Zero-Knowledge Password Policy Checks and Verifier-Based PAKE[*]

Franziskus Kiefer and Mark Manulis

Surrey Centre for Cyber Security
Department of Computing, University of Surrey, UK
mail@franziskuskiefer.de, mark@manulis.eu

Abstract. Zero-Knowledge Password Policy Checks (ZKPPC), introduced in this work, enable blind registration of client passwords at remote servers, i.e., client passwords are never transmitted to the servers. This eliminates the need for trusting servers to securely process and store client passwords. A ZKPPC protocol, executed as part of the registration procedure, allows clients to further prove compliance of chosen passwords with respect to password policies defined by the servers.

The main benefit of ZKPPC-based password registration is that it guarantees that registered passwords never appear 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 a suitable Verifier-based Password Authenticated Key Exchange (VPAKE) protocol.

We give general and concrete constructions of ZKPPC protocols and suitable VPAKE protocols for ASCII-based passwords and policies that are commonly used on the web. To this end we introduce a reversible mapping of ASCII characters to integers that can be used to preserve the structure of the password string and a new randomised password hashing scheme for ASCII-based passwords.

Keywords: Password policies, password registration, authentication, verification, password hashing, ASCII passwords, verifier-based PAKE

1 Introduction

Password policies set by organizations 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 organization 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 [13].

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.

When 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.

[*] An earlier version of this paper was published in the proceedings of the Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II. This is the expanded version with few modifications and is supposed to supersede the proceedings version.

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 or users who are lazy to remember complicated passwords can easily circumvent such script-based verification and register passwords that are not compliant with the policy. 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 policy 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 [14,23] or bcrypt [20], with a random salt to protect against server compromise or re-use attacks, there have been many known cases, e.g. [21,17,22,10], where passwords have been stored in clear and compromised subsequently. The ultimate goal, therefore, is to avoid trusting servers with secure processing and storage of user passwords.

This goal imposes two main challenges: (1) in the registration phase users must be able to choose passwords and prove their policy compliance to a remote server without actually transmitting their passwords, and (2) after the registration phase users must be able to authenticate themselves to the server using their passwords without transmitting them. Interestingly, authentication protocol addressing the second challenge already exist in form of Password-Authenticated Key Exchange (PAKE) protocols, e.g. [2,1,7,19]. PAKE protocols offer authentication and computation of secure session keys in a password-only setting in a way that makes it hard for an active adversary to recover passwords via offline dictionary attacks. Traditional PAKE protocols, however, assume that the password is used in clear on the server sides. To alleviate the threat that passwords are revealed immediately when server's database is compromised, the so-called Verifier-based PAKE (VPAKE) protocols [3,12,5] assume that instead of plain password servers are using some verification information that is derived from the password such that if an attacker breaks into the server and compromises its database it must still execute an expensive offline dictionary attack to recover the plain password. For this reason VPAKE protocols offer a better protection than PAKE. 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 challenge; namely the client must be able to prove that the verification information for VPAKE has been derived from a password that complies with the server's password policy.

Zero-Knowledge Password Policy Checks (ZKPPC) Our first contribution, in Section 5, is the concept of Zero-Knowledge Password Policy Checks (ZKPPC), a new class of protocols that allows servers to perform policy checks on client passwords without ever receiving them in clear. ZKPPC protocols can be used for blind registration of policy-conform passwords and thus solve the aforementioned challenge of password setup where only password verification information is supposed to be stored at the server and where the server cannot be trusted to process passwords securely. We present a security model for ZKPPC, a general ZKPPC framework, and a concrete ZKPPC protocol based on Pedersen commitments. In the construction of ZKPPC protocols we make use of the new randomised password hashing scheme, introduced in Section 4 and the reversible structure-preserving mapping of ASCII-based password strings to integers, introduced in Section 3.

ZKPPC-compliant VPAKE Our second contribution are one-round VPAKE protocols, in Section 6, that can be used with verification information obtained from our blind password registration protocols based on ZKPPC. We design VPAKE protocols based on the framework from [5]. We propose a general VPAKE protocol that can be used in combination with our general ZKPPC framework for ASCII-based passwords and policies and a concrete VPAKE construction that suits particularly well with our ZKPPC-based blind password registration protocol that is based on Pedersen commitments and our randomised password hashing scheme.

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 later use of this verification information on the server side as input to a suitable VPAKE protocol for the purpose of authentication. A client wishing to register its user id 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 client's password complies with server's password policy and is linked to the verification information communicated at the end of the registration phase. This verification information is computed through a randomised password hashing scheme and includes (partial) randomness that was used by the client in the ZKPPC protocol. Plain password is never transmitted to the server and the only way for the server to reveal it is to execute an expensive offline dictionary attack. That is, an honest-but-curios server would have to perform about the same amount of computation to recover plain passwords as an attacker who breaks into that server at any time. The server will be able to recognise and reject any cheating attempt of the client to set up a non-policy conform password, still without learning the latter. In the realization of this concept we apply the following building blocks.

Zero-Knowledge Proofs A *proof of knowledge* PoK between prover P and verifier V for a (public) binary relation $R = \{(C, w)\}$ is denoted $\mathsf{PoK}\{(w) : (C, w) \in R\}$ where w is a secret witness for C. PoK is a *zero-knowledge proof of knowledge* ZKPoK if V is convinced that $(C, w) \in R$ without learning any information about w known by P. More formally, an interactive PoK for $R = \{(C, w)\}$ between P and V is a ZKPoK if the following holds:

- Completeness: For any $(C, w) \in R$, honest verifier $V(C)$ accepts in the interaction with an honest prover $P(C, w)$.
- Soundness: If an honest $V(C)$ accepts in the interaction with a malicious prover $P^*(C)$ then there exists an efficient *knowledge extractor* Ext that extracts a witness w for C from the interaction with $P^*(C)$.
- Zero-Knowledge: For any $(C, w) \in R$ there exists an efficient *simulator* Sim such that the views of a malicious verifier $V(C)$ in interactions with $\mathsf{Sim}(C)$ and an honest prover $P(C, w)$ remain indistinguishable.

Commitments A *commitment scheme* $\mathtt{C} = (\mathtt{CSetup}, \mathtt{Com}, \mathtt{Open})$ contains three polynomial time algorithms and satisfies the following properties:

- Completeness: For all $\mathtt{p}_{\mathtt{C}} \leftarrow \mathtt{CSetup}(\lambda)$, $x \in \mathbb{X}$, $r \in \mathbb{S}$: $x \leftarrow \mathtt{Open}(\mathtt{p}_{\mathtt{C}}, C, d)$ for all $(C, d) \leftarrow \mathtt{Com}(\mathtt{p}_{\mathtt{C}}, x; r))$.
- Binding: For all PPT algorithms A that on input $\mathtt{p}_{\mathtt{C}} \leftarrow \mathtt{CSetup}(\lambda)$ output (C, d, d') there exists a negligible function $\varepsilon(\cdot)$ such that

$$\Pr[x \neq x' \wedge x \leftarrow \mathtt{Open}(\mathtt{p}_{\mathtt{C}}, C, d) \wedge x' \leftarrow \mathtt{Open}(\mathtt{p}_{\mathtt{C}}, C, d')] \leq \varepsilon(\lambda)$$

- Hiding: For all PPT algorithms $A = (A_1, A_2)$ where A_1 on input $\mathtt{p}_{\mathtt{C}} \leftarrow \mathtt{CSetup}(\lambda)$ outputs x_0 and x_1 of the same length and where A_2 on input $(C, d) \leftarrow \mathtt{Com}(\mathtt{p}_{\mathtt{C}}, x_b; r)$ for a random bit $b \in \{0, 1\}$, $r \in \mathbb{S}$ outputs bit b' there exists a negligible function $\varepsilon(\cdot)$ such that $|\Pr[b = b'] - 1/2| \leq \varepsilon(\lambda)$.

A commitment scheme is said to be *(additively) homomorphic* if for all $\mathtt{p}_{\mathtt{C}} \leftarrow \mathtt{CSetup}(\lambda)$, $(C_i, d_i) \leftarrow \mathtt{Com}(\mathtt{p}_{\mathtt{C}}, x_i; r_i)$ with $x_i \in \mathbb{X}$ and $r_i \in \mathbb{S}$ for $i \in 0, \ldots, m$ it holds that $\prod_{i=0}^{m} C_i = \mathtt{Com}(\mathtt{p}_{\mathtt{C}}, \sum_{i=0}^{m} x_i;\ \psi_{i=0}^{m} r_i)$ for some function ψ. We will omit $\mathtt{p}_{\mathtt{C}}$ and d from the notation and write $C \leftarrow \mathtt{Com}(x; r)$ to denote the commitment of x using randomness r.

Pedersen commitments [18] The commitment scheme from [18] is perfectly hiding and additively homomorphic. Its $\texttt{CSetup}(\lambda)$ algorithm outputs (g, h, p, λ), where g and h are generators of a cyclic group G of prime order p of length λ and the discrete logarithm of h with respect to g is unknown. $\texttt{Com}(x; r)$ for $x, r \in \mathbb{Z}_p^*$ outputs $C = g^x h^r$ and $d = (x, r)$. $\texttt{Open}(C, d)$ returns x iff $C = g^x h^r$.

