Client-Server Password Recovery

(Extended Abstract)

Łukasz Chmielewski[1] Jaap-Henk Hoepman[1,2] Peter van Rossum[1]

[1] Digital Security Group
Radboud University Nijmegen, the Netherlands
{lukaszc,jhh,petervr}@cs.ru.nl
[2] TNO Information and Communication Technology, The Netherlands
jaap-henk.hoepman@tno.nl

Abstract. Human memory is not perfect – people constantly memorize new facts and forget old ones. One example is forgetting a password, a common problem raised at IT help desks. We present several protocols that allow a user to automatically recover a password from a server using partial knowledge of the password. These protocols can be easily adapted to the *personal entropy* setting [7], where a user can recover a password only if he can answer a large enough subset of personal questions.
We introduce *client-server* password recovery methods, in which the recovery data are stored at the server, and the recovery procedures are integrated into the login procedures. These methods apply to two of the most common types of password based authentication systems. The security of these solutions is significantly better than the security of presently proposed password recovery schemes. For our protocols we propose a variation of threshold encryption [5, 8, 17] that might be of independent interest.

Key words: password recovery, threshold encryption scheme, private computing, personal entropy

1 Introduction

People constantly memorize new facts, but also forget old ones. One quite common example is forgetting a password. It is one of the most common problem raised at IT help-desks. Therefore, many systems for password recovery (PR) have been built. The common aim of all these systems is to provide reliable solutions for legitimate users to recover lost passwords or to receive a new password (i.e., resetting the old password), without significantly increasing the vulnerability against attackers.

The simplest way to authenticate the user is to use an out-of-band channel, like a phone call, or show up physically at a system administrator. This is costly however, and cumbersome. More user-friendly, but less secure, is the common method used by many websites that store the password of the user in the clear and resend it to the user's email address on request. Sometimes websites require a user to answer some personal question, like "what is your mother's maiden name?". However, this method is insecure because a password sent in cleartext can be easily intercepted and it is relatively easy to answer such a single question.

Another widely used method to cope with forgetting passwords is a password reset system. In this system when a user forgets the password then the server sets

a new password and emails the new password to the client (again maybe after answering a personal question). Now the legitimate user can regain system access easily. However, the security of this system depends heavily on the security of the email server, and therefore, this system is uninteresting from our point of view.

There is quite a lot of research on more sophisticated PR methods that do not fully trust the server. One approach is to use secret sharing [2,18]. This solution divides a password into n shares (that are stored on trusted servers) in such a way that for the reconstruction, it is necessary to collect at least a threshold t of these shares. However, the user still needs to authenticate somehow to the servers, and therefore this system does not fully solve our problem.

In [7] a PR system, based on *personal entropy*, is proposed. In this system, a user is asked some questions about his personal history during password registration. The system generates a random secret key, and encrypts the real password with it. Subsequently, the answers given by the user are used to "encrypt" the random secret key. The user then stores the questions, the "encryption" of the secret value, and the encryption of the password on his computer. A secret sharing scheme is used to enable password recovery, even if some questions are answered incorrectly. The drawback of this scheme is the lack of a rigorous security analysis. In fact, [3] demonstrates a serious weakness of this scheme: with the parameters recommended for a security level of 2^{112}, the system is in fact vulnerable to an attack that requires only 2^{64} operations.

The ideas from [7] were improved in [9]. This improved password recovery uses error-correcting codes instead of a secret sharing scheme. A rigorous security analysis is performed in the chosen model. The solution of [9] uses techniques that are very close to secure sketches.

Secure sketches and fuzzy extractors (described e.g., in [6]), and their robust versions [12,15], are cryptographic tools useful for turning noisy information into cryptographic keys and securely authenticating biometric data. They may also be used to solve the password recovery problem. However, contrary to intuition, it seems hard to use these cryptographic primitives to solve password recovery in our most secure model, as show in Section 5.

We believe that [7, 9] are a significant step towards a practical PR solution. However, such so-called *local* PR systems are vulnerable to attackers that steal the recovery data from the user's machine (which is quite often inadequately secured) and then mount an *offline* brute force attack to recover the password. To avoid this scenario, we introduce *client-server* password recovery, in which the recovery data should be stored at the server, and PR should be integrated into the login procedure. In such a setting (under the more reasonable assumption that the recovery data cannot be stolen from the secure server) an attacker can only perform an *online* brute force attack. Security then can be increased by limiting the number of tries per account, or increasing the response time.

