Zero-Knowledge Password Policy Checks and Verifier-Based PAKE

Franziskus Kiefer and Mark Manulis

Department of Computing, University of Surrey, UK
mail@franziskuskiefer.de, mark@manulis.eu

Abstract. We propose the concept of Zero-Knowledge Password Policy Checks (ZKPPC) to enable remote registration of client passwords without their actual transmission to the server. The ZKPPC protocol executed as part of the client registration process allows the client to prove compliance of the chosen password with the password policy defined by the server. The main benefit of ZKPPC-based password registration is that it guarantees that passwords can never be processed nor stored in clear on the server side. At the end of the registration phase the server only receives and stores some verification information that can later be used for authentication in suitable Verifier-based Password Authenticated Key Exchange (VPAKE) protocols.

To this end, we first formalize the requirements of ZKPPC protocols and propose a general framework for their construction in the standard model using randomised password hashing and set membership proofs. We design a suitable encoding scheme for password characters and show how to express password policies to allow the adoption of set membership proofs. Finally, we present a concrete ZKPPC-based registration protocol that is based on efficient Pedersen commitments and corresponding proofs, and analyse its performance.

To complete the ZKPPC-based registration and authentication framework we propose a concrete VPAKE protocol, where the server can use the obtained verification information from the ZKPPC-based registration phase to subsequently setup secure communication sessions with the client. Our VPAKE protocol follows the recent framework for the construction of such protocols and is secure in the standard model.

1 Introduction

Password policies set by organisations aim at enforcing a higher level of security on used passwords by specifying various requirements that apply during their selection process and the actual usage. Especially, when passwords are selected and used by users in a remote way strong password policies can help not only to protect data behind individual user accounts but also to prevent malicious activities from compromised accounts that could further harm the organisation due to liability issues or even lead to a compromise of the entire system or service. It is known that in the absence of any password policy users tend to

choose "weak" passwords that are easily guessable and have higher risk of being compromised through dictionary attacks [24]. It is worth noting that coming up with a good password policy is still considered a difficult task since policies must also remain usable in practice [15].

In this work we focus on widely used password policies that specify the requirements on the selection of passwords such as the minimum password length, define sets of admissible password characters, and may contain further restrictions on the number of characters from each set. These requirements are typically enforced during the initial password registration process and aim at preventing users from choosing "weak" passwords. These policies are often extended with additional restrictions on the usage of passwords by requiring users to change their passwords within a certain period of time.

In case where users select passwords for remote access to systems or services on their own, the password policy enforcement mechanism must be able to verify that selected passwords comply with the existing policy. This compliance check can be performed either on the client side or on the server side. For instance, when a commodity web browser is used to register for some web service the policy can be checked within the browser using scripts embedded into the registration website, or on the server side upon the initial transmission of the password (e.g. over a TLS channel). Both approaches, however, have security risks as discussed in the following. If policy enforcement is performed solely on the client side, the server must trust the client to obey the policy and execute the check correctly. This is not a threat if the compliance check is assumed to be in the interest of an honest user. Nonetheless, malicious users can easily circumvent such script-based verification and submit passwords that are not compliant with the policy. Depending on the nature of the service, the corresponding service provider might want to exclude this threat. In this case the compliance check must be performed on the server side. In order to perform such a check with available technologies the client's password must be transmitted to the server, possibly over a secure channel. This ultimately requires the client to trust the server to process and store the received password in a secure way. While many servers adopt the current state-of-the-art approach for storing passwords in a hashed form, e.g. using PBKDF2 [16,23] or bcrypt [20], with a random salt to protect against server compromise or re-use attacks, there have been many known cases, e.g. [21,17,22,12], where passwords have been stored in clear and compromised subsequently. Therefore, trusting servers with secure processing and storage of user passwords is clearly not the best approach.

This imposes the following questions: How can users remotely choose passwords and prove their policy compliance to a remote server without actually transmitting their passwords to the server? The second question, that follows the first one, is how can users authenticate themselves remotely to the server with passwords without transmitting their passwords? Interestingly, cryptographic techniques for password-based authentication without password transmission already exist in form of Password-Authenticated Key Exchange (PAKE) protocols, e.g. [2,1,9,19]. PAKE protocols offer mutual authentication and computation of

secure session keys based solely on the input password in a way that makes it hard for an active adversary to recover passwords via offline dictionary attacks. Traditional PAKE protocols assume that the password is input in clear on both sides. To alleviate the threat that stored passwords are revealed immediately in case of a server compromise, the so-called Verifier-based PAKE (VPAKE) protocols [3,14,5] define the server's input as some verification information. Knowledge of these verification information still requires an offline dictionary attack to reveal the password. VPAKE protocols thus offer better protection than PAKE since they force the attacker who breaks into the server and is willing to recover passwords to perform an additional offline dictionary attack. The aforementioned trust assumption on the server to securely process and store passwords becomes irrelevant if the password setup only transmits password verification information to the server, which can later be used in VPAKE protocols. In combination with password policy enforcement this approach would however require a solution to the first question; namely the client must be able to prove that the verification information for VPAKE have been derived from a password that complies with the server's password policy.

Zero-Knowledge Password Policy Checks (ZKPPC): Our first contribution is the concept of Zero-Knowledge Password Policy Checks (ZKPPC) in Section 5. This is a new class of protocols that allows a server to perform policy checks on passwords in a zero-knowledge manner. The use of ZKPPC allows clients to register policy-conform passwords without disclosing them to the server. ZKPPC can be used to solve the aforementioned problem of password setup where only password verification information is supposed to be transmitted to the server. We present a security model for ZKPPC, a general framework for their construction and a concrete instantiation. As a building block for the construction of ZKPPC protocols we introduce a new method for processing passwords and policies in Section 3.

ZKPPC-compliant VPAKE: Our second contribution is a new verifier-based PAKE protocol in the standard model in Section 6, which instantiates the generic construction from [5]. This VPAKE protocol is ZKPPC-compliant and allows us to solve the second aforementioned problem of password authenticated key exchange using only a password verifier on the server side. In order to build the new VPAKE protocol we make use of a new randomised password-hashing scheme, defined in Section 4.

2 Concept Overview and Building Blocks

Our concept entails performing a zero-knowledge password policy check during the client registration phase, which results in password verification information being passed on to the server, and subsequent use of this verification information on the server side as input to a suitable VPAKE protocol for the purpose of password-based authentication. A client wishing to register its username and password at a remote server that maintains a password policy will initially pick a password and execute the ZKPPC protocol with the server. The ZKPPC protocol

ensures that the password chosen by the client complies with server's password policy and is linked to the verification information communicated to the server at the end of the registration phase. This verification information is computed through a randomised password hashing scheme and includes the randomness that was used by the client. The actual client's password is not transmitted to the server and the only way for the server to reveal the password is to execute an offline dictionary attack using the obtained verification information. That is, a malicious server would have to perform about the same amount of computation to recover passwords as an attacker who breaks into the server. At the same time an honest server will be able to recognise and reject any attempt of a malicious client in setting up a non-policy conform password, still without learning the latter. In order to realise the ZKPPC functionality (incl. the randomised password hashing scheme) and to construct a suitable VPAKE protocol we make use of the following building blocks.

Zero Knowledge Proofs A proof of knowledge PoK between prover and verifier proves that a word C is in language L. A word is in language L if a witness w exists proving so. We use common notation $\mathsf{PoK}\{(\alpha,\beta) : \ y = l(\alpha,\beta)\}$ to describe a proof of knowledge of (α,β) such that $y = l(\alpha,\beta)$ for public value y and public function l. PoK is a zero knowledge proof of knowledge (ZKPoK) if the verifier learns nothing about (α,β). More formally: An interactive protocol PoK for a language L between prover P and verifier V is a zero knowledge proof of knowledge (ZKPoK) if the following holds:

- Completeness: If $x \in L$, V accepts if P holds a witness w proving so.
- Soundness: There exists an efficient knowledge extractor Ext that can extract a witness w from any malicious prover $P^*(x)$ with $x \in L$ that has non-negligible probability of making $V(x, L)$ accept.
- Zero-Knowledge: If $x \in L$, there exists an efficient simulator Sim, on input x, able to generate a view, indistinguishable from a (malicious) verifier's view.

Note that we have to consider malicious verifiers, i.e. not only honest verifier ZK, since our ZKPPC protocol should remain secure even if a malicious server tries to retrieve the password from the interaction with the client.