Set Membership Proofs on Committed Values These zero-knowledge proofs can be used to prove that a committed value x is an element of a specific set Ω. Let $C \leftarrow \texttt{Com}(x; r)$ be some commitment of x with randomness r. The corresponding proof for $x \in \Omega$ is defined as $\mathsf{ZKPoK}\{(\xi, \rho) : C \leftarrow \texttt{Com}(\xi; \rho) \wedge \xi \in \Omega\}$. We will use $\mathsf{SMP}(\xi, \rho, \Omega)$ as a shorter notation for this proof.

Labeled Public Key Encryption A *labeled encryption scheme* $\texttt{E} = (\texttt{KGen}, \texttt{Enc}, \texttt{Dec})$ is IND-CCA2 secure for all PPT algorithms $A = (A_1, A_2)$ where A_1 on input pk for $(pk, sk) \leftarrow \texttt{KGen}(\lambda)$ and access to the decryption oracle $\texttt{Dec}(sk, \cdot)$ outputs two messages m_0 and m_1 of equal length and a label ℓ and where A_2 on input $c \leftarrow \texttt{Enc}^{\ell}(pk, m_b; r)$ for a random bit $b \in_R \{0, 1\}$ with access to the decryption oracle outputs bit b' without querying $\texttt{Dec}(sk, (\ell, c))$ there exists a negligible function $\varepsilon(\cdot)$ such that $|\Pr[b' = b] - \frac{1}{2}| \leq \varepsilon(\lambda)$.

Labeled Cramer-Shoup Encryption [9] The labeled CS encryption scheme from [9] is IND-CCA2 secure. Its key generation algorithm $\texttt{KGen}(\lambda)$ outputs $sk = (x_1, x_2, y_1, y_2, z)$ and $pk = (p, g_1, g_2, h, c, d, H_k)$ with $c = g_1^{x_1} g_2^{x_2}, d = g_1^{y_1} g_2^{y_2}, h = g_1^z$, where g_1 and g_2 are generators of a cyclic group G of prime order p of length λ and $H_k : \{0, 1\}^* \mapsto \mathbb{Z}_p^*$ is a hash function. The encryption algorithm $\texttt{Enc}^{\ell}(pk, m; r)$ outputs $C = (u_1, u_2, e, v)$ where $u_1 = g_1^r$, $u_2 = g_2^r$, $e = mh^r$ and $v = (cd^{\xi})^r$ with $\xi = H_k(\ell, u_1, u_2, e)$. The decryption algorithm $\texttt{Dec}^{\ell}(sk, C)$ outputs $m = e/u_1^z$ if $u_1^{x_1 + y_1 \cdot \xi'} u_2^{x_2 + y_2 \cdot \xi'} = v$ with $\xi' = H_k(\ell, u_1, u_2, e)$.

Smooth Projective Hashing (SPHF) Let $L = \{C\}$ denote a language with $L \subset X$ such that $C \in L$ if there exists a witness w for C. A SPHF for $L \subset X$, as defined in [4], consists of the following algorithms:

- $\texttt{HKGen}(L)$ generates a hashing key $\texttt{hk}$ for L.
- $\texttt{PKGen}(\texttt{hk}, L, C)$ derives the projection key $\texttt{hp}$, possibly depending on C.
- $\texttt{Hash}(\texttt{hk}, L, C)$ outputs the hash value h, for any $C \in X$.
- $\texttt{PHash}(\texttt{hp}, L, C, w)$ outputs the hash value h, for any $C \in L$ with witness w.

A SPHF is correct if for all $C \in L$ with witness w: $\texttt{Hash}(\texttt{hk}, L, C) = \texttt{PHash}(\texttt{hp}, L, C, w)$. A SPHF is smooth if for all $C \notin L$, the hash value h is indistinguishable from a random element in G.

SPHF for Labeled CS Ciphertexts [4] Let $L_m = \{(\ell, C) | \exists r, C \leftarrow \texttt{Enc}^{\ell}(pk, m; r)\}$, where $pk = (p, g_1, g_2, h, c, d, H_k)$. Note that $C = (u_1, u_2, e, v)$, where $u_1 = g_1^r$, $u_2 = g_2^r$, $e = mh^r$, and $v = (cd^{\xi})^r$ with $\xi = H_k(\ell, u_1, u_2, e)$. A perfectly smooth SPHF from [4] for L_m is defined as follows:

- $\texttt{HKGen}(L_m)$ generates a hashing key $\texttt{hk} = (\eta_1, \eta_2, \theta, \mu, \nu) \in_R \mathbb{Z}_p^{1 \times 5}$.
- $\texttt{PKGen}(\texttt{hk}, L_m)$ derives the projection key $\texttt{hp} = (\texttt{hp}_1, \texttt{hp}_2) = (g_1^{\eta_1} g_2^{\theta} h^{\mu} c^{\nu}, g_1^{\eta_2} d^{\nu})$.
- $\texttt{Hash}(\texttt{hk}, L_m, C)$ outputs the hash value $h = u_1^{\eta_1 + \xi \eta_2} u_2^{\theta} (e/m)^{\mu} v^{\nu}$.
- $\texttt{ProjHash}(\texttt{hp}, L_m, C, r)$ outputs the hash value $h = (\texttt{hp}_1 \texttt{hp}_2{}^{\xi})^r$.

3 Modeling Passwords and Policies

In the following we model passwords and their dictionaries. Note that password strings are typically mapped to integers before they are processed in cryptographic operations. For our purposes such integer mapping must be able to preserve password structures. In particular, the way a password string is composed from single characters must remain visible from the resulting integer value. As part of password modeling we describe an appropriate encoding scheme that maps password strings defined over the alphabet of printable ASCII characters to integers while preserving their structures. We further model and define password policies as regular expressions over different ASCII character sets.

3.1 Password Strings and Dictionaries

We consider *password strings* pw over the *ASCII alphabet* Σ containing all 94 *printable* ASCII characters.[1] We split $\Sigma = d \cup u \cup l \cup s$ into four subsets:

- set of **digits** $d = [0 - 9]$ (or ASCII codes $[48 - 57]$),
- set of **upper case letters** $u = [A - Z]$ (or ASCII codes $[65 - 90]$)
- set of **lower case letters** $l = [a - z]$ (or ASCII codes $[97 - 122]$)
- set of **symbols** $s =$ [!"#$%&’()*+,-./ :;<=>?@ [\]^_‘ {|}~] (or ASCII codes $[33 - 47, 58 - 64, 91 - 96, 123 - 126]$)

By $\mathcal{D}$ we denote a *general dictionary* containing all strings that can be formed from printable ASCII characters, i.e. all power sets of Σ. A *password string* $pw = (c_0, \ldots, c_{n-1}) \in \Sigma^n \subset \mathcal{D}$ of length n is an ordered set of characters $c_i \in \Sigma$.

3.2 Structure-Preserving Mapping of Password Strings to Integers

In the following we show how a password string pw can be mapped to an integer π for further cryptographic processing in a way that preserves the character composition of pw and makes it possible to efficiently reconstruct pw from π.

Mapping of Password Characters to Integers In order to preserve the character structure of a password string pw upon its mapping to an integer we first define a *character mapping* function $\mathtt{CHRtoINT} : \Sigma \mapsto \mathbb{Z}_{95}$ for any printable ASCII character $c \in \Sigma$ that internally uses its decimal ASCII code $\mathtt{ASCII}(c)$ to output an integer in $\mathbb{Z}_{95}$:

$$\mathtt{CHRtoINT}(c) = \begin{cases} \perp & \text{if } \mathtt{ASCII}(c) < 32 \\ \mathtt{ASCII}(c) - 32 & \text{if } 33 \leq \mathtt{ASCII}(c) \leq 126 \\ \perp & \text{if } 126 < \mathtt{ASCII}(c) \end{cases}$$

Position-Dependent Mapping of Password Characters to Integers A printable ASCII character $c \in \Sigma$ may appear at any position $i \in [0, n-1]$ in a password string $pw \in \Sigma^n$. For every position i we require a different integer to which $c_i \in pw$ can be mapped to. Assuming a reasonable upper bound $n_{\mathtt{max}}$ on the password length n, i.e. $n \leq n_{\mathtt{max}}$, and some *shift base*[2] $b \in \mathbb{N}$, we define four integer sets Ω_x, $x \in \Sigma' = \{d, u, l, s\}$, where d, u, l, s are the identifiers of the four ASCII character subsets that were used to define Σ as follows:

- $\Omega_d = \{b^i \mathtt{CHRtoINT}(c)\}$ for all digits $c \in d$ and $i = 0, \ldots, n_{\mathtt{max}} - 1$. Note that $|\Omega_d| = 10 n_{\mathtt{max}}$.
- $\Omega_u = \{b^i \mathtt{CHRtoINT}(c)\}$ for all upper case letters $c \in u$ and $i = 0, \ldots, n_{\mathtt{max}} - 1$. Note that $|\Omega_u| = 26 n_{\mathtt{max}}$.
- $\Omega_l = \{b^i \mathtt{CHRtoINT}(c)\}$ for all lower case letters $l \in u$ and $i = 0, \ldots, n_{\mathtt{max}} - 1$. Note that $|\Omega_l| = 26 n_{\mathtt{max}}$.
- $\Omega_s = \{b^i \mathtt{CHRtoINT}(c)\}$ for all symbols $c \in s$ and $i = 0, \ldots, n_{\mathtt{max}} - 1$. Note that $|\Omega_s| = 32 n_{\mathtt{max}}$.

