Resistance First note that the used password hashing scheme that computes $(H_1, H_2) = (g^{s_P}, g^{s_P \pi} h^{s_H})$ with $s_P, s_H \in_R \mathbb{Z}_q^*$ is secure, cf. [25]. We show in the following that a successful attacker $\mathcal{A}$ on the dictionary attack resistance of the BPR protocol would be able to distinguish between identical distributions of real and simulated values. It is easy to see that **PoM** and **PoE** on its own are zero-knowledge proofs. The zero-knowledge property of **PoS** can be found in the full version.

We start by observing that breaking the dictionary attack resistance of the protocol implies that $\mathcal{A}$ is able to find a password pw from a BPR transcript and the server's information, i.e. the password verifier $v_C = (H_1, H_2, s_H)$, using less than $2^{\beta_{\mathcal{D}_f, |pw|}}$ exponentiations (pre-hash computations). This implies that there exists at least one index i such that **PoM** leaks the shuffled character $c_j \in pw$ for $j = \phi(i)$, or **PoS** exposes the relationship between C'_j and the according C_i and therefore the set R_j from which c_i is chosen. Let i denote the index of such a character. If $\mathcal{A}$ can identify index i, he can distinguish between $\mathcal{X}_i = (t_i = g^{\pi_i} h^{k_{\rho_i}}, s_i = k_{\rho_i} - c_i(r_{\phi^{-1}(i)} + r'_i), c_i)$ and $\mathcal{X}_0 = (t_o = g^{\pi_o} h^{s_o} (C'_{\phi(i)} / g^{\pi_o})^{c_o}, s_o, c_o)$ with $c, s_o, c_o \in_R \mathbb{Z}_q^*$ and $c_i = c \oplus \bigoplus c_o$ for $o \in [1, n], o \neq i$. This however is impossible since $\mathcal{X}_i$ and all $\mathcal{X}_o$ are identically distributed. Similarly, we can argue that distinguishing between a real and a simulated proof of shuffle is impossible. Considering that **PoM** and **PoS** do not offer any attack possibilities we see that if $\mathcal{A}$ is able to win the dictionary attack resistance game, we can distinguish between a real and a simulated **PoE**. However, this is impossible due to the zero-knowledge property of **PoE**, i.e. values t_{s_P}, t_H and t_{C^*} are identically distributed in both cases. □

Claim 2 (PoM Soundness) **PoM** *is sound, i.e. for every client C using $pw \notin \mathcal{D}_f$ the probability that the server accepts* **PoM** *is negligible.*

Proof. Note that while **PoM** is a proof of knowledge, we are not actually interested in the knowledge soundness as this comes implicitly under the discrete logarithm assumption. Instead it is sufficient in our case that client C can not make the server accept **PoM** with a password $pw \notin \mathcal{D}_f$. Soundness of **PoM** implies that if there exists a commitment C'_{k_i} that commits to an encoded character π_i *not* in the respective set ω_{k_i}, then

$$\boldsymbol{t}_i[j] \stackrel{?}{=} g^{\pi_j} h^{\boldsymbol{s}_i[j]} (C'_{k_i} / g^{\pi_j})^{\boldsymbol{c}_i[j]}$$

does not hold with overwhelming probability for given values. This holds under the assumption that the discrete logarithm problem is hard in G and $\boldsymbol{c}_{k_i}[l] \in \boldsymbol{c}$ is uniformly distributed in $\mathbb{Z}_q^*$. The first assumption is clear and the second one holds as long as $c \stackrel{?}{=} \bigoplus_{j=1}^{|\omega_i|} \boldsymbol{c}_i[j]$ holds for a uniformly at random chosen $c \in_R \mathbb{Z}_q^*$. Note that this also holds for our case in which we use the same c in all n proofs. Assuming that the client can convince the server that the equation holds in case π_i is not in ω_{k_i} it is easy to see that this is equivalent to breaking the discrete logarithm problem in G, i.e. the client can either compute $\hat{r}$ such that $C'_{k_i} = g^{\pi_i} h^{(r+r')} = g^{\pi_j} h^{\hat{r}}$, or he can compute $\boldsymbol{s}_i[j] = \pi_j \log_h(g) - \log_h(t) + c \log_h(C'_{k_i} g^{-\pi_j})$. Therefore, the claim follows since the client can not fool the server in accepting a set membership proof for a character $c_i \notin \omega_{k_i}$ in pw and the server additionally verifies that sets ω_i are necessary and sufficient to fulfil policy f. □