Commitments Commitments allow us to commit to the password's characters without disclosing them, and thus reason on the policy conformity of passwords. $\mathtt{C} = (\mathtt{CSetup}, \mathtt{Com})$ is a commitment scheme if it offers the following properties:

- Efficient: $\mathtt{p_C} \leftarrow \mathtt{CSetup}(\lambda)$ and $(C,(x,r)) = (C,d) \leftarrow \mathtt{Com}(\mathtt{p_C}, x; r)$ for $x \in \mathbb{X}$ and $r \in_R \mathbb{S}$ are poly-time algorithms.
- Complete: For all $x \in \mathbb{X}, r \in \mathbb{S}_C$ and all $\mathtt{p_C} \leftarrow \mathtt{CSetup}(\lambda) : \ \mathtt{Com}(\mathtt{p_C}, d) = (C,d)$ for $(C,d) \leftarrow \mathtt{Com}(\mathtt{p_C}, x; r)$
- Binding: For all PPT algorithms A there exists a negligible function $\varepsilon_{\mathrm{bi}}(\cdot)$ such that for all $(x, x', r, r', C) \leftarrow A(\mathtt{p_C})$ with $\mathtt{p_C} \leftarrow \mathtt{CSetup}(\lambda)$:

$$\Pr[x \neq x' \wedge (C,d) = \mathtt{Com}(\mathtt{p_C}, x; r) \wedge (C,d') = \mathtt{Com}(\mathtt{p_C}, x'; r')] \leq \varepsilon_{\mathrm{bi}}(\lambda)$$

– Hiding: For all PPT algorithms A there exists a negligible function $\varepsilon_{\text{hi}}(\cdot)$ such that for all x_0, x_1 with $|x_0| = |x_1|$ and $\mathtt{p_C} \leftarrow \mathtt{CSetup}(\lambda), b \in_R \{0,1\}, (C,d) \leftarrow \mathtt{Com}(\mathtt{p_C}, x_b; r)$ and $b' \leftarrow A(\mathtt{p_C}, C, x_1, x_2)$:

$$\Pr[b = b'] \leq 1/2 + \varepsilon_{\text{hi}}(\lambda)$$

A commitment scheme is *perfectly binding* if $\varepsilon_{\text{bi}}(\cdot)$ is zero and *perfectly hiding* if $\varepsilon_{\text{hi}}(\cdot)$ is zero. A commitment scheme is said to be *homomorphic* if there exist functions Ψ, ψ and Φ such that for all $\mathtt{p_C} \leftarrow \mathtt{CSetup}(\lambda), (C_i, d_i) \leftarrow \mathtt{Com}(\mathtt{p_C}, x_i; r_i)$ with $x_i \in \mathbb{X}$ and $r_i \in \mathbb{S}$ for $i \in 0, \ldots, m$ it holds that $\Psi_{i=0}^{m} C_i = \mathtt{Com}(\mathtt{p_C}, \psi_{i=0}^{m} x_i; \Phi_{i=0}^{m} r_i)$. Note that we write $C \leftarrow \mathtt{Com}(x; r)$ to generate a commitment C for x using randomness r and omit parameters $\mathtt{p_C}$ and decommitment d for convenience.

Instantiation [18] In this work we use the Pedersen commitment scheme [18], which is perfectly hiding and suits our needs as it is homomorphic. Let $\mathtt{C_P} = (\mathtt{CSetup}, \mathtt{Com})$ with $(g, h, p, \lambda) \leftarrow \mathtt{CSetup}(\lambda)$ 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 p with bit-length in the security parameter λ and the discrete logarithm of h with respect to base g is not known.

Set Membership Proofs Set membership proofs are special zero knowledge proofs, which prove that a committed value is an element of a specific set. 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. A proof of set membership for $\mathtt{C}$ and $x \in L$ is defined as $\mathsf{ZKPoK}\{(\xi, \rho) : C \leftarrow \mathtt{Com}(\xi; \rho) \wedge \xi \in L\}$. Note that we write $\mathsf{SMP}(\xi, \rho, L)$ as an abbreviation.

An Efficient Set Membership Proof [7] We use the efficient set membership proof from Camenisch, Chaabouni and Shelat [7]. The scheme uses Boneh-Boyen (BB) signatures [6] that are defined as follows: $(x, y = g^x) \leftarrow \mathsf{KGen}(\lambda)$ with $x \in_R \mathbb{Z}_p$, $\omega = g^{1/(x+m)} \leftarrow \mathsf{Sign}(m, x)$ and $e(\omega, y g^m) \stackrel{?}{=} e(g, g) \leftarrow \mathtt{Verify}(m, \omega, y)$. Let Ω denote the set we want to prove membership of. Figure 3 in Appendix B depicts the set membership proof from [7] between a prover P (the client) and a verifier V (the server) on common input $(C, \Omega, \mathtt{par})$. The client further knows ω and r such that $C \leftarrow \mathtt{Com}(\omega; r)$. Note that we omit standard checks such as verification if an element is a generator, but notice that the prover has to verify the signatures from set $\{\Xi_i\}$. The verifier creates BB signatures Ξ_i for all elements i in Ω and sends the public signature key y together with the set of signatures $\{\Xi_i\}$ to the prover. What follows now is a zero-knowledge proof of knowledge of a signature Ξ_ω, the same ω commited to in C. The prover blinds the signature $V \leftarrow \Xi_\omega^v$ with $v \in_R \mathbb{Z}_p$ and sends it together with verification values $a = e(V, g)^{-s} e(g, g)^t$ and $d = g^s h^m$ for $s, t, m \in_R \mathbb{Z}_p$ back to the server. After getting the challenge c from the server, the prover returns $z_\omega = s - \omega c, z_H = t - vc$ and $z_r = t - vc$ to the server that can check the values to complete the zero-knowledge proof of knowledge. The described set membership proof is only secure against honest verifiers. To transform it into a general zero-knowledge set membership proof

one can use common mechanisms from [10] or resort to the random oracle model and use it in a non-interactive version using the Fiat-Shamir transformation [13].

Encryption A labelled encryption scheme is IND-CCA2 secure if for all PPT adversaries $\mathcal{A}$ there exists a negligible function $\varepsilon(\cdot)$ such that :

$$\mathsf{Adv}_{\mathcal{A}}^{\mathrm{CCA2}}(\lambda) = \left|\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathrm{CCA2}}(\lambda) = 1] - \tfrac{1}{2}\right| \leq \varepsilon(\lambda)$$

$\mathsf{Exp}_{\mathcal{A}}^{\mathrm{CCA2}}(\lambda)$: For key-pair $(\mathtt{pk}, \mathtt{sk})$ the adversary generates (m_0, m_1) on input of $\mathtt{pk}$ and access to an encryption oracle. After generating $c \leftarrow \mathtt{Enc}_{\mathtt{pk}}(\ell, m_b; r)$ for random bit $b \in_R \{0,1\}$ the adversary gets $(\mathtt{pk}, m_0, m_1, c)$ as input and access to an encryption and decryption oracle and has to output a bit b'. The experiment returns $b = b'$. Note that $\mathcal{A}$ must not query the decryption oracle with c.

Cramer-Shoup Encryption [11] We use the labelled Cramer-Shoup (CS) encryption scheme [11] over cyclic group G of prime-order p with generators g_1 and g_2 defined as follows: Let $C = (\ell, \boldsymbol{u}, e, v) \leftarrow \mathtt{Enc}_{\mathtt{pk}}^{\mathtt{CS}}(\ell, m; r)$ with $\boldsymbol{u} = (u_1, u_2) = (g_1^r, g_2^r)$, $e = h^r m$ and $v = (cd^{\xi})^r$ with $\xi = H_k(\ell, \boldsymbol{u}, e)$ denote a labelled Cramer-Shoup ciphertext for a message $m \in G$. The CS public key is defined as $\mathtt{pk} = (p, G, g_1, g_2, h, c, d, H_k)$ with $c = g_1^{x_1} g_2^{x_2}, d = g_1^{y_1} g_2^{y_2}, h = g_1^z$ and hash function H_k such that $\mathtt{sk} = (x_1, x_2, y_1, y_2, z)$ denotes the decryption key. Decryption is defined as $m = \mathtt{Dec}_{\mathtt{sk}}^{\mathtt{CS}}(C) = e/u_1^z$ if $u_1^{x_1+y_1\cdot\xi'} u_2^{x_2+y_2\cdot\xi'} = v$ with $\xi' = H_k(\ell, \boldsymbol{u}, e)$.