Any password character $c_i \in pw$, $i \in [0, n_{\mathtt{max}} - 1]$ can therefore be mapped to one of the four sets Ω_x, $x \in \Sigma'$ with the *position-dependent character mapping* function $\mathtt{CHRtoINT_i} : \Sigma \mapsto \Omega_x$, defined as

$$\mathtt{CHRtoINT_i}(c, i) = b^i \mathtt{CHRtoINT}(c),$$

where the shift base $b \in \mathbb{N}$ is a public constant. We write $\pi_i \leftarrow \mathtt{CHRtoINT_i}(c, i)$ for the integer value of the ith character $c_i \in pw$.

[1] Although we do not consider password strings consisting of other characters, our approach is easily adaptable to UTF-8 and other character sets.

[2] In the proceedings version [16], the shift base was not parameterized and was set to be 95. While this choice guarantees that each password string pw formed of printable ASCII characters will be mapped to a unique integer π it is not sufficient for secure processing of π within the password hashing scheme, as discussed in Remark 4.1.

Mapping of Password Strings to Integers A *password mapping* function $\mathtt{PWDtoINT} : \Sigma^n \mapsto \mathbb{Z}_{b^{n_{\mathtt{max}}}}$ that maps any password string $pw = (c_0, \ldots, c_{n-1}) \in \Sigma^n$ to an integer in a larger set $\mathbb{Z}_{b^{n_{\mathtt{max}}}}$ in a way that preserves the ith position of each character c_i is defined as follows:

$$\mathtt{PWDtoINT}(pw) = \sum_{i=0}^{n-1} b^i \mathtt{CHRtoINT}(c_i) = \sum_{i=0}^{n-1} \mathtt{CHRtoINT_i}(c_i, i) \text{ for } c_i \in pw$$

We will use pw to denote a password string and $\pi \leftarrow \mathtt{PWDtoINT}(pw)$ for its integer value. Note that $\pi = \sum_{i=0}^{n-1} \pi_i$. The mapping computed through $\mathtt{PWDtoINT}$ is injective and reversible. For example, $\pi = 883318$ with $b = 100$ is the integer value of password string $pw = (2, \mathrm{A}, \mathrm{x})$. The string can be recovered by concatenation of $883318 \mod 100 = 18 \widehat{=} 2$ at position 0, $(797353 \mod 100^2) - (797353 \mod 100) = 3300 = 33 \cdot 100^1 \widehat{=}$ A at position 1 and $797353 - (797353 \mod 95^2) = 880000 = 88 \cdot 100^2 \widehat{=}$ x at position 2.

3.3 Password Policies

A *password policy* $f = (R, n_{\mathtt{min}}, n_{\mathtt{max}})$ is modeled using a *regular expression* R over $\Sigma' = \{d, u, l, s\}$, a *minimum length* $n_{\mathtt{min}}$ and a *maximum length* $n_{\mathtt{max}}$ that a password string pw must fulfill.[3] We write $f(pw) = \mathtt{true}$ to indicate that the policy is satisfied by the password string pw. For example,

- $f = (\mathtt{ds}, 6, 10)$ means that pw must have between 6 and 10 characters with at least one digit and one symbol.
- $f = (\mathtt{uss}, 8, 12)$ means that pw must have between 8 and 12 characters with at least one upper-case letter and two symbols.
- $f = (\mathtt{duls}, 8, 16)$ means that pw must have between 8 and 16 characters with at least one character of each type.

Remark 1. Note that in practice password policies do not specify $n_{\mathtt{max}}$. We leave it for the server administrator to decide whether $n_{\mathtt{max}}$ should be mentioned explicitly in f or fixed in the system to allow for all reasonable password lengths.

4 Randomised Password Hashing

A *password hashing* scheme Π that is used to compute password verification information for later use in VPAKE protocols from [5] is defined as follows:

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

In the above syntax the algorithm $\mathtt{PPreHash}$ is *randomised* with a pre-hash salt s_P, which extends the syntax from [5], where $\mathtt{PPreHash}$ is deterministic (and realised in constructions as a random oracle output $\mathcal{H}(pw)$). In contrast we are interested in algebraic constructions of both $\mathtt{PPreHash}$ and $\mathtt{PHash}$ to allow for efficient proofs of knowledge involving pre-hash values P. The randomisation of $\mathtt{PPreHash}$ further increases the complexity of an offline dictionary attack that recovers pw from P since it removes the ability of an attacker to pre-compute pairs (P, pw) and use them directly to recover pw (see also Section 5.4). We write

[3] The way password policies are modeled in this work is suitable for policies that put restrictions on the password length and the nature of password characters. Other types of policies, e.g. search for natural words in a password (cf. Dropbox password-meter, https://tech.dropbox.com/2012/04/zxcvbn-realistic-password-strength-estimation) are currently not supported by our framework and thus left for future work.

$H \leftarrow \mathtt{Hash_P}(pw, r)$ to denote $H \leftarrow \mathtt{PHash}(\mathtt{p_P}, P, s_P, s_H)$ with $P \leftarrow \mathtt{PPreHash}(\mathtt{p_P}, pw, s_P)$, where $r = (s_P, s_H)$ combines the randomness used in PHash and PPreHash. A secure Π must satisfy the following security properties. Note that password-hiding is a new property that is used in ZKPPC to ensure that password hashes H do not leak any information about pw. The remaining four properties are from [5], updated where necessary to account for the randomised PPreHash:

- Password hiding: For all PPT algorithms $A = (A_1, A_2)$ where A_1 on input $\mathtt{p_P} \leftarrow \mathtt{PSetup}(\lambda)$ outputs two equal-length password strings pw_0 and pw_1 and where A_2 on input $H \leftarrow \mathtt{PHash}(\mathtt{p_P}, P, s_P, s_H)$, where $s_H \leftarrow \mathtt{PHSalt}(\mathtt{p_P})$, $s_P \leftarrow \mathtt{PPHSalt}(\mathtt{p_P})$, and $P \leftarrow \mathtt{PPreHash}(\mathtt{p_P}, pw_b, s_P)$ for a random bit $b \in_R \{0,1\}$ outputs bit b' there exists a negligible function $\varepsilon(\cdot)$ such that $|\Pr[b' = b] - \frac{1}{2}| \leq \varepsilon(\lambda)$.
- Pre-image resistance (called tight one-wayness in [5]): For all PPT algorithms A running in time at most t, there exists a negligible function $\varepsilon(\cdot)$ such that

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

  for small α and $t_{\mathtt{PPreHash}}$ being the running time of PPreHash, where $\mathtt{p_P} \leftarrow \mathtt{PSetup}(\lambda)$ and each ith invocation of $Hash_P(\cdot)$ returns (H, s_H) with $H \leftarrow \mathtt{PHash}(\mathtt{p_P}, P, s_P, s_H)$ and stores $T[i] \leftarrow \mathtt{PPreHash}(\mathtt{p_P}, pw, s_P)$, where $s_H \leftarrow \mathtt{PHSalt}(\mathtt{p_P})$, $s_P \leftarrow \mathtt{PPHSalt}(\mathtt{p_P})$, and $pw \in_R \mathcal{D}$. $\mathtt{Finalise}(i, P) = 1$ if $T[i] = P$. (Note that $Hash_P(\cdot)$ does not return s_P.)
- Second pre-image resistance: For all PPT algorithms A there exists a negligible function $\varepsilon(\cdot)$ such that for $P' \leftarrow A(\mathtt{p_P}, P, s_H)$

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

  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 $P \leftarrow \mathtt{PPreHash}\ (\mathtt{p_P}, pw, s_P)$ for any $pw \in \mathcal{D}$.
- 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 $(P, s_P) \leftarrow A(\mathtt{p_P})$ with $\mathtt{p_P} \leftarrow \mathtt{PSetup}(\lambda)$ and random password $pw \in_R \mathcal{D}$:

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

- 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 $(H, s_P, s_H) \leftarrow A(\mathtt{p_P})$

$$\Pr[s_P \in \mathbb{S}_P \wedge s_H \in \mathbb{S}_H \wedge H = \mathtt{Hash_P}(\mathtt{p_P}, pw, s_P, s_H)] \leq 2^{-\beta} + \varepsilon(\lambda),$$

  where $\mathtt{p_P} \leftarrow \mathtt{PSetup}(\lambda)$ and $pw \in_R \mathcal{D}$.