4.3 Performance

In the implementation of the BPR protocol we can adopt several tricks aiming to improve its performance. First, we can pre-compute and reuse values g^{π_i} on the client and server side. The computation of b^i can be performed in a way that allows to re-use previously calculated values and the implementation of the proof can be optimised allowing the client to use π. Considering this, we can estimate the performance of the BPR protocol by counting the number of exponentiations as follows. Note that we do not count exponentiations with exponents smaller than 5. The client in our BPR protocol performs $4n + 2\sum_i |\omega_i| + 113$ exponentiations. The server must perform $5n + 2\sum_i |\omega_i| + 16$ exponentiations if g^{π_i} is pre-computed and re-used. In contrast, the generic approach for ZKPPC from [25] requires $3n + 3\sum_i (|\omega_i^*| - 1) + 7$ exponentiations on the client side and $3\sum_i |\omega_i^*| + 8$ on the server side. Note that ω_i^* in [25] depends on the maximum password length and thus contains all characters from ω_i *plus* all characters from ω_i shifted by $j = 1, \ldots, n_{\max}$ positions. Therefore, the costs of the protocol from [25] are given by $3n + 2n_{\max} \sum_i (|\omega_i| - 1) + 7$ exponentiations on the client side and $2n_{\max} \sum_i |\omega_i| + 8$ on the server side. The protocol from [25] is thus much less efficient than our BPR protocol: in the optimal case where $n = n_{\max}$ the difference can be estimated by $2(n-1)\sum_i |\omega_i| - 2n^2 - n - 106$ additional exponentiations for the client and $2(n-1)\sum_i |\omega_i| - 5n - 8$ for the server.

Implementation.

We implement an unoptimised prototype of the BPR protocol over the NIST P-192 elliptic curve [34] in Python using the Charm framework [5] and measure its performance.

To this end we set $b = 10^5$ in order to achieve security guarantees for all reasonable password lengths and policies. We also implement the ZKPPC approach from [25] in order to compare its performance with our BPR implementation. The performance tests (completed on a laptop with an Intel Core Duo P8600 at 2.40GHz) underline the theoretical findings from the previous paragraph. In particular, execution of the proposed BPR protocol with a password of length 10 and policy $(dl, 5)$ needs 0.72 seconds on the client and 0.67 seconds on the server side while the ZKPPC execution requires 9.1 seconds on the client and 8.9 seconds on the server side with a maximum password length of 10. Increasing the maximum password length to 20 slows down the client to 22.7 and the server to 22.2 seconds. Our measurements show that our BPR protocol is *at least* 10 times faster than the ZKPPC-based registration approach from [25]. With the overall running time of 1.5 seconds for 10-character passwords, 2.5 seconds for 15-character passwords, and 3.3 seconds for 20-character passwords the proposed BPR protocol can be deemed practical.

4.4 Discussion

Our BPR protocol is proven secure in a strong security model, but does not hide the length of the password from the server. Arguably, this is a strong security requirement (cf. Section 3.1) that may not be needed in many practical scenarios since password policies usually aim at offering some minimum password strength such that every password of the required minimum length or longer is considered to be secure. With this in mind, it makes no difference whether the password length is known to an attacker or not since the password is assumed to be strong enough.