Smooth Projective Hashing (SPHF) We consider only SPHFs on cyclic groups here and use the notation $\odot$ and common matrix and vector operations on it from [4]: for $a \in G,\ r \in \mathbb{Z}_p :\ a \odot r = r \odot a = a^r \in G$. Let $L_{\mathtt{aux}}$ denote a language such that ciphertext $C \in L_{\mathtt{aux}}$ if there exists a witness w proving so. A *smooth projective hash function* for ciphertext language $L_{\mathtt{aux}}$ consists of the following four algorithms:

- $\mathtt{KGen}_{\mathtt{H}}(L_{\mathtt{aux}})$ generates a hashing key $\mathtt{k_h} \in_R \mathbb{Z}_p^{1\times n}$ for language $L_{\mathtt{aux}}$.
- $\mathtt{KGen}_{\mathtt{P}}(\mathtt{k_h}, L_{\mathtt{aux}})$ derives the projection key $\mathtt{k_p} = \Gamma \odot \mathtt{k_h} \in G^{k\times 1}$.
- $\mathtt{Hash}(\mathtt{k_h}, L_{\mathtt{aux}}, C)$ outputs the hash value $h = \Theta_{\mathtt{aux}}(C) \odot \mathtt{k_h} \in G$.
- $\mathtt{PHash}(\mathtt{k_p}, L_{\mathtt{aux}}, C, w)$ returns the hash value $h = \lambda \odot \mathtt{k_p} \in G$, with $\lambda = \Omega(w, C)$ for some $\Omega : \{0,1\}^* \mapsto G^{1\times k}$.

A SPHF is correct if for all $C \in L$, with w proving so, $\mathtt{Hash}(\mathtt{k_h}, L_{\mathtt{aux}}, C) = \mathtt{PHash}(\mathtt{k_p}, L_{\mathtt{aux}}, C, w)$. It is smooth if for all $C \notin L_{\mathtt{aux}}$, the hash value h is indistinguishable from a random element in G.

Instantiation [4] A perfectly smooth SPHF for labelled CS encryption $C \leftarrow \mathtt{Enc}_{\mathtt{pk}}^{\mathtt{CS}}(\ell, x; r)$ from [4] on $L_{(x)}$ is defined as follows:

- $\mathtt{KGen}_{\mathtt{H}}(L_x)$ generates five random elements $\mathtt{k_h} = (\eta_1, \eta_2, \theta, \mu, \nu) \in_R \mathbb{Z}_p^{1\times 5}$.
- $\mathtt{KGen}_{\mathtt{P}}(\mathtt{k_h}, L_x)$ on input of language L_x and hashing key $\mathtt{k_h}$ returns the projection key $\mathtt{k_p} = (g_1^{\eta_1} g_2^{\theta} h^{\mu} c^{\nu}, g_1^{\eta_2} d^{\nu})$.

– $\texttt{Hash}(\texttt{k}_\texttt{h}, L_x, C)$ on input of a ciphertext C, language L_x and hashing key $\texttt{k}_\texttt{h}$ returns the hash value $h = u_1^{\eta_1+\xi\eta_2} u_2^{\theta}(e/x)^{\mu} v^{\nu}$.
– $\texttt{ProjHash}(\texttt{k}_\texttt{p}, L_x, C, r)$ on input of projection key $\texttt{k}_\texttt{p}$, language L_x, ciphertext C and according randomness r returns the hash value $h = (\texttt{k}_{\texttt{p}_1}\texttt{k}_{\texttt{p}2}^{\xi})^r$.

3 Modelling Passwords and Policies

In cryptography, passwords are usually modelled as integers, i.e. using a mapping from dictionaries to $\mathbb{Z}^+$. This allows to use them in computations like any other secret key. However, such mapping destroys the structure of a password. To perform operations on the actual password, i.e. the character string, we need a more careful translation from character strings to integers. In the following we define passwords, their composition from characters and how they build dictionaries. We further formally define password policies as regular expressions over certain character sets.

3.1 Passwords and Dictionaries

We consider passwords consisting of all 94 *printable* ASCII characters, which are either *digits*, *upper case letters*, *lower case letter* or *symbols*. Note that we do not consider passwords consisting of other characters. However, it is straightforward to extend our approach to UTF-8 or any other character set. Let Σ denote the set of all possible characters. We define the following subsets of Σ:

– $d = [0-9]$ or ASCII codes $[48-57]$
– $u = [A-Z]$ or ASCII codes $[65-90]$
– $l = [a-z]$ or ASCII codes $[97-122]$
– $s = [!"\#\$\%\&'()*+,-./\ :;<=>?@\ [\backslash]\hat{}\_`\ \{|\}\sim]$
  or ASCII codes $[33-47, 58-64, 91-96, 123-126]$

We denote the general dictionary $\mathcal{D}$, which comprises all power sets of Σ, i.e. all strings formed from printable ASCII characters. The set of possible characters is defined as $\Sigma = d \cup u \cup l \cup s$. A password $\text{pw} = (c_0, \ldots, c_{n-1}) \in \Sigma^n \subset \mathcal{D}$ is an ordered set (string) of n characters chosen from Σ.

Encoding In order to maintain the character structure of passwords when mapping them to integers we need a suitable encoding scheme. The standard ASCII encoding is not directly suitable, neither are common hashing techniques. Instead we define a modified ASCII encoding, to encode characters from d, u, l and s to $\mathbb{Z}_{95}^+$. Let $p : \Sigma \mapsto \mathbb{Z}_{95}^+$ denote a partially defined mapping function characterised as

$$p(x) = \begin{cases} \perp & \text{if } \texttt{ASCII}(x) < 32 \\ \texttt{ASCII}(x) - 32 & \text{if } 33 \leq \texttt{ASCII}(x) \leq 126 \\ \perp & \text{if } 126 < \texttt{ASCII}(x) \end{cases}$$

where `ASCII` returns the decimal ASCII code of a character. A password pw is then defined as $P = \sum_{i=0}^{n-1} 95^i p(c_i)$ for $c_i \in \Sigma$. We use $\emptyset$ as special character denoting zero. Note that $\emptyset$ is not a valid character in passwords, e.g., $95 = \emptyset!$ is not a valid password but $97 =$ "! is one. As P is in fact conversion from base 95 to base 10 it is an injective function from character strings of size n to $\mathbb{Z}^+_{95^n}$ with the common modulo operation as its inverse function, e.g., the encoded password 797353 corresponds to the password string 2Ax, which is computed as concatenation of $797353 \mod 95 = 18 \mathrel{\widehat{=}} 2$ at position 0, $(797353 \mod 95^2) - (797353 \mod 95) = 3135 = 33 \cdot 95^1 \mathrel{\widehat{=}}$ A at position 1 and $797353 - (797353 \mod 95^2) = 794200 = 88 \cdot 95^2 \mathrel{\widehat{=}}$ x at position 2.

3.2 Password Policies

We consider a password policy as a regular expression[1] in combination with upper and lower bounds for the password length, which on input pw evaluates to `true` or `false`, depending on whether the password is policy conform or not. We do not need the full power of regular expressions to model password policies. Instead, we can focus on a specific subset of regular expressions that is specified in the following. Note that for useful regular expressions we do not work on Σ but on $\Sigma' = \{d, u, l, s\}$. In particular, the alphabet consists of identifiers for the four aforementioned ASCII subsets. A password policy $f = (R, m, n)$ consists of a regular expression R, a minimum password length m and an upper limit for passwords n. The upper limit n can be picked such that sufficiently long passwords are admitted and will have impact on the efficiency of the corresponding zero-knowledge proofs for policy compliance. The following examples illustrate the definition of password policies f using regular expressions:

- "between 5 and 15 characters and at least one symbol and one digit" corresponds to $f = (\texttt{sd}, 5, 15)$
- "between 8 and 15 characters and at least two symbols and one digit" corresponds to $f = (\texttt{ssd}, 8, 15)$
- "between 10 and 20 characters and at least one digit, one upper case letter, and one symbol" corresponds to $f = (\texttt{dus}, 10, 20)$

We denote the set of *policy conform passwords* by $\mathcal{D}_f$, i.e. all passwords pw such that $f(\text{pw})$ returns `true`.

4 Randomised Password Hashing

To compute password verifier H from some password pw for later use in a VPAKE protocol we define randomised password hashing by extending the definition from [5] to allow the pre-hash computation to use a random salt as well.

[1] Other definitions for policies are possible and maybe even more appropriate, for example including search for natural words in the password (e.g. dropbox password-meter[2]). However, it seems difficult to realise proofs with such complex constraints at this stage so that we leave their consideration for future work.