4.1 Randomised Password Hashing from Pedersen Commitments

We introduce a randomised password hashing scheme $\Pi = (\mathtt{PSetup}, \mathtt{PPHSalt}, \mathtt{PPreHash}, \mathtt{PHSalt}, \mathtt{PHash})$ for ASCII-based passwords using Pedersen commitments. We assume that $\pi \leftarrow \mathtt{PWDtoINT}(pw)$ for an appropriate choice of the shift base b (see Remark 4.1) and construct Π as follows:

- $\mathtt{PSetup}(\lambda)$ generates $\mathtt{p_P} = (p, g, h, \lambda)$ where g, h are independent generators of a cyclic group G of prime order p of length λ.
- $\mathtt{PPHSalt}(\mathtt{p_P})$ generates a pre-hash salt $s_P \in_R \mathbb{Z}_p^*$.
- $\mathtt{PPreHash}(\mathtt{p_P}, \pi, s_P)$ outputs the pre-hash value $P = g^{s_P \pi}$.
- $\mathtt{PHSalt}(\mathtt{p_P})$ generates a hash salt $s_H \in_R \mathbb{Z}_p^*$.
- $\mathtt{PHash}(\mathtt{p_P}, P, s_P, s_H)$ outputs hash value $H = (H_1, H_2) = (g^{s_P}, Ph^{s_H})$.

Observe that $H_2 = H_1^{\pi} h^{s_H}$, i.e., H_1 can be seen as a fresh generator that is used to compute the Pedersen commitment H_2. The security properties of our password hashing scheme Π follow from the properties of the underlying cyclic group G and from the security of Pedersen commitments. We argue informally:

- The *password hiding* property of the scheme, assuming that pw_0 and pw_1 are mapped to corresponding integers π_0 and π_1 in $\mathbb{Z}_{b^n}$, is perfect and holds based on the perfect hiding property of the Pedersen commitment scheme. Note that the adversary receives the corresponding hash value $H = (H_1, H_2) = (g^{s_P}, Ph^{s_H})$, where $H_2 = g^{s_P\pi}h^{s_H}$ is a Pedersen commitment on π with respect to two independent bases g^{s_P} and h. The ability of A to distinguish between π_0 and π_1 can thus be turned into the attack on the hiding property of the commitment scheme.
- The *pre-image resistance* holds since s_P and s_H are randomly chosen on every invocation of $\mathit{Hash}_P(\cdot)$ with a negligible probability for a collision and H_2 is a perfectly hiding commitment with bases g^{s_P} and h. Therefore, for any given output $(H = (H_1, H_2), s_H)$ of $\mathit{Hash}_P(\cdot)$, A must perform 2^β exponentiations $H_1^{\pi^*}$, one for each candidate π^*, in order to find $P = H_2h^{-s_H}$. This roughly corresponds to 2^β invocations of `PPreHash`.
- The *second pre-image resistance* holds since H_1 is uniform in G and H_2 is a computationally binding commitment with bases g^{s_P} and h. Note that for any P' generated by A, $H_1^\pi h^{s_H} = P'h^{s_H}$ is true only if $P' = H_1^\pi$.
- The *pre-hash entropy* and *hash entropy* preservation hold since H_1 is a generator of G such that for every (P, s_P) chosen by the pre-hash entropy adversary, $\Pr[P = H_1^\pi] \leq 2^{-\beta} + \varepsilon(\lambda)$, and for every (H, s_H) chosen by the hash entropy adversary, $\Pr[H_2 = H_1^\pi h^{s_H}] \leq 2^{-\beta} + \varepsilon(\lambda)$ for a random $pw \in_R \mathcal{D}$.

Choosing the Shift Base b As Benhamouda and Pointcheval point out in [5], the encoded password (integer) π in $P = H_1^\pi$ can be computed from (H, s_H) in time $O(\sqrt{n})$ for $\pi \in [0, n-1]$, e.g. using variants of the Pollard's kangaroo algorithm, and if n corresponds to the dictionary size $|\mathcal{D}|$ then the above password hashing scheme is no longer pre-image resistant. We observe, however, that the size of n is *larger* than the dictionary size $|\mathcal{D}|$. To see this, first note that

$$n = \sum_{i=0}^{|pw|-1} b^i \cdot 93,$$

which denotes the largest possible value of $\pi \leftarrow$ `PWDtoINT`(pw) obtained from a password string of length $|pw|$ (normalised to 0 being the smallest possible π). In order to see the difference to the dictionary size $|\mathcal{D}|$ we analyse how the password length $|pw|$ impacts the interval of the corresponding integer π: for example, if $|pw| = 1$ then π is in $[1, 94]$; if $|pw| = 2$ then π is in $[b+1, b+94] \cup \cdots \cup [94b+1, 94b+94]$; if $|pw| = 3$ then π is in $[1+b+b^2, 94+b+b^2] \cup \cdots \cup [1+94b+94b^2, 94+94b+94b^2]$, and so on. That is in contrast to the size of the dictionary $\mathcal{D}$ (assuming for now that policy f does not restrict the character choice), which corresponds to the number of $|pw|$-tuples containing characters chosen out of all 94 printable ASCII characters and is therefore given by

$$|\mathcal{D}| = 94^{|pw|}$$

and therefore independent of the shift base b. In order to guarantee the pre-image resistance of the password hashing scheme we need to ensure that $|\mathcal{D}| \approx \sqrt{n}$. This can be achieved by choosing the shift base b such that computing π from $P = H_1^\pi$ becomes as difficult as a brute-force search over the dictionary $\mathcal{D}$. In particular, the shift base b must be chosen such that

$$|\mathcal{D}|^2 \leq \sum_{i=0}^{|pw|-1} b^i \cdot 93. \tag{1}$$

Since the required value of b depends on $|pw|$ as well as policy f it is possible for a specific password policy $f = (R, n_{\mathtt{min}}, n_{\mathtt{max}})$ to compute the optimal value for b using the password length restrictions $n_{\mathtt{min}}, n_{\mathtt{max}}$ and regular expression R, and by this to optimise the performance of the password hashing scheme (and of our ZKPPC protocol) with respect to the given policy. Using dictionary size

$$|\mathcal{D}_f| = 94^{|pw|-|R|} \prod_{i=0}^{|R|-1} |R_i|,$$

for policy f allows to compute an optimal b such that Eq. 1 holds for all policy compliant passwords. If such optimisation is not required then we recommend setting $b = 10^5$, which should be a safe choice for all sensible policies and password lengths (in contrast to $b = 95$ that was used in [16]). Figure 1 depicts relations among password length and optimal shift base b for the general case (without regular expression) and with regular expression $R =$ `duls`. First it shows that $b = 10^5$ is a safe choice. Further, it shows that the regular expression has a significant influence on the choice of an optimal base b, i.e. can be used to tweak performance of a ZKPPC protocol. In the general case we see that base b grows exponentially the shorter the password gets, which is responsible for the relatively large suggested default base $b = 10^5$. With regular expression `duls` in contrast b has to *grow* with increasing password length. This is due to the restriction on the character sets from the regular expression.

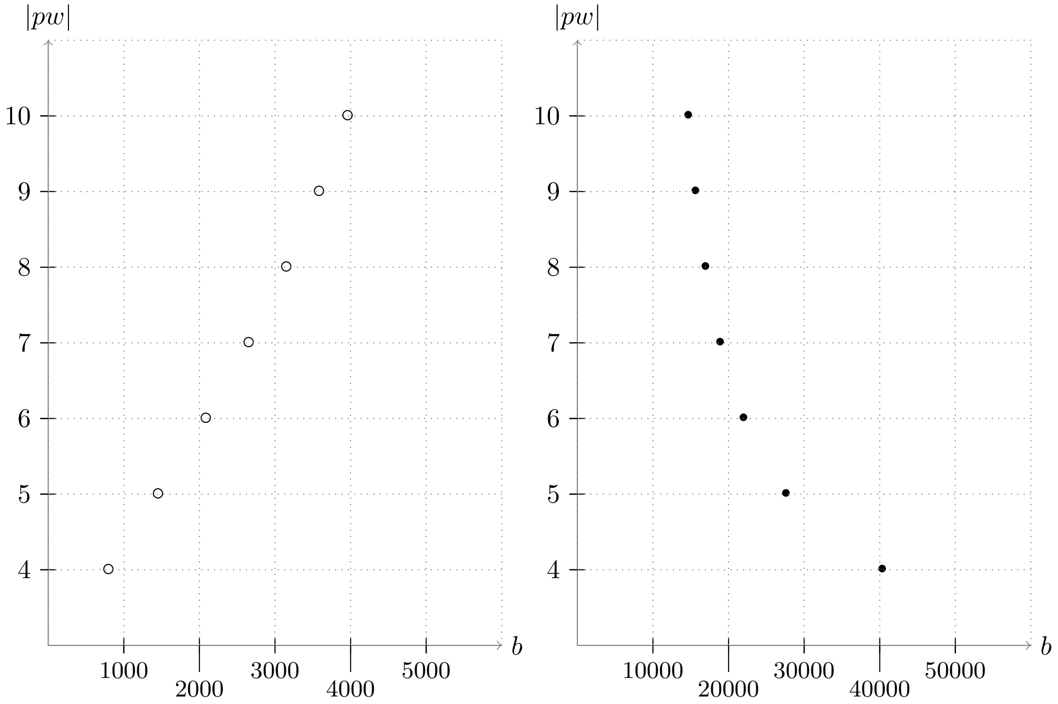


Fig. 1: Optimal Choice for b
• without regular expression R
○ with regular expression $R =$ `duls`

5 ZKPPC and Password Registration

We first define the ZKPPC concept enabling a client to prove compliance of its chosen passwords pw with respect to a server's password policy f without disclosing pw. We propose a general framework for building ZKPPC protocols for ASCII-based passwords and a concrete ZKPPC instantiation. We further explain how to build registration protocols that use ZKPPC as a building block.