Our contributions are the following. Firstly, we introduce the password recovery problem and the client-server PR security model, together with a short analysis of password authentication systems, in Section 2. All our client-server PR systems apply to a simple (low entropy) password login system. In all these

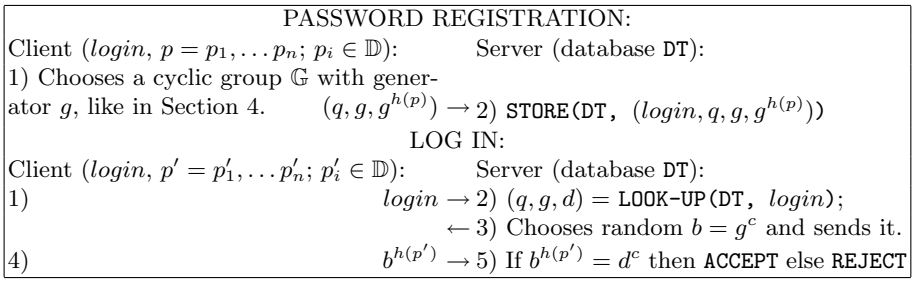

Fig. 1. challenge-response password authentication system

PR systems, the client is stateless, and all recovery data is stored at the server. Our solutions reduce the entropy somewhat, but are still more secure than other approaches. Moreover, our ideas can be straightforwardly applied to the personal entropy system, as shown in Subsection 2.2, making the recovery phase more secure. We elaborate on using secure sketches and fuzzy extractors for PR in Section 5. Subsequently, we present a new algorithm (Section 3) for local PR that is based on intraceability Assumption 2 from [14]. In Section 4, we introduce a new variant of threshold encryption [5,8,17], called *equivocal threshold encryption*, that does not provide validity proofs for the decryption shares. Combining these two, we present protocols for client-server PR integrated into two classes of systems for password based login: the most common, hash based one in which the server keeps hashes of passwords but receives cleartext passwords during the login phase (Section 6), and the most secure solution, based on challenge response, in which the server never sees passwords in clear at all (Section 7). Moreover, in Appendix A we briefly present a simple substring-knowledge PR working in the challenge response setting. Furthermore, all our password recovery systems can be easily modified to work as password reset systems. Due to space constraints we omit these easy transformations.

Due to space constraints in this version of the paper, proofs of security and correctness of the presented protocols are short and informal.

2 Password Recovery Security Model

In this section we discuss the kinds of password authentication (PA) systems for which we consider password recovery, define exactly what we mean by password recovery, and talk about the kinds of adversaries our protocols need to withstand.

2.1 Password Authentication (PA) Systems

Two kinds of participants are involved in PA systems: users (also called clients) and servers. Clients have a username (also called login) and a password $p{=}p_1, \ldots p_n$, where $p_i \in \mathbb{D}$ and $\mathbb{D}$ is the domain of characters of passwords ($\mathbb{D}$ is usually small, e.g., $|\mathbb{D}| \approx 100$). For simplicity, we assume that clients always remember their logins, and that the length of the password n is fixed for all users.

Initially, a client registers himself (in the *registration phase*) with the server by submitting a username and an authenticator (derived from the password), which the server stores in its database. Subsequently, the client can authenticate (in the *log in phase*) to the server using his username and a proof of knowledge of the password. The server, using the authenticator from the database, and the proof of knowledge, can efficiently verify that the user knows the corresponding password. We distinguish three different PA schemes with respect to the security requirements. These systems differ in the way that authenticators and proofs of knowledge are defined: an authenticator can be equal to a password, a proof can be equal to a password (this is the case in hash based systems, where the server stores hashes of passwords), or neither of the above (which is the case for challenge-response type systems, an example of which is presented in Figure 1).

The password recovery for the first system is trivial (because the server stores passwords in clear), and we omit it in this paper. The PR solutions for the other two PA systems are presented in Sections 6 and 7, respectively.