Using an empty pre-hash salt $\perp$, our definition for password hashing is equivalent to the one used in [5]. We will see advantages of this definition later in the construction of a suitable VPAKE protocol. A password hashing scheme Π consists of five algorithms:

- $\texttt{PSetup}(\lambda)$ on input of security parameter λ generates password hashing parameters $\texttt{p}_\texttt{P}$.
- $\texttt{PPHSalt}(\texttt{p}_\texttt{P})$ on input of parameters $\texttt{p}_\texttt{P}$ generates a random pre-hash salt $s_P \in_R \mathbb{S}_P$ from randomness space $\mathbb{S}_P$.
- $\texttt{PPreHash}(\texttt{p}_\texttt{P}, \text{pw}, s_P)$ on input of parameters $\texttt{p}_\texttt{P}$, password pw and pre-hash salt s_P deterministically computes the pre-hash value P.
- $\texttt{PHSalt}(\texttt{p}_\texttt{P})$ on input of parameters $\texttt{p}_\texttt{P}$ generates a random hashing salt $s_H \in_R \mathbb{S}_H$ from randomness space $\mathbb{S}_H$.
- $\texttt{PHash}(\texttt{p}_\texttt{P}, P, s_H)$ on input of parameters $\texttt{p}_\texttt{P}$, pre-hash P and salt s_H deterministically computes the hash value H.

The use of a pre-hash salt is motivated by the fact that algebraic hash functions usually allow to precompute certain values. Using these precomputed values and the salt it is easy for a server to retrieve the password. Using separate salts for pre-hash and hash functions allows us to prevent the server from precalculating password hashes. While random oracle-like password hashing is not usually vulnerable to precalculations, it seems difficult to build precalculation-secure, pure algebraic password hashing schemes without additional randomness. The main reason for the authors of [5] not to use randomness in `PPreHash` seems the increased round complexity in VPAKE protocols using these password hashes. However, it is possible to build one-round VPAKE protocols from their framework using randomness in `PPreHash`, as we will see later.

Note that we write $H \leftarrow \texttt{Hash}_\texttt{P}(\text{pw}, r)$ as short form of $H \leftarrow \texttt{PHash}(\texttt{p}_\texttt{P}, P, s_H)$ with $P \leftarrow \texttt{PPreHash}(\texttt{p}_\texttt{P}, \text{pw}, s_P)$, where $r = (s_P, s_H)$ combines the randomness used in `PHash` and `PPreHash`. We consider (second) pre-image resistance and entropy preservation from [5] as security properties for password hashing. Due to space limitations we refer to Appendix A for formal definition of those properties. While those definitions are password hashing related, we further consider common one-wayness security of $\texttt{Hash}_\texttt{P}$ to ensure security against an attacker that does not know the according salts. Note that one-wayness from [5] is called pre-image resistance here to separate it from the common one-wayness.

One-wayness: We use the common definition for one-wayness with syntax appropriate for the used definition of password hashing. A password hashing protocol Π is a one-way function if for all PPT algorithms A, there exists a negligible function $\varepsilon(\cdot)$ such that for $\text{pw}' \leftarrow A^{Hash_P(\cdot)}(\texttt{p}_\texttt{P})$: $\Pr[\texttt{Finalise}(i, \text{pw}') = 1] \leq \varepsilon(\lambda)$, where $Hash_P(\text{pw})$ returns $H \leftarrow \texttt{Hash}_\texttt{P}(\texttt{p}_\texttt{P}, \text{pw}, r)$ for $s_H \leftarrow \texttt{PHSalt}(\texttt{p}_\texttt{P})$, $s_P \leftarrow \texttt{PPHSalt}(\texttt{p}_\texttt{P})$ and global parameters $\texttt{p}_\texttt{P} \leftarrow \texttt{PSetup}(\lambda)$ on the i-th invocation, and stores $T[i] \leftarrow \text{pw}$. `Finalise` returns 1 if $T[i] = \text{pw}'$, otherwise 0.

4.1 Instantiation based on Pedersen commitments

We introduce a new password hashing scheme based on Pedersen commitments. The password hashing scheme $\Pi = (\texttt{PSetup}, \texttt{PPHSalt}, \texttt{PPreHash}, \texttt{PHSalt}, \texttt{PHash})$

is a modified Pedersen commitment with a random generator used for the message. In particular $\mathtt{p_P} = (p, g, h, \lambda) \leftarrow \mathtt{PSetup}(\lambda)$, $\mathbb{Z}_p \ni_R s_P \leftarrow \mathtt{PPHSalt}(\mathtt{p_P})$, $\mathbb{Z}_p \ni_R s_H \leftarrow \mathtt{PPHSalt}(\mathtt{p_P})$, $P = g^{s_P \cdot \mathrm{pw}} \leftarrow \mathtt{PPreHash}(\mathtt{p_P}, \mathrm{pw}, s_P)$ and $H = Ph^{s_H} \leftarrow \mathtt{PHash}(\mathtt{p_P}, P, s_H)$.

Security One-wayness of Π follows immediately from the hiding property of the Pedersen commitment. Pre-image resistance, second pre-image resistance and entropy preserving properties from Appendix A follow from the group properties of the used cyclic group of prime-order p with generators g and h. We use g_P for g^{s_P} in the following.

- The *second pre-image resistance* holds since g_P is a generator, i.e. $g_P^{\mathrm{pw}} h^{s_H} = P' h^{s_H}$ with adversarial generated P' can only be true iff $P' = g_P^{\mathrm{pw}}$.
- The *pre-hash entropy* and *hash entropy* preservation hold since g_P is a generator such that for every (P, s_P) the pre-hash entropy attacker chooses, it holds that $\Pr[P = g_P^{\mathrm{pw}}] \leq 2^{-\beta} + \varepsilon(\lambda)$, and for every (H, s_H) the hash entropy attacker chooses it holds that $\Pr[H = g_P^{\mathrm{pw}} h^{s_H}] \leq 2^{-\beta} + \varepsilon(\lambda)$ for a random $\mathrm{pw} \in_R \mathcal{D}$.
- The *pre-image resistance* holds since the attacker has to compute 2^{β} `PPreHash` and 2^{β} `PHash` values to produce a pre-hash $P = g_P^{\mathrm{pw}}$ with $H = Ph^{s_H}$, which corresponds to a brute-force attack, i.e. $\Pr[\mathtt{Finalise}(i, P) = 1] \leq 2^{-\beta} \frac{\alpha t}{t_{\mathtt{PHash}} t_{\mathtt{PPreHash}}} + \varepsilon(\lambda)$. Note that salts s_P and s_H are randomly chosen on every $Hash_P$ invocation such that the probability for collisions is negligible.

5 Password Registration with ZKPPC

The following concept of Zero-Knowledge Password Policy Checks (ZKPPC) enables clients to prove the compliance of chosen passwords with password policies without password disclosure. A ZKPPC protocol will be executed as part of password registration protocols. We further present a general framework for ZKPPC and one concrete instantiation based on Pedersen commitments and corresponding proofs.

5.1 Zero-Knowledge Password Policy Check

A ZKPPC protocol is an interactive protocol between a client A and a server B, such that the server accepts (A, H) with verifier $H \leftarrow \mathtt{Hash_P}(\mathrm{pw}, r)$ for password pw if and only if the client knows (pw, r), and the password pw complies with the server's password policy, i.e. $f(\mathrm{pw}) = \mathtt{true}$. A PPC protocol can be seen as a proof of knowledge of pw and r such that $H \leftarrow \mathtt{Hash_P}(\mathrm{pw}, r)$ and $f(\mathrm{pw}) = \mathtt{true}$. A ZKPPC protocol is thus a special zero-knowledge proof of knowledge. If the client's prospective password is not policy conform, i.e. $f(\mathrm{pw}) = \mathtt{false}$, the server rejects the setup process.

Definition 1 (PPC). *For password hashing function* $\Pi = (\mathtt{PSetup}, \mathtt{PPHSalt}, \mathtt{PPreHash}, \mathtt{PHSalt}, \mathtt{PHash})$ *and password policy* f *a PPC protocol is an interactive proof of knowledge between a client (prover) and a server (verifier)*

$$\mathsf{PoK}\{(\alpha,\rho):\ f(\alpha)=\texttt{true}\wedge H=\texttt{Hash}_\texttt{P}(\alpha,\rho)\}.$$

PPC is a ZKPPC if PoK is a zero-knowledge proof of knowledge. The security properties of the proof of knowledge PoK can be translated as follows: For any honest server B with policy f and honest client A the server accepts the client's password pw if and only if $f(\text{pw})=\texttt{true}$ (Completeness). For any honest server B with policy f and (malicious) client A on input of H with $f(\text{pw})=\texttt{true}$ and $H\leftarrow\texttt{Hash}_\texttt{P}(\text{pw},r)$, the server outputs 0 with overwhelming probability, i.e. rejects (A,H) (Soundness). In other words, there exists an efficient extractor Ext that extracts (pw,r) from any (malicious) client A that can convince an honest server to accept (A,H). A PPC protocol is a ZKPPC protocol if there exists a simulator $\mathsf{Sim}(f)$ for every H with $f(\text{pw})=\texttt{true}$ and $H\leftarrow\texttt{Hash}_\texttt{P}(\text{pw};r)$ that can produce a view that is indistinguishable from the view of a possibly malicious server B, interacting with a client A (Zero-Knowledge).