Nonetheless, an attacker knowing the password length can try passwords of the given length and thus use the reduced search space to speed up the dictionary attack. An initial idea for hiding the password length in our BPR protocol could be to combine commitments for *non-significant* password characters into a single commitment and use only $n_{\mathtt{min}}$ commitments in the proof. This, however, would allow a malicious client to register passwords that do not comply with the policy unless the client can prove that the exponent of the combined commitment is of the form $\sum b^i \pi_j$, which is only possible when the length of the polynomial (and therefore the password length) is known. Our BPR protocol can be modified to hide the password length at the cost of its efficiency. This can be achieved by defining a constant length $l \in \mathbb{N}$ larger than any practical $n = |pw|$, e.g., $l = 50$ or $l = 100$, and apply the following modifications. First, we change the way shuffling is performed. In particular, $\boldsymbol{C}$ is still randomly shuffled to $\boldsymbol{C'}$, but it is ensured that the first $|R|$ commitments C'_i are for characters that are significant for the policy f. All computations in the protocol are now performed over the password $\pi^* = \pi || 0 \ldots 0$, where π is the original client-chosen, encoded password, and $|\pi^*| = l$. This allows to define set ω_i for character commitment C'_i as either some R_j if significant, or Σ if $i \leq n_{\mathtt{min}}$ and the character in C'_i is not significant, or $\Sigma \cup \{0\}$ otherwise. The remaining protocol steps remain unchanged. Through these modifications the original password length remains hidden so that stronger flavour of dictionary attack resistance can be proven for the modified BPR protocol using min-entropy $\beta_{\mathcal{D}_f} = -\max_{pw \in \mathcal{D}_f} \sum_{i=0}^{n-1} [\mathbb{D}_\Sigma(c_i) \lg(\mathbb{D}_\Sigma(c_i))]$ for the dictionary $\mathcal{D}_f$ containing all policy-compliant passwords of length *up to* l. Note that this modification trades off stronger security for efficiency due to the use of l for all shorter passwords.

Our BPR protocol can also be made more efficient if we are willing to sacrifice privacy of character positions for significant characters and reveal information about corresponding character sets (as in [25]). In this case the proof **PoS** becomes redundant and all steps related to it can be removed. This would significantly reduce the number of exponentiations to about $2n$ on the client and $4n$ on the server side. The resulting BPR protocol would still offer a weaker flavour of dictionary attack resistance that does not hide positions and sets of significant password characters as discussed in Section 3.1 yet remain more efficient than the ZKPPC-based registration protocol from [25], which seems to offer comparable security guarantees.

5. REFERENCES

[1] M. Abdalla, E. Bresson, O. Chevassut, B. Möller, and D. Pointcheval. Provably secure password-based authentication in tls. In *ASIACCS '06*, pages 35–45, New York, NY, USA, 2006. ACM.

[2] M. Abdalla, E. Bresson, O. Chevassut, B. Möller, and D. Pointcheval. Strong password-based authentication in TLS using the three-party group Diffie Hellman protocol. *Int. J. Secur. Netw.*, 2(3/4):284–296, Apr. 2007.

[3] M. Abdalla, D. Catalano, C. Chevalier, and D. Pointcheval. Efficient two-party password-based key exchange protocols in the uc framework. In *CT-RSA'08*, volume 4964 of *LNCS*, pages 335–351. Springer, 2008.

[4] M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. In *PKC'05*, volume 3386 of *LNCS*, pages 65–84. Springer, 2005.

[5] 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*, 3(2):111–128, 2013.

[6] S. Bayer and J. Groth. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In *EUROCRYPT'12*, volume 7237 of *LNCS*, pages 263–280. Springer, 2012.

[7] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT'00*, volume 1807, pages 139–155. Springer, 2000.

[8] S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In *IEEE S&P'92*, pages 72–84. IEEE, 1992.

[9] S. M. Bellovin and M. Merritt. Augmented Encrypted Key Exchange: A Password-Based Protocol Secure against Dictionary Attacks and Password File Compromise. In *ACM CCS'93*, pages 244–250. ACM, 1993.

[10] F. Benhamouda, O. Blazy, C. Chevalier, D. Pointcheval, and D. Vergnaud. New Techniques for SPHFs and Efficient One-Round PAKE Protocols. In *CRYPTO'13*, volume 8042 of *LNCS*, pages 449–475. Springer, 2013.

[11] F. Benhamouda and D. Pointcheval. Verifier-Based Password-Authenticated Key Exchange: New Models and Constructions. *IACR Cryptology ePrint Archive*, 2013:833, 2013.

[12] Boyko, MacKenzie, and Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *EUROCRYPT'00*, volume 1807 of *LNCS*, pages 156–171. Springer, 2000.

[13] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally Composable Password-Based Key Exchange. In *EUROCRYPT'05*, pages 404–421. Springer, 2005.

[14] S. Chiasson, P. C. van Oorschot, and R. Biddle. A usability study and critique of two password managers. In *Proceedings of the 15th USENIX Security Symposium, Vancouver, BC, Canada, July 31 - August 4, 2006*. USENIX Association, 2006.