2.2 Client-Server Password Recovery (PR)

A system for *client-server* PR has the same participants and log in routine as a PA system. Moreover, it provides an additional routine called *password recovery* (PR), in which the client tries to recover the lost password. The password registration is also modified: besides submitting the login, and the authenticator, it also submits the recovery data. The client's input in the PR phase is login and a perturbed (incorrect) password $p'{=}p'_1, \ldots p'_n$, while the server's input is the database with the logins and the registration data. Local password recovery is similar to client-server password recovery, except that the recovery data is stored locally at the client, and the recovery protocol is run locally at the client.

The requirement is that the client recovers the password, if and only if, p' is similar to the password p corresponding to his login. To be precise, we define similarity between strings x and y as $x \approx_t y$ (x matches y), if and only if, $t \leq |\{i \in \{1, \ldots n\} : x_i = y_i\}|$. We assume that the parameters n and t are public.

Note, that having partial knowledge of the password is a very similar recovery condition to the personal entropy one [7, 9]. In the personal entropy system the client needs to answer some threshold of questions (i.e., t out of n questions) to recover the password. The answers to the questions can be considered as an additional password, where every single answer can be treated as a letter. It is easy to transform our systems to work with an auxiliary password, and therefore, with personal questions. We skip these straightforward transformations in this paper.

We develop our protocols based on the following assumptions. We assume existence of the secure channels between the server and clients (which can be achieved using TLS connections). We work in the Random Oracle Model (ROM) [1], which means that we assume that hash functions work like random functions. Moreover, we often use message authentication codes (MACs), that we implement using keyed hash functions of the form $\mathfrak{h} : \{0,1\}^k \times \mathbb{D} \rightarrow \mathbb{F}$, where $\mathbb{F}$ is a field. The first parameter of $\mathfrak{h}$ is a random string of length k (the security parameter). For simplicity, we often omit this parameter in our descriptions.

We look for efficient protocols, i.e., $O(nk)$, at the server side (because many clients might want to perform password recovery simultaneously), but we do allow a certain time penalty at the client side.

2.3 Adversaries and Security Requirements

All our client-server protocols defend against an adversary impersonating a client. Such an adversary is computationally bounded by k (but not by $n \log |\mathbb{D}|$) and is malicious [11], which means he can disobey the protocols routine. This adversary tries to break a server's privacy that can be informally defined as follows. The impersonator, after any number of unsuccessful PR runs, can recover more information about the password, than following from the fact that the PR invocations failed, only with a negligible probability in k. Notice however, that this adversary can always perform an online brute force attack on the PR routine (even using the password's distribution). But this is easily mitigated by adding timeouts or allowing only a fixed number of tries before blocking an account.

We also consider an adversary accessing the server's database in all our client-server protocols. We model this adversary differently than the one impersonating client, because this adversary can perform offline brute force attack using the PR routine. Therefore, we define the adversary to not know the password distribution and to be computationally bounded with respect to k and the parameters n, t, $|\mathbb{D}|$ (in a way that the problem from Assumption 3.1 is hard). The adversary tries to break a client's privacy that can be informally, defined as follows. For every two passwords p' and p'', the corresponding two PR data instances are indistinguishable. An adversary accessing local PR (see Section 3) is defined in the same way.

Only the challenge-response protocol (Section 7) is resistant against a fully corrupted server. The adversary corrupting the server is computationally bounded by k and tries to gain information about client's password guesses from the data received in PR runs. We assume that this adversary is malicious in the sense, that he performs any actions to break the *guesses* privacy. However, there is no point for him to alter the client's output: the client can easily verify correctness of the recovery by logging in. This approach is very similar to *private computation* from [14]. The guesses privacy can be defined as follows: from a PR run the adversary gains negligible knowledge about the client's guess.

3 Local Password Recovery

As explained in the introduction, a client of *local* password recovery, similarly to [7, 9], keeps the recovery data on his machine (there is no server). The client generates the recovery data and later on, tries to recover the lost password from the password guess and the recovery data. In Figure 2 we present a solution for local PR. Its security is based on the following intraceability assumption derived from [14], which is related to the polynomial list reconstruction problem.