5.2 General Framework for ZKPPC Protocols

We present a general framework to construct ZKPPC protocols from set membership proofs, commitments and password hashing protocols. The server's policy $f=(R,m,n)$ is given as a regular expression R on Σ', a lower limit m and an upper limit n, as introduced in Section 3.2. Let R_j denote the j-th element in the regular expression R, i.e. the ASCII subset defined in R_j. Thus every element R_j from the regular expression is linked to a specific set Ω that can be tested in a set membership proof. Let $\mathsf{SMP}(c,r,\Omega)$ denote a proof of set membership that character c from password pw is in set Ω using randomness r. To prove membership of a character c_i at position i in password pw we have to build Ω such that it contains all possible values for encoded c_i. We therefore define a function $E'(\Omega',n)$ that computes a set Ω containing all $c_k\in\Omega'$ for regular expression R_j and all $c_{k,l}=95^l c_k$ for all $l\in\{1,\dots,n-1\}$. This leads to four sets $\Omega_d,\Omega,s,\Omega_u,\Omega_l$ for the four ASCII subsets d,s,u,l and a fifth set Ω_σ for all characters, i.e. $\Omega'=\Sigma$.

The ZKPPC framework is depicted in Figure 1. Note that figure and description are simplified in favour of readability. We omit exact specification of password and character encoding, which can be deduced from Section 3, i.e. password hashing and character commitments are performed on position-specific encoded characters. Both parties have password hashing parameters $\texttt{p}_\texttt{P}\leftarrow\texttt{PSetup}(\lambda)$, commitment parameters $\texttt{p}_\texttt{C}\leftarrow\texttt{CSetup}(\lambda)$, and policy f as common input. The client starts the protocol by choosing a password pw and computing the password verifier $H\leftarrow\texttt{Hash}_\texttt{P}(\text{pw},r_H)$. Additionally, the client computes a public value $g_p=\phi(s_P)$ from the pre-hash salt using a one-way function ϕ, and sends it together with H and his identifier A to the server. Then, he computes commitments $C_i\leftarrow\texttt{Com}(c_i;r_i)$ for all characters in his password pw. Server and client now run zero-knowledge set membership proofs $\mathsf{SMP}(c_i,r_i,\Omega_x)$ for all characters $c_i\in\text{pw}$, proving $c_i\in\Omega_c$ for all $R_j\in R$ using appropriate sets Ω_x for $x\in\Sigma'=\{d,s,u,l\}$. The remaining characters in pw, i.e. characters that are not necessary to fulfil the regular expression R, are then proven to be

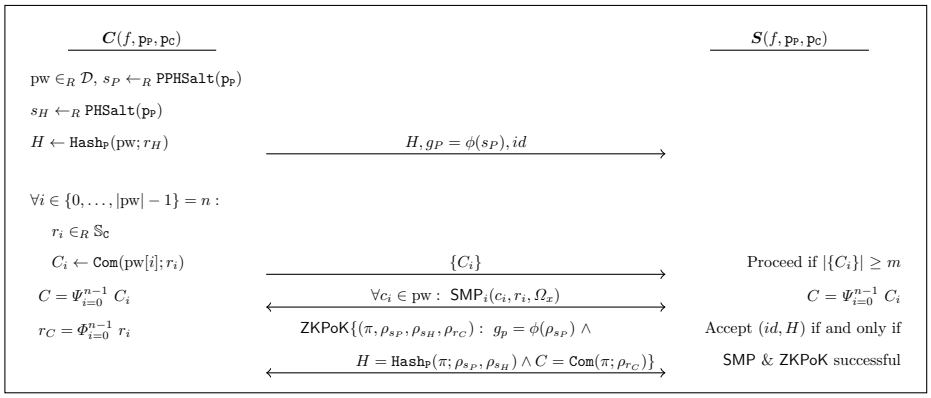

Fig. 1: A Framework for Zero-Knowledge Password Policy Checks

from Ω_σ by running $\mathsf{SMP}_i(c_i, r_i, \Omega_\sigma)$. The SMPs thus prove that the password pw used to generate commitments C_i, is policy conform and chosen from the dictionary of printable ASCII characters. When all set membership proofs are successful, the server is convinced that the characters in commitments C_i are from the requested sets, i.e. that the password in the combined commitment $C = \Psi_{i=0}^{n-1} C_i = \mathtt{Com}(\mathrm{pw}, \Phi_{i=0}^{n-1} r_i)$, is policy conform. However, the server still does not know whether the client actually knows the password in H and if the password committed to in H is the same as the one used in C. Therefore, server and client execute a zero-knowledge proof of knowledge that proves to the server that the client knows pw, used to generate H and C, and the according randomness. If this proof and all set membership proofs are validated by the server, the latter accepts the password verifier H for A.

Definition 1 considers a policy f in general. However, it seems impossible to prove anything on the general case of f without considering an actual definition of the regular expression R and how the verification $f(\mathrm{pw}) = \mathtt{true}$ is implemented. We thus specify the proof statement in our framework for ZKPPC protocols as

$$\mathsf{PoK}\{(\pi, \rho_H, \{\rho_i\}_{i\in[0,n-1]}) : C_i = \mathtt{Com}(\pi_i, \rho_i) \wedge H = \mathtt{Hash_P}(\pi, \rho_H) \\ \wedge \Psi_{i=0}^{n-1} C_i = \mathtt{Com}(\pi, \Phi_{i=0}^{n-1} \rho_i) \wedge \pi_i \in \Omega_x\},$$

where Ω_x is the set specified in R for π_i. This is the adaptation of Definition 1 to our construction using set membership proofs and commitments, i.e. $f(\mathrm{pw}) = \mathtt{true}$ is interpreted as $\mathsf{PoK}\{(\pi, \{\rho_i\}_{i\in[0,n-1]}) : C_i = \mathtt{Com}(\pi_i, \rho_i) \wedge \pi_i \in \mathcal{D}_J\}$ for all $i \in 0, \ldots, n-1$ and sets Ω_x from regular expression R.

Theorem 1. *If* $\mathtt{C} = (\mathtt{CSetup}, \mathtt{Com})$ *is a homomorphic commitment scheme,* $\Pi = (\mathtt{PSetup}, \mathtt{PPHSalt}, \mathtt{PPreHash}, \mathtt{PHSalt}, \mathtt{PHash})$ *a one-way password hashing scheme,* ϕ *a one-way function,* SMP *a zero-knowledge set membership proof and* ZKPoK *a zero-knowledge proof of knowledge, then the framework from Figure 1 is a Zero-Knowledge PPC protocol.*

Proof. *Completeness* of the protocol follows by inspection. Let Ext denote a successful ZKPPC extractor. To prove *soundness* we have to show how to build extractors for zero-knowledge set membership proofs SMP_i and zero-knowledge proof ZKPoK from Ext. An extractor SMP_i outputs witness (pw_i, r_i) for commitment C_i and the ZKPoK extractor outputs $(\mathrm{pw}, s_P, r_H, r_C)$ for a password verifier H, $\phi(s_P)$ and commitment C. Since Ext outputs $(\mathrm{pw}, s_P, r_H, \{r_i\}_{i\in 0,\ldots,n-1})$ building extractors for all SMP_i is straightforward by taking the i-th character from pw and r_i. To build a successful ZKPoK extractor we just output $(\mathrm{pw}, s_P, r_H, \Phi_{i=0}^{n-1} r_i)$.

To prove the *zero-knowledge* property we have to build a simulator Sim for the ZKPPC framework. Let V denote the view of a possibly malicious server after interaction with an honest user. We construct a simulator Sim that outputs a view V', indistinguishable from V. Using simulators Sim_i of set membership proofs SMP_i and Sim' of the final zero-knowledge proof, it is easy to see that the resulting view V' is indistinguishable from V. □

Remark 1. We use set membership proofs in our construction to verify the password policy. Depending on the maximal password length and the complexity of the policy, using range proofs instead of set membership proofs, may be more efficient. However, note that passwords are usually rather short and policies not too complex (e.g. at least one symbol and one upper and lower case letter) such that set membership proofs will be sufficiently efficient in most cases. However, complexity of the framework is clearly dominated by the complexity of the set membership proofs SMP_i, which mainly depends on the upper bound n on the password length. See Section 5.4 for further performance discussions with the instantiation from the following section in mind.