5.1 Zero-Knowledge Password Policy Checks

A Password Policy Check (PPC) is an interactive protocol between a client $\boldsymbol{C}$ and a server $\boldsymbol{S}$ where server's password policy f and the public parameters of a password hashing scheme Π are used as a common input.

At the end of the PPC execution $\boldsymbol{S}$ accepts $H \leftarrow \mathtt{Hash_P}(pw, r)$ for any password $pw \in \mathcal{D}$ of client's choice if and only if $f(pw) = \mathtt{true}$. A PPC protocol is a proof of knowledge for pw and r such that $H \leftarrow \mathtt{Hash_P}(pw, r)$ and $f(pw) = \mathtt{true}$. It thus includes the requirements on completeness and soundness. In addition, a ZKPPC protocol is a PPC protocol with zero-knowledge property to ensure that no information about pw is leaked to $\boldsymbol{S}$. More formally,

Definition 1 (ZKPPC). *Let $\Pi = (\mathtt{PSetup}, \mathtt{PPHSalt}, \mathtt{PPreHash}, \mathtt{PHSalt}, \mathtt{PHash})$ be a password hashing scheme and f be a password policy. A ZKPPC protocol is a zero-knowledge proof of knowledge protocol between a prover $\boldsymbol{C}$ (client) and a verifier $\boldsymbol{S}$ (server), defined as*

$$\mathsf{ZKPoK}\{(pw, r): \ f(pw) = \mathtt{true} \wedge H = \mathtt{Hash_P}(pw, r)\}.$$

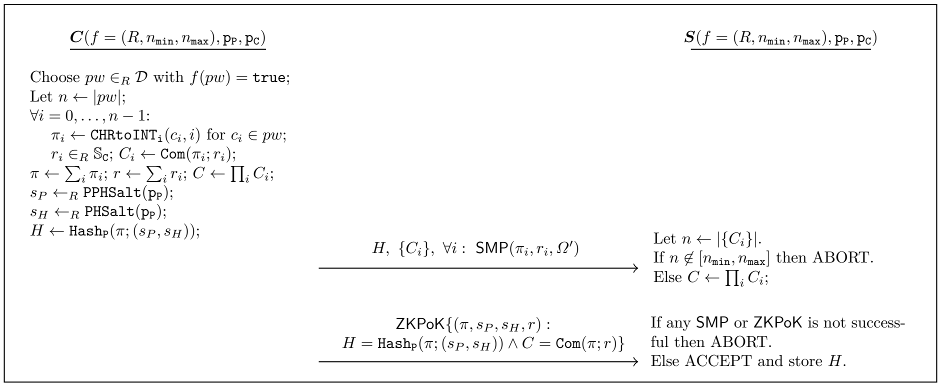
5.2 A General ZKPPC Framework for ASCII-based Passwords

We present a general ZKPPC construction for password strings pw composed of printable ASCII characters using a commitment scheme $\mathtt{C} = (\mathtt{CSetup}, \mathtt{Com}, \mathtt{Open})$, a password hashing scheme $\Pi = (\mathtt{PSetup}, \mathtt{PPHSalt}, \mathtt{PPreHash}, \mathtt{PHSalt}, \mathtt{PHash})$ and appropriate set membership proofs SMP. We assume that the common input of $\boldsymbol{C}$ and $\boldsymbol{S}$ includes $\mathtt{p_P} \leftarrow \mathtt{PSetup}(\lambda)$, $\mathtt{p_C} \leftarrow \mathtt{CSetup}(\lambda)$, and the server's password policy $f = (R, n_{\mathtt{min}}, n_{\mathtt{max}})$ that is communicated to $\boldsymbol{C}$ beforehand.

The ZKPPC protocol proceeds as follows (see also Figure 2 for an overview). Let R_j be the jth character of R. R_j uniquely identifies one of the four ASCII subsets of $\Sigma = d \cup u \cup l \cup s$ and one of the four integer sets Ω_x, $x \in \Sigma' = \{d, u, l, s\}$. Let $\Omega_\Sigma = \bigcup_{x \in \Sigma'} \Omega_x$ be a joint integer set of these four sets. The client picks an ASCII string $pw = (c_0, \ldots, c_{n-1})$ such that $f(pw) = \mathtt{true}$, computes integer values $\pi_i \leftarrow \mathtt{CHRtoINT_i}(c, i)$ for all $i = 0, \ldots, n-1$ and $\pi \leftarrow \mathtt{PWDtoINT}(pw) = \sum_{i=0}^{n-1} \pi_i$, and the password hash $H \leftarrow \mathtt{Hash_P}(\pi, (s_P, s_H))$ using salt $s_P \leftarrow \mathtt{PPHSalt}(\lambda)$ and $s_H \leftarrow \mathtt{PHSalt}(\lambda)$. For each position $i = 0, \ldots, n-1$ the client computes commitment $C_i \leftarrow \mathtt{Com}(\pi_i, r_i)$ and sends its password hash H with the set of commitments $\{C_i\}$ to $\boldsymbol{S}$ that by checking $|\{C_i\}| \in [n_{\mathtt{min}}, n_{\mathtt{max}}]$ will be able to check the password length requirement from f. Since $f(pw) = \mathtt{true}$, for each R_j in R the client can determine the first character $c_j \in pw$ that fulfils R_j and mark it as *significant*. Let $\{c_{i_1}, \ldots c_{i_{|R|}}\}$ denote the set of significant characters from pw that is sufficient to fulfill R. For each significant $c_{i_j} \in pw$, $j = 1, \ldots, |R|$ client $\boldsymbol{C}$ as prover and server $\boldsymbol{S}$ as verifier execute a set membership proof $\mathsf{SMP}(\pi_{i_j}, r_{i_j}, \Omega_x)$, i.e. proving that position-dependent integer value π_{i_j} committed to in C_{i_j} is in Ω_x for one of the four ASCII subsets in Σ identified by R_j. These SMPs ensure that characters fulfill R. For every other character $c_i \in pw$, $i \neq i_j$, $j = 1, \ldots, |R|$ client $\boldsymbol{C}$ as prover and server $\boldsymbol{S}$ as verifier execute $\mathsf{SMP}(\pi_i, r_i, \Omega_\Sigma)$ proving that position-dependent integer value π_i committed to in C_i is in the joint integer set Ω_Σ. This proves that each remaining c_i is a printable ASCII character without disclosing its type and thus ensures that $\boldsymbol{S}$ doesn't learn types of (remaining) password characters that are not necessary for R. Note that in the notation $\mathsf{SMP}(\pi_i, r_i, \Omega')$ used in Figure 2, set Ω' is either one of Ω_x, $x \in \Sigma'$ if π_i represents a significant character or Ω_Σ for all remaining characters.

If all SMPs are successful then $\boldsymbol{S}$ is convinced that commitments $\{C_i\}$ contain some integer values π_i representing characters c_i that fulfill R and that $n \in [n_{\mathtt{min}}, n_{\mathtt{max}}]$. This doesn't complete the proof yet since two issues remain: (1) committed π_i are not yet linked to the integer value π that represents pw, and (2) the client hasn't proved yet that this π was used to compute the hash value H. In order to address (1) and (2) our ZKPPC framework first uses the homomorphic property of the commitment scheme. Both $\boldsymbol{C}$ and $\boldsymbol{S}$ independently compute $C \leftarrow \prod_{i=0}^{n-1} C_i = \mathtt{Com}(\sum_{i=0}^{n-1} \pi_i, r) = \mathtt{Com}(\pi, r)$, where $r = \sum_{i=0}^{n-1} r_i$, whereas $\boldsymbol{C}$ additionally uses the knowledge of all r_i to compute r. As a last step of the ZKPPC protocol client $\boldsymbol{C}$ as prover and server $\boldsymbol{S}$ as verifier execute a ZKPoK proof that $\boldsymbol{C}$ knows π and random salts (s_P, s_H) that were used to compute H and that π is an integer contained in the (combined) commitment C for which the client knows the (combined) randomness r. If this final ZKPoK is successful then $\boldsymbol{S}$ accepts the hash value H.