[15] Dan Goodin. Hack of Cupid Media dating website exposes 42 million plaintext passwords. http://goo.gl/oeT6Mp, 2014. Accessed: 25/03/2015.

[16] J. Furukawa. Efficient and Verifiable Shuffling and Shuffle-Decryption. *IEICE Transactions*, 88-A(1):172–188, 2005.

[17] J. Furukawa and K. Sako. An Efficient Scheme for Proving a Shuffle. In *CRYPTO'01*, volume 2139 of *LNCS*, pages 368–387. Springer, 2001.

[18] C. Gentry, P. D. MacKenzie, and Z. Ramzan. A Method for Making Password-Based Key Exchange Resilient to Server Compromise. In *CRYPTO'06*, volume 4117 of *LNCS*, pages 142–159. Springer, 2006.

[19] F. Hao and P. Ryan. J-PAKE: authenticated key exchange without PKI. *Transactions on Computational Science*, 11:192–206, 2010.

[20] D. Harkins and G. Zorn. RFC 5931 - Extensible Authentication Protocol (EAP) Authentication Using Only a Password, aug 2010.

[21] P. Inglesant and M. A. Sasse. The true cost of unusable password policies: password use in the wild. In *CHI*, pages 383–392. ACM, 2010.

[22] D. P. Jablon. Extended Password Key Exchange Protocols Immune to Dictionary Attacks. In *WETICE'97*, pages 248–255. IEEE Computer Society, 1997.

[23] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT01*, volume 2045 of *LNCS*, pages 475–494. Springer, 2001.

[24] J. Katz and V. Vaikuntanathan. Round-optimal password-based authenticated key exchange. In *Proceedings of the 8th conference on Theory of cryptography*, TCC'11, pages 293–310. Springer, 2011.

[25] F. Kiefer and M. Manulis. Zero-knowledge password policy checks and verifier-based PAKE. In *ESORICS'14*, volume 8713 of *LNCS*, pages 295–312. Springer, 2014. (The proceedings version is superseded by the updated full version in http://eprint.iacr.org/2014/242).

[26] T. H. Kim, H. C. Stuart, H. Hsiao, Y. Lin, L. Zhang, L. Dabbish, and S. B. Kiesler. YourPassword: applying feedback loops to improve security behavior of managing multiple passwords. In *ASIACCS'14*, pages 513–518. ACM, 2014.

[27] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, and S. Egelman. Of passwords and people: measuring the effect of password-composition policies. In *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*, pages 2595–2604. ACM, 2011.

[28] D. Kuegler and Y. Sheffer. RFC 6631 - Password Authenticated Connection Establishment with the Internet Key Exchange Protocol version 2 (IKEv2), jun 2012.

[29] M. Manulis, D. Stebila, and N. Denham. Secure modular password authentication for the web using channel bindings. In *1st International Conference on Security Standardization Research (SSR 2014)*, volume 8893 of *LNCS*, pages 167–189. Springer, 2014.

[30] M. L. Mazurek, S. Komanduri, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, P. G. Kelley, R. Shay, and B. Ur. Measuring password guessability for an entire university. In *CCS'13*, pages 173–186. ACM, 2013.

[31] nakedsecurity. Anatomy of a password disaster - Adobe's giant-sized cryptographic blunder. http://goo.gl/enB4Oi, 2014. Accessed: 25/03/2015.

[32] A. Narayanan and V. Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *ACM Conference on Computer and Communications Security*, pages 364–372. ACM, 2005.

[33] Nik Cubrilovic. RockYou Hack: From Bad To Worse. http://goo.gl/oJqj4D, 2014. Accessed: 25/03/2015.

[34] NIST. National Institute of Standards and Technology. Recommended elliptic curves for federal government use. http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf, 1999.

[35] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, 1991.

[36] C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.

[37] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, M. L. Mazurek, L. Bauer, N. Christin, and L. F. Cranor. Encountering stronger password requirements: user attitudes and behaviors. In *SOUPS'10*, volume 485. ACM, 2010.

[38] S. Shin and K. Kobara. Efficient Augmented Password-Only Authentication and Key Exchange for IKEv2. RFC 6628 (Experimental), June 2012.

[39] M. Weir, S. Aggarwal, M. P. Collins, and H. Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *CCS'10*, pages 162–175. ACM, 2010.