Password Registration using ZKPPC In order to complete the password registration protocol based on ZKPPC, the randomness r that was used by the client to compute the password hash needs to be securely transmitted to the server. This step is not part of the ZKPPC protocol and is performed as a final step in the password registration process. That is, the complete password registration protocol based on ZKPPC is executed over a secure channel and proceeds as follows: On invocation the server sends a password policy f to the client. After proving policy conformity and knowledge of pw and r_H using the ZKPPC protocol, the client sends the randomness $r_H = (s_P, s_H)$ used in the hash computation of H to the server. The server accepts the password setup and stores (A, H, r_H) in its database if and only if the ZKPPC protocol for (A, H) was successful.

Remark 2. The information stored on the server side at the end of a successful password registration consists of the password hash and the salt. This is essentially what servers store with the current state-of-the-art approach mentioned in the introduction. A malicious attacker who breaks-in into the server would still have to perform an offline dictionary attack to recover passwords. The benefit of our approach over the state-of-the-art is that the password is never transmitted to the server.

5.3 Instantiation

To instantiate the ZKPPC framework, we use the Pedersen commitment scheme and set membership proof from Section 2. Further, we use the instantiation of our randomised password hashing protocol from Section 4. Note that parameters chosen in $\texttt{CSetup}(\lambda)$ and $\texttt{PSetup}(\lambda)$ have to be identical, i.e. only one of the setup algorithms is necessary to generate group elements g and h as well as prime p. That leaves us to show how to build the zero-knowledge proof of knowledge ZKPoK and definition of utility functions. Definition of combining functions Ψ and Φ are straight forward, i.e. Ψ is multiplication such that $C = \prod_{i=0}^{n-1} C_i$ and Φ is addition such that $r_C = \sum_{i=0}^{n-1} r_i$. Further, the function ϕ to compute a public value from the pre-hash salt s_P is defined as $g_P = g^{s_P}$.

Building zero-knowledge proof $\mathsf{ZKPoK}\{(\pi, \rho_{s_P}, \rho_{s_H}, \rho_{r_C}) : g_p = g^{\rho_{s_P}} \wedge H = \texttt{Hash}_\texttt{P}(\pi; \rho_{v1}, \rho_{v2}) \wedge C = \texttt{Com}(\pi; \rho_{r_C})\}$ is straightforward. Considering that the used $\texttt{Hash}_\texttt{P}$ function and commitment scheme $\texttt{Com}$ are essentially Pedersen commitments, the zero-knowledge proof is a standard zero-knowledge argument given as follows. The prover chooses k_π, k_{s_P}, k_{s_H} and k_{r_C} from $\mathbb{Z}_p$, computes $t_1 = g_p^{k_\pi} h^{k_{s_H}}$, $t_2 = g^{k_\pi} h^{k_{r_C}}$ and $t_3 = g^{k_{s_P}}$, and sends (t_1, t_2, t_3) to the verifier. On receiving the challenge c from the verifier, the prover computes $a_1 = k_\pi + c\pi \mod p$, $a_2 = k_{s_H} + cs_H \mod p$, $a_2' = k_{r_C} + cr_C \mod p$ and $a_3 = k_{s_P} + c\rho_{s_P} \mod p$, and sends them to the verifier, who checks that $g_p^{a_1} h^{a_2} = t_1 H^c$, $g^{a_1} h^{a_2'} = t_2 C^c$ and $g^{a_3} = t_3 g_p^c$. Completeness of the zero-knowledge proof follows by inspection. To show soundness we build an extractor Ext, extracting a witness $(\text{pw}, s_P, s_H, r_C)$ from interactions with a malicious prover, i.e. from two transcripts $\{H, C, g_P, t_1, t_2, t_3, c, c^*, a_1, a_1^*, a_2, a_2^*, a_2', a_2'^*, a_3, a_3^*\}$. It is easy to see that this allows to extract $(\text{pw}, s_P, s_H, r_C)$ if $c - c^*$ is invertible in $\mathbb{Z}_p$, i.e. $\text{pw} = (a_1 - a_1^*)/(c - c^*)$, $s_P = (a_3 - a_3^*)/(c - c^*)$, $s_H = (a_2 - a_2^*)/(c - c^*)$ and $r_C = (a_2' - a_2'^*)/(c - c^*)$. To prove the honest verifier zero-knowledge property we construct a simulator Sim, generating a view V' that is indistinguishable from a view V of an honest verifier after interaction with an honest prover. Sim therefore chooses $c, a_1, a_2, a_2', a_3 \in_R \mathbb{Z}_p$ and computes $t_1 = g^{a_1} h^{a_2} H^{-c}$, $t_2 = g^{a_1} h^{a_2'} C^{-c}$ and $t_3 = g^{a_3} g_p^{-c}$. The simulator's view $V' = \{H, C, g_p, t_1, t_2, t_3, c, a_1, a_2, a_2', a_3\}$ is then indistinguishable from V. The same remarks regarding honest verifier and general zero-knowledge as for the set membership proof apply here. Security of the instantiation of Figure 1 follows from Theorem 1 considering that all building blocks fulfil our requirements there.

5.4 Performance and Discussion

Runtime and communication complexity of the framework and its instantiation are dominated by the set membership proof, i.e. the computation, exchange and verification of the signature sets $\{\Xi_i\}$, which grow linearly in the upper bound n of the password length. The set $\{\Xi_j\}$ is of size $z(10n + 52n + 32n + 94n)$ for signature length z in bytes (not considering overhead for the encoding) if the policy uses all four available sets. Considering a 512-bit elliptic curve this results in $|\{\Xi_j\}| = 24064n$ bytes, e.g., $n = 10$ leads to 235 kB and $n =$

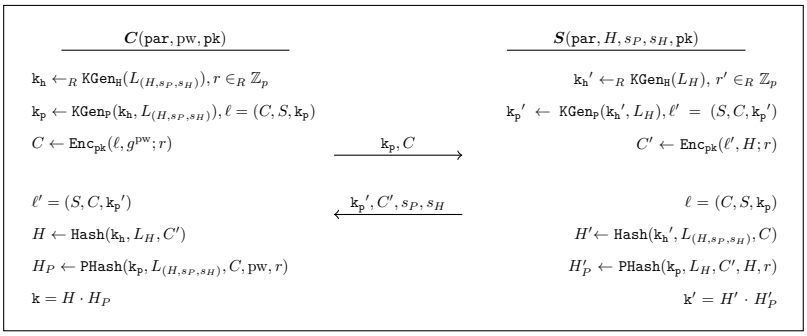

Fig. 2: VPAKE Framework [5]

20 generates 470 kB. Runtime scales accordingly and thus dominates the total execution time of the protocol. It would therefore be interesting to use more efficient set membership proofs, or deploy a different encoding that allows to use more efficient range proofs. With the used encoding it is not possible to use range proofs efficiently. This is due to the fact that encoded character sets are not continuous. For example, the set Ω of digits for $n = 2$ is given by $\Omega = \{16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 1410, 1520, 1615, 1710, 1805, 1900, 1995, 2090, 2185, 2280, 2375\}$. Another possibility to reduce message size and computation time would be not to verify characters that are not relevant for the policy, i.e. only perform set membership proofs for the character sets specified in the policy and allow arbitrary characters for the remaining ones.

Considering that password set up is not performed regularly and usually involves a longer process of filling sign-up forms that can be used to perform the policy check in the background, the proposed instantiation is sufficiently efficient for reasonable limits on the password length.

6 A Suitable VPAKE Protocol for ZKPPC-set Passwords

We now focus on suitable VPAKE protocols that can be used with the password verification information obtained from our ZKPPC protocol. Benhamouda and Pointcheval [5] recently proposed two new security models based on the BPR model that deal with VPAKE protocols and related passwords. We recall their related password VPAKE model with adaptations to our definition of randomised password hashing in Appendix C. This allows us to build VPAKE protocols without relying on idealised assumptions for the security proof while preserving round complexity. The only other security model for VPAKE protocols is given in the UC framework [8] and due to Gentry, MacKenzie and Ramzan [14]. However, idealised models are necessary to reach security in their model.