In reference to Definition 1, our ZKPPC framework in Figure 2 tailors the general statement $f(pw) = \mathtt{true}$ to ASCII-based policies $f = (R, n_{\mathtt{min}}, n_{\mathtt{max}})$ and corresponding password hashing schemes Π so that the



$\boldsymbol{C}(f = (R, n_{\mathtt{min}}, n_{\mathtt{max}}), \mathtt{p_P}, \mathtt{p_C})$

Choose $pw \in_R \mathcal{D}$ with $f(pw) = \mathtt{true}$;
Let $n \leftarrow |pw|$;
$\forall i = 0, \ldots, n-1$:
  $\pi_i \leftarrow \mathtt{CHRtoINT_i}(c_i, i)$ for $c_i \in pw$;
  $r_i \in_R \mathbb{S}_{\mathtt{C}}$; $C_i \leftarrow \mathtt{Com}(\pi_i; r_i)$;
$\pi \leftarrow \sum_i \pi_i$; $r \leftarrow \sum_i r_i$; $C \leftarrow \prod_i C_i$;
$s_P \leftarrow_R \mathtt{PPHSalt}(\mathtt{p_P})$;
$s_H \leftarrow_R \mathtt{PHSalt}(\mathtt{p_P})$;
$H \leftarrow \mathtt{Hash_P}(\pi; (s_P, s_H))$;

$\boldsymbol{S}(f = (R, n_{\mathtt{min}}, n_{\mathtt{max}}), \mathtt{p_P}, \mathtt{p_C})$

$\xrightarrow{H,\ \{C_i\},\ \forall i:\ \mathsf{SMP}(\pi_i, r_i, \Omega')}$

Let $n \leftarrow |\{C_i\}|$.
If $n \notin [n_{\mathtt{min}}, n_{\mathtt{max}}]$ then ABORT.
Else $C \leftarrow \prod_i C_i$;

$\xrightarrow{\mathsf{ZKPoK}\{(\pi, s_P, s_H, r) :\ H = \mathtt{Hash_P}(\pi; (s_P, s_H)) \wedge C = \mathtt{Com}(\pi; r)\}}$

If any SMP or ZKPoK is not successful then ABORT.
Else ACCEPT and store H.

Fig. 2: General ZKPPC Framework for ASCII-based Passwords

resulting ZKPPC proof is of the following form:

$$\begin{aligned}&\mathsf{ZKPoK}\{(\pi, r, \{\pi_i\}, \{r_i\} \text{ for } i = 0, \ldots, n-1) :\\ &\qquad C_i = \mathtt{Com}(\pi_i, r_i) \wedge \prod_i C_i = \mathtt{Com}(\pi, \sum_i r_i) \wedge \pi_i \in \Omega' \wedge H = \mathtt{Hash_P}(\pi, r)\}.\end{aligned}$$

Theorem 1. *If* $\mathtt{C} = (\mathtt{CSetup}, \mathtt{Com}, \mathtt{Open})$ *is an (additively) homomorphic commitment scheme,* $\mathit{\Pi} = (\mathtt{PSetup}, \mathtt{PPHSalt}, \mathtt{PPreHash}, \mathtt{PHSalt}, \mathtt{PHash})$ *a secure randomised password hashing scheme,* SMP *a zero-knowledge set membership proof and* ZKPoK *a zero-knowledge proof of knowledge, then the protocol from Figure 2 is a ZKPPC protocol according to Definition 1.*

Proof. Protocol *completeness* follows by inspection. To prove *soundness* we assume that the server accepts H from a malicious client that was not computed as $\mathtt{Hash_P}(\pi, r)$ for integer π that represents a policy-conform password string pw. By construction of the protocol the client must have either (1) cheated in one of the $\mathsf{SMP}(\pi_i, r_i, \Omega')$ proofs or the final ZKPoK proof which contradicts the soundness properties of those proofs, or (2) was is able to compute H in two different ways, as $\mathtt{Hash_P}(\pi, r)$ using π that corresponds to a policy-conform $pw \in \mathcal{D}$ and as $\mathtt{Hash_P}(\pi^*, r^*)$ using π^* for some $pw^* \in \mathcal{D}$ that is not policy-conform, which contradicts to the second pre-image resistance of $\mathit{\Pi}$, or (3) was is able to compute at least one C_i in two different ways, as $\mathtt{Com}(\pi_i, r_i)$ using π_i that corresponds to a character c_i that is significant for the regular expression R and as $\mathtt{Com}(\pi_i^*, r_i^*)$ using π_i^* that doesn't fulfill any character R_j from R, which contradicts to the binding property of $\mathtt{C}$.

To prove the *zero-knowledge* property we need to build a simulator Sim to simulate the view of the server. Sim internally uses the simulators for SMP proofs and the ZKPoK proofs to simulate server's view, thereby relying on the password hiding property of $\mathit{\Pi}$ and the hiding property of $\mathtt{C}$ in the simulation of H and every C_i, respectively. □

Remark 2. Depending on the maximal password length $n_{\mathtt{max}}$ and complexity of $f = (R, n_{\mathtt{min}}, n_{\mathtt{max}})$ using range proofs instead of set membership proofs, may be more efficient. Although ZKPPC complexity is currently dominated by set membership proofs, passwords in practice are rather short and policies not too complex, so that set membership proofs will be sufficiently efficient in most cases. Further notice that leakage of password length n to the server is not considered as an attack against the ZKPPC protocol. For policies those regular expression R implicitly defines $n_{\mathtt{min}}$ the length n can be hidden using the homomorphic property the commitment scheme $\mathtt{C}$, i.e., by combining commitments C_i for π_i representing (remaining) password characters that are not needed to satisfy R.

5.3 A Concrete ZKPPC Protocol for ASCII-based Passwords

We show feasibility of our approach by giving a concrete ZKPPC protocol construction for ASCII-based passwords in a cyclic group G of prime order p. The protocol is built from the Pedersen commitment scheme $\mathtt{C} = (\mathtt{CSetup}, \mathtt{Com}, \mathtt{Open})$ from Section 2 and the randomised password hashing scheme $\mathit{\Pi} = (\mathtt{PSetup}, \mathtt{PPHSalt}, \mathtt{PPreHash}, \mathtt{PHSalt}, \mathtt{PHash})$ from Section 4.1 that share the same group G. In particular, public parameters used by $\boldsymbol{C}$ and $\boldsymbol{S}$ in the ZKPPC protocol are defined as (p, g, h, λ) where g and h are independent generators of G. For set membership proofs $\mathsf{SMP}(\pi_i, r_i, \mathit{\Omega}')$ we adopt a three-move honest-verifier proof $\mathsf{ZKPoK}\{(\pi_i, r_i): \ C_i = g^{\pi_i} h^{r_i} \wedge (\pi_i = \omega_0 \vee \cdots \vee \pi_i = \omega_{|\mathit{\Omega}'|})\}$ for $\omega_j \in \mathit{\Omega}'$, whose length is proportional to $|\mathit{\Omega}'|$. Assuming that for each $\omega_j \in \mathit{\Omega}'$ the corresponding value $g^{\omega_j} \in G$ is pre-computed this proof can be realised as $\mathsf{ZKPoK}\{(\pi_i, r_i): \ C_i = g^{\pi_i} h^{r_i} \wedge (C_i = g^{\omega_0} h^{r_i} \vee \cdots \vee C_i = g^{\omega_{|\mathit{\Omega}'|}} h^{r_i})\}$.[4] The final ZKPoK proof is instantiated as a three-move honest-verifier proof $\mathsf{ZKPoK}\{(\pi, s_P, s_H, r) \ : \ H_1 = g^{s_P} \wedge H_2 = H_1^{\pi} h^{s_H} \wedge C = g^{\pi} h^{r}\}$ that proceeds in the following classical way. $\boldsymbol{C}$ picks random $k_\pi, k_{s_P}, k_{s_H}, k_r \in \mathbb{Z}_p$, computes $t_1 = g^{k_{s_P}}$, $t_2 = H_1^{k_\pi} h^{k_{s_H}}$, and $t_3 = g^{k_\pi} h^{k_r}$, and sends (t_1, t_2, t_3) to $\boldsymbol{S}$ that replies with a random challenge $c \in \mathbb{Z}_p$. $\boldsymbol{C}$ computes $a_1 = k_{s_P} + c s_P \mod p$, $a_2 = k_\pi + c\pi \mod p$, $a_3 = k_{s_H} + c s_H \mod p$ and $a_4 = k_r + cr \mod p$, and sends (a_1, a_2, a_3, a_4) to $\boldsymbol{S}$ that accepts the proof if $g^{a_1} = t_1 H_1^c$, $H_1^{a_2} h^{a_3} = t_2 H_2^c$, and $g^{a_2} h^{a_4} = t_3 C^c$ holds.

Remark 3. The honest-verifier ZK property of the adopted three-move SMP and ZKPoK protocols is sufficient since ZKPPC will be executed as part of the registration protocol over a server-authenticated secure channel (cf. Section 5.4) where the server is assumed to be honest-but-curios. If ZKPPC protocol is executed outside of such secure channel then common techniques from [8] can be applied to obtain ZK property in presence of malicious verifiers. We also observe that all SMP and ZKPoK protocols can be made non-interactive (in the random oracle model) using the techniques from [11].

5.4 Blind Registration of Passwords based on ZKPPC

Blind registration of passwords based on our generic ZKPPC construction from Section 5.2 proceeds in *three* main stages and requires server-authenticated secure channel (e.g. TLS) between $\boldsymbol{C}$ and $\boldsymbol{S}$: (1) $\boldsymbol{S}$ sends its password policy f to $\boldsymbol{C}$; (2) $\boldsymbol{C}$ picks its user login credentials, containing id (e.g. its email address) which $\boldsymbol{C}$ wants to use for later logins at $\boldsymbol{S}$, and initiates the execution of the ZKPPC protocol. If the ZKPPC protocol is successful then $\boldsymbol{C}$ has a policy-conform password pw and $\boldsymbol{S}$ receives id and the password hash $H = \mathtt{Hash}_{\mathtt{P}}(\pi, r)$; (3) $\boldsymbol{C}$ sends used random salt r to $\boldsymbol{S}$ and $\boldsymbol{S}$ stores a tuple (id, H, r) in its password database.

The use of server-authenticated secure channel guarantees that no active adversary A can impersonate honest $\boldsymbol{S}$ and obtain (id, H, r) nor can A mount an attack based on modification of server's policy f, e.g. by replacing it with a weaker one. Especially, r needs protection since knowledge of (H, r) enables an offline attack that recovers pw. Assuming an efficiently samplable dictionary $\mathcal{D}$ with min-entropy β a brute force attack would require at most 2^β executions of $\mathtt{Hash}_{\mathtt{P}}(\pi^*, r)$, where $\pi^* \leftarrow \mathtt{PWDtoINT}(pw^*)$, $pw^* \in \mathcal{D}$.

The execution of the ZKPPC protocol in the second stage doesn't require a secure channel due to the assumed ZK property. However, if secure channel is in place then we can work with the *honest-verifier* ZK property, which may lead to more efficient ZKPPC constructions. Note that $\boldsymbol{S}$ is not assumed to be fully malicious but rather honest-but-curios since it cannot be trusted to process plain passwords in a secure way. By modeling $\boldsymbol{S}$ as a malicious party in the ZKPPC protocol we can offer strong guarantees that no information about pw is leaked to $\boldsymbol{S}$ in the second stage and so the only way for $\boldsymbol{S}$ to recover pw at the end is to mount an offline dictionary attack using r from the third stage.

The resulting password registration protocol guarantees that no server $\boldsymbol{S}$ can do better in recovering client's pw than any attacker A who compromises $\boldsymbol{S}$ during or after the registration phase. This is an ideal security requirement for the registration of passwords that will be used in authentication protocols with password verifiers on the server side. Note that security of such verifier-based authentication protocols

[4] More efficient SMPs, e.g. [6], can possibly be used with a different commitment and password hashing scheme. In this case care must be taken when it comes to the instantiation of VPAKE that must be able to handle password hashes generated in ZKPPC (cf. Section 6).

implies that any A who breaks into $\boldsymbol{S}$ cannot recover pw better than by mounting an offline dictionary attack. Our approach thus extends this requirement to password registration protocols.

For our concrete ZKPPC construction from Section 5.3 we can modify the third stage of the registration protocol such that instead of $r = (s_P, s_H)$ server $\boldsymbol{S}$ receives only s_H and stores (id, H, s_H), where $H = (H_1, H_2)$, $H_2 = H_1^{\pi} h^{s_H}$. This trick helps to significantly increase the complexity of an offline dictionary attack. Note that pre-image resistance of $\mathit{\Pi}$ guarantees that an offline password test based on equality $H_1^{\pi} = H_2 h^{-s_H}$ would require 2^{β} exponentiations $H_1^{\pi^*}$ until $\pi^* = \pi$ is found. Note that if s_P is disclosed then the above equality can be re-written to $g^{\pi} = (H_2 h^{-s_H})^{1/s_P}$ and a pre-computed table $T = (\pi^*, g^{\pi^*})$ would immediately reveal $\pi^* = \pi$. The computation of T requires 2^{β} exponentiations g^{π^*} but T would need to be computed only once. This also explains why we use $\mathit{\Pi}$ with randomised $\texttt{PPreHash}$.

6 VPAKE Protocols for ZKPPC-registered Passwords

We now focus on suitable VPAKE protocols where the server $\boldsymbol{S}$ using (id, H, r) stored from the ZKPPC-based registration protocol can authenticate the client $\boldsymbol{C}$ that uses only its pw. Such protocols can be constructed with a general VPAKE framework introduced by Benhamouda and Pointcheval [5]. Their framework constructs one-round VPAKE protocols with $\boldsymbol{C}$ and $\boldsymbol{S}$ sending one message each, independently, using a generic password hashing scheme $\mathit{\Pi} = (\texttt{PSetup}, \texttt{PPHSalt}, \texttt{PPreHash}, \texttt{PHSalt}, \texttt{PHash})$ with deterministic $\texttt{PPreHash}$, labeled public key encryption scheme $\texttt{E} = (\texttt{KGen}, \texttt{Enc}, \texttt{Dec})$, and secure SPHFs $(\texttt{HKGen}, \texttt{PKGen}, \texttt{Hash}, \texttt{ProjHash})$ for two languages $L_H = \{(\ell, C) | \exists r : C = \texttt{Enc}^{\ell}(pk, H; r)\}$ and $L_{s,H} = \{(\ell, C) | \exists P, \exists r : C = \texttt{Enc}^{\ell}(pk, H; r) \ \wedge \ H = \texttt{PHash}(\texttt{p}_{\texttt{P}}, P, s)\}$. Their approach can directly be used for our generic scheme $\mathit{\Pi}$ with randomised $\texttt{PPreHash}$ if we assume that $L_{s,H}$ is defined using $s = (s_P, s_H)$. This readily gives us a generic VPAKE protocol that is suitable for our general ZKPPC construction for ASCII-based passwords in Figure 2 and those security follows from the analysis of the framework in [5].

For the concrete VPAKE construction based on our scheme $\mathit{\Pi}$ from Section 4.1 we can use labeled CS encryption scheme for $\texttt{E}$ from Section 2. The common input of $\boldsymbol{C}$ and $\boldsymbol{S}$ contains the CS public key $pk = (p, g_1, g_2, h, c, d, H_k)$, where generators $g_1 = g$ and h must be the same as in the ZKPPC protocol from Section 5.3. Since $H = (H_1, H_2)$ we need to slightly update the language $L_H = \{(\ell, C) | \exists r : C = \texttt{Enc}^{\ell}(pk, H_2; r)\}$ by using H_2 as an encrypted message. We can still use the SPHF for CS ciphertexts from Section 4.1 to handle this L_H. Since the pre-hash salt s_P is not transmitted in the registration phase, i.e. $\boldsymbol{S}$ stores (id, H, s_H) where $H = (H_1, H_2)$ with $H_1 = g_1^{s_P}$ and $H_2 = H_1^{\pi} h^{s_H}$, we replace $L_{s,H}$ with the following language $L_{s_H,H} = \{(\ell, C) | \exists \pi, \exists r : C = \texttt{Enc}^{\ell}(pk, g_1^{\pi}; r) \ \wedge \ H_2 = H_1^{\pi} h^{s_H}\}$ and construct a suitable SPHF for $L_{s_H,H}$ as follows:

- $\texttt{HKGen}(L_{s_H,H})$ generates $\texttt{hk} = (\eta_1, \eta_2, \theta, \mu, \nu) \in_R \mathbb{Z}_p^{1 \times 5}$.
- $\texttt{PKGen}(\texttt{hk}, L_{s_H,H})$ derives $\texttt{hp} = (\texttt{hp}_1, \texttt{hp}_2, \texttt{hp}_3) = (g_1^{\eta_1} g_2^{\theta} h^{\mu} c^{\nu}, g_1^{\eta_2} d^{\nu}, g_1^{\mu} H_1^{-\mu})$.
- $\texttt{Hash}(\texttt{hk}, L_{s_H,H}, C)$ outputs hash value $h = u_1^{\eta_1 + \xi \eta_2} u_2^{\theta} [e/(H_2 h^{-s_H})]^{\mu} v^{\nu}$.
- $\texttt{ProjHash}(\texttt{hp}, L_{s_H,H}, C, \pi, r)$ outputs hash value

$$h = (\texttt{hp}_1 \texttt{hp}_2{}^{\xi})^r \texttt{hp}_3{}^{\pi} = g_1^{\eta_1 r} g_2^{\theta r} h^{\mu r} c^{\nu r} g_1^{\eta_2 \xi r} d^{\nu \xi r} (g_1^{\mu} H_1^{-\mu})^{\pi}.$$