Therefore, we use the model from [5] and give a new instantiation of the generic VPAKE construction from [5] in the standard model with common reference string. We recall the generic VPAKE construction in Figure 2. Both parties

have parameters $\mathtt{par}$ and public key $\mathtt{pk}$ as common input. The client further knows a password pw, and the server has a password verifier H with according randomness s_P and s_H, set up with the password registration protocol from Section 5.3. To instantiate the framework it is enough to define suitable smooth projective hash functions, an encryption scheme, and a password hashing scheme. We use the previously defined password hashing function and labelled Cramer-Shoup encryptions. That leaves us with the definition of appropriate SPHFs. We need SPHFs for the following two languages $L_{(H,s_P,s_H)} = \{(\ell, c) \mid \exists \mathrm{pw}, r : C = \mathtt{Enc}^{\mathtt{CS}}_{\mathtt{pk}}(\ell, g^{\mathrm{pw}}; r) \wedge H = \mathtt{PHash}(\mathtt{p_P}, P, s_H) \wedge P \leftarrow \mathtt{PPreHash}(\mathtt{p_P}, \mathrm{pw}, s_P)\}$ and $L_H = \{(\ell, c) \mid \exists r : C = \mathtt{Enc}^{\mathtt{CS}}_{\mathtt{pk}}(\ell, H; r)\}$. We use the perfectly smooth SPHF for labelled Cramer-Shoup encryptions from Section 2 for L_H. For $L_{(H,s_P,s_H)}$ we introduce a new SPHF:

- $\mathtt{KGen_H}(L_{(H,s_P,s_H)})$ choose five random elements $\mathtt{k_h} = (\eta_1, \eta_2, \theta, \mu, \nu) \in_R \mathbb{Z}_p^{1\times 5}$.
- $\mathtt{KGen_P}(\mathtt{k_h}, L_{(H,s_P,s_H)})$ on input of language $L_{(H,s_P,s_H)}$ and hashing key $\mathtt{k_h}$ returns the projection key $\mathtt{k_p} = (g_1^{\eta_1} g_2^{\theta} h^{\mu} c^{\nu}, g_1^{\eta_2} d^{\nu}, g_1^{\mu})$.
- $\mathtt{Hash}(\mathtt{k_h}, L_{(H,s_P,s_H)}, C)$ on input of a ciphertext C, language $L_{(H,s_P,s_H)}$ and hashing key $\mathtt{k_h}$ returns the hash value $h = u_1^{\eta_1+\xi\eta_2} u_2^{\theta} [e/(H/h^{s_H})]^{\mu} v^{\nu}$.
- $\mathtt{ProjHash}(\mathtt{k_p}, L_{(H,s_P,s_H)}, C, \mathrm{pw}, r)$ on input of projection key $\mathtt{k_p}$, language $L_{(H,s_P,s_H)}$, ciphertext C, password pw, and randomness r returns the hash value $h = (\mathtt{k_{p_1}} \mathtt{k_{p_2}}^{\xi})^r \mathtt{k_{p_3}}^{\mathrm{pw}-s_P \mathrm{pw}} = g_1^{\eta_1 r} g_2^{\theta r} h^{\mu r} c^{\nu r} g_1^{\eta_2 \xi r} d^{\nu \xi r} g_1^{\mu(\mathrm{pw}-s_P \mathrm{pw})}$.

Security of the SPHF for $L_{(H,s_P,s_H)}$ is straight forward as the only change to the aforementioned SPHF for L_H is $\mathtt{k_{p_3}}$. Correctness of the new SPHF follows by inspection. In the following we show its smoothness. Let pw denote the client's password, $H \leftarrow \mathtt{Hash_P}(\mathrm{pw}, r)$ with $r = (s_P, s_H)$ the according hash value and $C = (u_1, u_2, e, v, \ell)$ a word not in $L_{(H,s_P,s_H)}$, i.e. $C \leftarrow \mathtt{Enc}^{\mathtt{CS}}_{\mathtt{pk}}(\ell, g^{\mathrm{pw}'}; r)$ with $\mathrm{pw}' \neq \mathrm{pw}$. It therefore follows that $(u_1, u_1^{\xi}, u_2, e/(H/h^{s_H}), v) \neq (g_1^r, g_1^{r\xi}, g_2^r, g_1^{\mathrm{pw}-s_P \mathrm{pw}} h^r, (cd^{\xi})^r)$ with overwhelming probability for all $(r, r\xi) \in \mathbb{Z}_p^2$ considering the second pre-image resistance of $\mathtt{Hash_P}$. The hash value $u_1^{\eta_1} u_1^{\xi\eta_2} u_2^{\theta} [e/(Hh^{-s_H})]^{\mu} v^{\nu}$ is then uniformly distributed in G since the elements of the projection key are linearly independent from the hash value for $z = \mathrm{pw}' - s_P \mathrm{pw} \neq 0$.

7 Conclusion

In this work we proposed a new protocol class called password policy checker (PPC), a special proof of knowledge protocol, that allows servers to validate client passwords against a policy without disclosing the password. Zero-Knowledge PPC protocols neither leak the password to the server, nor allow a client to register a non-policy conform password. We defined a security model, a general framework for ZKPPC protocols, and a concrete construction based on Pedersen commitments. We made use of a new structure-preserving encoding for character strings (passwords) and a formal model for password policies. Password verifiers registered through ZKPPC protocols can be used in verifier-based PAKE. We proposed a new VPAKE protocol in the standard model that can work with password verifiers that were set up using our ZKPPC construction.

References

1. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT'00*, volume 1807 of *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*, pages 139–155, Berlin, Heidelberg, 2000. Springer-Verlag. 2
2. S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols SecureAgainst Dictionary Attacks. In *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on*, SP'92, page 72, Washington, DC, USA, 1992. IEEE Computer Society. 2
3. S. M. Bellovin and M. Merritt. Augmented Encrypted Key Exchange: A Password-Based Protocol Secure against Dictionary Attacks and Password File Compromise. In *CCS'93*, ACM Conference on Computer and Communications Security, pages 244–250. ACM, 1993. 3
4. F. Benhamouda, O. Blazy, C. Chevalier, D. Pointcheval, and D. Vergnaud. New smooth projective hash functions and one-round authenticated key exchange. Cryptology ePrint Archive, Report 2013/034, 2013. http://eprint.iacr.org/. 6
5. F. Benhamouda and D. Pointcheval. Verifier-Based Password-Authenticated Key Exchange: New Models and Constructions. *IACR Cryptology ePrint Archive*, 2013:833, 2013. 3, 8, 9, 15, 18, 19
6. D. Boneh and X. Boyen. Short Signatures Without Random Oracles. In *EUROCRYPT'04*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer-Verlag, 2004. 5
7. J. Camenisch, R. Chaabouni, and A. Shelat. Efficient Protocols for Set Membership and Range Proofs. In *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 234–252. Springer-Verlag, 2008. 5
8. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, FOCS'01, page 136, Washington, DC, USA, 2001. IEEE Computer Society. 15
9. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally Composable Password-Based Key Exchange. In *Proceedings of the 24th annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'05, pages 404–421, Berlin, Heidelberg, 2005. Springer-Verlag. 2
10. R. Cramer, I. Damgård, and P. D. MacKenzie. Efficient Zero-Knowledge Proofs of Knowledge Without Intractability Assumptions. In *PKC'00*, volume 1751 of *Lecture Notes in Computer Science*, pages 354–373. Springer-Verlag, 2000. 6
11. R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In *CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer-Verlag, 1998. 6
12. Dan Goodin. Hack of Cupid Media dating website exposes 42 million plaintext passwords. http://arstechnica.com/security/2013/11/hack-of-cupid-media-dating-website-exposes-42-million-plaintext-passwords/, 2014. Accessed: 01/04/2014. 2
13. A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1986. 6
14. 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 *Lecture Notes in Computer Science*, pages 142–159. Springer, 2006. 3, 15

15. 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. 2
16. B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898 (Informational), Sept. 2000. 2
17. Nik Cubrilovic. RockYou Hack: From Bad To Worse. http://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/, 2014. Accessed: 01/04/2014. 2
18. T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, 1991. 5
19. D. Pointcheval. Password-based authenticated key exchange. In *Proceedings of the 15th international conference on Practice and Theory in Public Key Cryptography*, PKC'12, pages 390–397, Berlin, Heidelberg, 2012. Springer-Verlag. 2
20. N. Provos and D. Mazières. A Future-Adaptable Password Scheme. In *USENIX Annual Technical Conference, FREENIX Track*, pages 81–91, 1999. 2
21. Reuters. Trove of Adobe user data found on Web after breach: security firm. http://www.reuters.com/article/2013/11/07/us-adobe-cyberattack-idUSBRE9A61D220131107, 2014. Accessed: 01/04/2014. 2
22. Thomson Reuters. Microsoft India store down after hackers take user data. http://ca.reuters.com/article/technologyNews/idCATRE81C0E120120213, 2014. Accessed: 01/04/2014. 2
23. M. S. Turan, E. Barker, W. Burr, , and L. Chen. Recommendation for password-based key derivation. *NIST Special Publication 800-132*, 2010. 2
24. B. Ur, P. G. Kelley, S. Komanduri, J. Lee, M. Maass, M. L. Mazurek, T. Passaro, R. Shay, T. Vidas, L. Bauer, N. Christin, and L. F. Cranor. How Does Your Password Measure Up? The Effect of Strength Meters on Password Creation. In *Security'12*, Proceedings of the 21st USENIX Conference on Security Symposium, pages 5–5, Berkeley, CA, USA, 2012. USENIX Association. 2

A Password Hashing

We recall security definitions for password hashing from [5] , i.e. (second) pre-image resistance and entropy preservation in $\mathtt{PPreHash}$ and $\mathtt{PHash}$, working with our extended notion of password hashing.

Second Pre-Image Resistance A password hashing protocol Π is second pre-image resistant if for all PPT algorithms A there exists a negligible function $\varepsilon(\cdot)$ such that for $P' \leftarrow A(\mathtt{p_P}, P, s_P, s_H)$

$$\Pr[P' \neq P \wedge \mathtt{PHash}(\mathtt{p_P}, P, s_H) = \mathtt{PHash}(\mathtt{p_P}, P', s_H)] \leq \varepsilon(\lambda),$$

with $\mathtt{p_P} \leftarrow \mathtt{PSetup}(\lambda), s_P \leftarrow \mathtt{PPHSalt}(\mathtt{p_P}), s_H \leftarrow \mathtt{PHSalt}(\mathtt{p_P})$ and $P \leftarrow \mathtt{PPreHash}(\mathtt{p_P}, \mathrm{pw}, s_P)$ for any $\mathrm{pw} \in \mathcal{D}$.

Pre-Image Resistance A password hashing protocol Π is pre-image resistant if for all PPT algorithms A with running time of at most t, there exists a negligible function $\varepsilon(\cdot)$ such that

$$\Pr[(i, P) \leftarrow A^{Hash_P(\cdot)}(\mathtt{p_P});\ \mathtt{Finalise}(i, P) = 1] \leq 2^{-\beta}\frac{\alpha t}{t_{\mathtt{PHash}} t_{\mathtt{PPreHash}}} + \varepsilon(\lambda),$$

for small α, where $Hash_P(\text{pw})$ returns (H, s_P, s_H) with $H \leftarrow \texttt{PHash}(\texttt{p}_\texttt{P}, P, s_H)$, $T[i] \leftarrow \texttt{PPreHash}(\texttt{p}_\texttt{P}, \text{pw}, s_P)$, $s_H \leftarrow \texttt{PHSalt}(\texttt{p}_\texttt{P})$, $s_P \leftarrow \texttt{PPHSalt}(\texttt{p}_\texttt{P})$ and global parameters $\texttt{p}_\texttt{P} \leftarrow \texttt{PSetup}(\lambda)$ on the i-th invocation. `Finalise` returns 1 if $T[i] = P$, otherwise 0. Let $t_{\texttt{PHash}}$ denote the running time of `PHash` and $t_{\texttt{PPreHash}}$ the running time of `PPreHash`.

Pre-hash entropy preservation For all polynomial time samplable dictionaries $\mathcal{D}$ with min-entropy β, and any PPT algorithm A, there exists a negligible function $\varepsilon(\lambda)$ such that for $(P, s_P) \leftarrow A(\texttt{p}_\texttt{P})$ with $\texttt{p}_\texttt{P} \leftarrow \texttt{PSetup}(\lambda)$ and random password $\text{pw} \in_R \mathcal{D}$:

$$\Pr[s_P \in \mathbb{S}_P \wedge P = \texttt{PPreHash}(\texttt{p}_\texttt{P}, \text{pw}, s_P)] \leq 2^{-\beta} + \varepsilon(\lambda).$$

Entropy preservation For all polynomial time samplable dictionaries $\mathcal{D}$ with min-entropy β, and any PPT algorithm A, there exists a negligible function $\varepsilon(\lambda)$ such that for $(H, s_P, s_H) \leftarrow A(\texttt{p}_\texttt{P})$

$$\Pr[s_P \in \mathbb{S}_P \wedge s_H \in \mathbb{S}_H \wedge H = \texttt{Hash}_\texttt{P}(\texttt{p}_\texttt{P}, \text{pw}, r)] \leq 2^{-\beta} + \varepsilon(\lambda),$$

where $\texttt{p}_\texttt{P} \leftarrow \texttt{PSetup}(\lambda)$, $\text{pw} \in_R \mathcal{D}$ and $r = (s_P, s_H)$.

B Efficient Set Membership Proofs

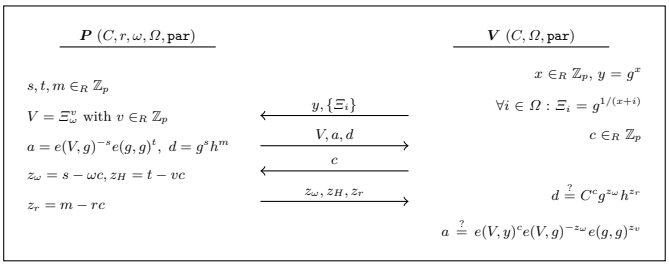

Fig. 3: ZK Set Membership Proof
Client proofs knowledge of element $\omega \in \Omega$ such that $C = g^\omega h^r$

C VPAKE Model

We recall the related password model for verifier-based PAKE protocols proposed by Benhamouda and Pointcheval [5] with modifications to work with our definition of password hashing. We consider clients $C \in \mathcal{C}$, holding password pw_C, and servers $S \in \mathcal{S}$, holding values (H, s_P, s_H) with $H \leftarrow \texttt{PHash}(\texttt{p}_\texttt{P}, P, s_H)$ and $P \leftarrow \texttt{PPreHash}(\texttt{p}_\texttt{P}, \text{pw}_S, s_P)$ for random salts $s_P \leftarrow \texttt{PPHSalt}(\texttt{p}_\texttt{P})$ and $s_H \leftarrow \texttt{PHSalt}(\texttt{p}_\texttt{P})$. Passwords pw_C and pw_S are drawn from dictionary $\mathcal{D}$ with min-entropy β. The adversary has access to the following oracles to interact with protocol participants:

- $\mathsf{Execute}(C, S)$ returns the transcript of the protocol execution between two new instance C_i and S_j. This models passive eavesdropping attacks.
- $\mathsf{Send}(P_i, P'_j, m)$ returns the result of P'_j on input of message m from alleged sender P_i. Invoking Send with an empty message initiates a session between P_i and P'_j. This models active attacks.
- $\mathsf{Corrupt}(S)$ returns the server's secret (H, s_P, s_H). Clients with pw_S are marked as corrupted.

Let b denote a bit chosen prior to every execution of the experiment. Security is modelled with a $\mathsf{Test}(P_i)$ oracle that, on input of participant instance P_i, returns a session key $\mathtt{k}$ chosen as follows:

- If P_i has not computed a session key or P_i is a partnered and corrupted client instance, return $\perp$.
- If P_i is partnered with compatible P'_j and a Test query has been asked for P'_j previously, then return the same session key as for P'_j.
- Otherwise return the real session key of P_i if $b = 1$, and a random session key if $b = 0$.

Two protocol participants are *partnered* if they have matching transcripts, i.e. the recorded transcript of one participant is a subset of the one recorded by the other party. Two protocol participants P, P' are compatible if w.l.o.g. $P \in \mathcal{C}$ and $P' \in \mathcal{S}$, and $\mathtt{PHash}(\mathtt{p_P}, \mathtt{PPreHash}(\mathtt{p_P}, \mathrm{pw}_P, s_P), s_H) = H_{P'}$. To define security we specify a real experiment $\mathsf{Exp}_{\mathtt{Real}}$ and ideal experiment $\mathsf{Exp}_{\mathtt{Ideal}}$. The real world adversary in $\mathsf{Exp}_{\mathtt{Real}}$ has access to the aforementioned oracles and interacts with real participants using passwords chosen according to the dictionary $\mathcal{D}$. The ideal world adversary in $\mathsf{Exp}_{\mathtt{Ideal}}$ interacts with the aforementioned oracles that are modified as follows: $\mathsf{Execute}$ and Send oracles operate with an invalid dummy passwords; Non-trivial Test queries are always answered with a random session key. Additionally, after the adversary returned his guess for bit b, an $\mathtt{Extract}$ function is queried for all participants P_i that have been target of an active attack and have been queried in a non-trivial Test query. The $\mathtt{Extract}$ function on input of a transcript t returns salts s_P and s_H along with a hash value H if P_i is a client and a password pw if P_i is a server. A PAKE protocol Π is secure if for all PPT adversaries $\mathcal{A}$ there exists a negligible function $\varepsilon(\cdot)$ such that

$$\Pr[\mathsf{Exp}_{\Pi,\mathtt{Real}}(\lambda) = 1] \leq \Pr[\mathsf{Exp}_{\Pi,\mathtt{Ideal}}(\lambda) = 1] + \varepsilon(\lambda).$$