Note that projection key $\texttt{hp}$ depends on $H_1 \in G$, which can be seen as a parameter in the definition of $L_{s_H,H}$, but $\texttt{hp}$ does not depend on C. The resulting VPAKE protocol can thus still proceed in one round. The smoothness of our SPHF construction for $L_{s_H,H}$ can be proven as follows. Let $\pi \leftarrow \texttt{PWDtoINT}(pw)$, $H_2 = H_1^{\pi} h^{s_H}$, with $H_1 = g_1^{s_P}$ for some unknown s_P, and $(\ell, C = (u_1, u_2, e, v)) \notin L_{s_H,H}$, i.e. $C \leftarrow \texttt{Enc}^{\ell}(pk, g_1^{\pi^*}; r)$ for some $\pi^* \neq \pi$. Assuming the second pre-image resistance of $\mathit{\Pi}$ it follows that $(u_1, u_1^{\xi}, u_2, e/(H_2 h^{-s_H}), v) \neq (g_1^r, g_1^{r\xi}, g_2^r, g_1^{\pi - s_P \pi} h^r, \ (cd^{\xi})^r)$ with overwhelming probability for all $(r, r\xi) \in \mathbb{Z}_p^2$. Since $(\texttt{hp}_1, \texttt{hp}_2, \texttt{hp}_3)$ are linearly independent the resulting hash value $h = u_1^{\eta_1} u_1^{\xi \eta_2} u_2^{\theta} [e/(H_2 h^{-s_H})]^{\mu} v^{\nu}$ is uniformly distributed in G.

Our concrete VPAKE construction is illustrated in Figure 3. We assume that $\boldsymbol{C}$ uses $\pi \leftarrow \texttt{PWDtoINT}(pw)$ as its input and has already sent its login name id to $\boldsymbol{S}$ who picked the corresponding tuple (id, H, s_H) from its password database. Note that $\boldsymbol{C}$ can also act as initiator and send its id as part of its message, in which

$\boldsymbol{C}(pk, (id, \pi))$		$\boldsymbol{S}(pk, (id, H = (H_1, H_2), s_H))$
$\texttt{hk} \leftarrow \texttt{HKGen}(L_H)$; $\texttt{hp} \leftarrow \texttt{PKGen}(\texttt{hk}, L_H)$; $r \in_R \mathbb{Z}_p$; $\ell = (id, \boldsymbol{S}, \texttt{hp})$; $C \leftarrow \texttt{Enc}^{\ell}(pk, g_1^{\pi}; r)$;		$\texttt{hk}' \leftarrow \texttt{HKGen}(L_{s_H,H})$; $\texttt{hp}' \leftarrow \texttt{PKGen}(\texttt{hk}', L_{s_H,H})$; $r' \in_R \mathbb{Z}_p$; $\ell' = (\boldsymbol{S}, id, \texttt{hp}')$; $C' \leftarrow \texttt{Enc}^{\ell'}(pk, H_2; r')$;
	$\xrightarrow{\texttt{hp},\ C}$	
	$\xleftarrow{\texttt{hp}', C', H_1, s_H}$	
$\ell' = (\boldsymbol{S}, id, \texttt{hp}')$; $H_2 \leftarrow H_1^{\pi} h^{s_H}$; $K_1 \leftarrow \texttt{Hash}(\texttt{hk}, L_H, C')$; $K_2 \leftarrow \texttt{ProjHash}(\texttt{hp}', L_{s_H,H}, C, \pi, r)$; $K \leftarrow K_1 \cdot K_2$		$\ell = (id, \boldsymbol{S}, \texttt{hp})$; $K_1 \leftarrow \texttt{ProjHash}(\texttt{hp}, L_H, C', r')$; $K_2 \leftarrow \texttt{Hash}(\texttt{hk}', L_{s_H,H}, C)$; $K \leftarrow K_1 \cdot K_2$

Fig. 3: A VPAKE Protocol for Blindly Registered ASCII-based Passwords

case $\boldsymbol{S}$ must act as a responder. Which SPHF algorithms HKGen, PKGen, Hash, ProjHash are used by $\boldsymbol{C}$ and $\boldsymbol{S}$ is visible from the input language, either L_H or $L_{s_H,H}$. By inspection one can see that if both $\boldsymbol{C}$ and $\boldsymbol{S}$ follow the protocol and H used on the server side is a password hash of π used on the client side then both parties compute the same (secret) group element $K = K_1 \cdot K_2$. Note that $\boldsymbol{C}$ derives K_1 using its own hashing key hk and received server's CS ciphertext C' that encrypts H_2, whereas $\boldsymbol{S}$ derives K_1 using client's projection key hp, its own C' and r'. Similarly, $\boldsymbol{S}$ derives K_2 using its own hashing key hk′ and received client's CS ciphertext C that encrypts g_1^{π}, whereas $\boldsymbol{C}$ derives K_2 using server's projection key hp′, its own C and r. Security of this VPAKE protocol follows from the security of the generic scheme.

7 Conclusion

The proposed ZKPPC framework with additional password registration and VPAKE protocols presented in this work can be used to securely register passwords chosen by clients at remote servers while simultaneously achieving the following properties: (1) registered passwords are never disclosed to the server and the only way for the server or any attacker who compromises the server to recover passwords is by mounting an expensive offline dictionary attack; (2) each registered password provably satisfies server's password policy, which is ensured through the use of homomorphic commitments and appropriate set membership proofs; (3) servers can authenticate clients those passwords were registered using the ZKPPC framework by means of efficient VPAKE protocols. We believe that the concept underlying the ZKPPC framework and its current realization for ASCII-based passwords and policies can solve problems related to the inappropriate handling of user passwords that frequently occurs in the real world.

Future work may include extension of the ZKPPC concept towards Two-Server PAKE (2PAKE) protocols, e.g. [15], where the client password is secretly shared amongst two servers from which at most one is assumed to be compromisable. Under this security assumption 2PAKE servers fully eliminate threats from offline dictionary attacks. However, blind registration of policy-conform passwords for 2PAKE protocols under this security assumption is a challenge.

Acknowledgements This research was supported by the German Science Foundation (DFG) through the project PRIMAKE (MA 4957).

References

1. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT'00*, volume 1807, pages 139–155, Berlin, Heidelberg, 2000. Springer. 2
2. S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. IEEE S&P'92, pages 72–84. IEEE CS, 1992. 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 *ACM CCS'93*, pages 244–250. ACM, 1993. 2

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/. 4
5. F. Benhamouda and D. Pointcheval. Verifier-Based Password-Authenticated Key Exchange: New Models and Constructions. *IACR Cryptology ePrint Archive*, 2013:833, 2013. 2, 6, 7, 8, 13, 15
6. 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. 12
7. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally Composable Password-Based Key Exchange. In *EUROCRYPT'05*, pages 404–421, Berlin, Heidelberg, 2005. Springer. 2
8. R. Cramer, I. Damgård, and P. D. MacKenzie. Efficient Zero-Knowledge Proofs of Knowledge Without Intractability Assumptions. In *PKC'00*, volume 1751 of *LNCS*, pages 354–373. Springer, 2000. 12
9. R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In *CRYPTO'98*, volume 1462 of *LNCS*, pages 13–25. Springer, 1998. 4
10. 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
11. 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. 12
12. 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. 2
13. 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. 1
14. B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898 (Informational), Sept. 2000. 2
15. F. Kiefer and M. Manulis. Distributed Smooth Projective Hashing and Its Application to Two-Server Password Authenticated Key Exchange. In *ACNS'14*, volume 8479 of *LNCS*, pages 199–216. Springer, 2014. 14
16. F. Kiefer and M. Manulis. Zero-knowledge password policy checks and verifier-based PAKE. In *ESORICS'14*, volume 8713 of *Lecture Notes in Computer Science*, pages 295–312. Springer, 2014. 5, 9
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 *LNCS*, pages 129–140. Springer-Verlag, 1991. 4
19. D. Pointcheval. Password-Based Authenticated Key Exchange. 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 *USENIX Security'12*, pages 5–5. USENIX Association, 2012. 1

A 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 $H \leftarrow \texttt{PHash}(\mathsf{p}_\mathsf{P}, P, s_H)$ and $P \leftarrow \texttt{PPreHash}(\mathsf{p}_\mathsf{P}, pw_S, s_P)$ for random salts $s_P \leftarrow \texttt{PPHSalt}(\mathsf{p}_\mathsf{P})$ and $s_H \leftarrow \texttt{PHSalt}(\mathsf{p}_\mathsf{P})$. 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 $\mathtt{PHash}(\mathtt{p_P}, \mathtt{PPreHash}(\mathtt{p_P}, pw_P, s_P), s_H) = H_{P'}$. 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 $\mathit{\Pi}$ is secure if for all PPT adversaries $\mathcal{A}$ there exists a negligible function $\varepsilon(\cdot)$ such that

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