The intraceability assumption. Let $C_{n,m}^{t,\alpha}$ denote the probability of distribution of sets generated in the following way:

Password Registration: The input is $p = p_1, \ldots p_n$, where $p_i \in \mathbb{D}$, and $|\mathbb{D}| = m$. The client:

1. Generates $v \in_R \{0,1\}^k$, and n values $\{\mathfrak{h}_1(p_1), \ldots \mathfrak{h}_n(p_n)\}$. Every $\mathfrak{h}_i$ is a MAC with implicit first parameter v as described in Section 2.2.
2. Generates n random values $s_1, \ldots s_n \in \mathbb{F}$ in such a way that points $\{(\mathfrak{h}_1(p_1), s_1), \ldots (\mathfrak{h}_n(p_n), s_n)\}$ define a polynomial P of degree $t-1$, and $P(0)=p$.
3. Returns: $PR=(v, \{s_1 - \mathfrak{g}_1(p_1), \ldots s_n - \mathfrak{g}_n(p_n))\}$; each $\mathfrak{g}_\mathfrak{i}$ is a similar MAC to h_i.

Password Recovery: The input is: $p' = p'_1, \ldots p'_n$, $PR = (v, \{s'_1, \ldots s'_n\})$.

1. The client computes set $S = \{(\mathfrak{h}_1(p'_1), s'_1 + \mathfrak{g}_1(p'_1)), \ldots (\mathfrak{h}_n(p'_n), s'_n + \mathfrak{g}_n(p'_n))\}$.
2. The client tries to reconstruct P from any subset of t elements of S (that is $\binom{n}{t}$ checks). He checks whether for any potentially recovered polynomial P' the following holds (let $p''=P'(0)$): $p'' \approx_t p'$ and $\{(\mathfrak{h}_1(p''_1), s'_1+\mathfrak{g}_1(p''_1)), \ldots (\mathfrak{h}_n(p''_n), s'_n + \mathfrak{g}_n(p''_n))\}$ defines a polynomial of degree n. If it holds then he outputs p''. If it does not hold for any P' then the client outputs $\emptyset$.

Fig. 2. Local Password Recovery

1. Pick a random polynomial P over $\mathbb{F}$ (denote $|\mathbb{F}| = f$), of degree at most t, such that $P(0) = \alpha$.
2. Generate nm random values $x_1, \ldots x_{nm} \in \mathbb{F}$ subject to the constraint that all x_i are distinct and different from 0.
3. Choose a random subset S of n different indexes in $\{1, \ldots nm\}$, and set $y_i = P(x_i)$ for all $i \in S$. For every $i \notin S$ set y_i to be a random value in $\mathbb{F}$.
4. Partition the nm (x_i, y_i) pairs in n random subsets subject to the following constraints. Firstly, the subsets are disjoint. Secondly, each subset contains exactly one pair whose index is in S (hence $y_i = P(x_i)$) and exactly $m-1$ pairs whose indexes are not in S. We denote these subsets as $S_i = \{(x_{(i,j)}, y_{(i,j)})\}$. Output the resulting subsets.

The intractability assumption states that for any α, α' the two probability ensembles $C^{t,\alpha}_{n,m}$, $C^{t,\alpha'}_{n,m}$ are computationally indistinguishable depending on the parameters f, t, m, and n.

Assumption 3.1 (Assumption 2 from [14]) *Let k be a security parameter, and let $n(k)$, $m(k)$, $t(k)$, $f(k)$ be at least linear polynomially bounded functions that define the parameters n, m, t and f. Let $\mathcal{C}^{t,\alpha}_{n,m}$ and $\mathcal{C}^{t,\alpha'}_{n,m}$ be random variables that are chosen according to the distributions $C^{t,\alpha}_{n,m}$ and $C^{t,\alpha'}_{n,m}$, respectively. Then it holds that for every $\alpha, \alpha' \in \mathbb{F}$, the probability ensembles $\mathcal{C}^{t,\alpha}_{n,m}$ and $\mathcal{C}^{t,\alpha'}_{n,m}$ are computationally indistinguishable.*

In our applications the assumption's parameters are set as follows: n and t like in PR, $m = |\mathbb{D}|$ and $\mathbb{F} = \mathbb{Z}_q$, where q is large prime. One may argue that n, t and $|\mathbb{D}|$ are relatively small parameters (e.g., n is the length of passwords) and that they might not deliver good security to the system. However, notice that in the personal entropy setting (i.e., the question-answer setting) the parameters can be significantly enlarged. Moreover, we are not aware of any algorithm solving

the assumption problem (i.e., finding α) in our setting faster than by guessing t proper points.

We are conscious that for similar problems there exist fast solutions. For example, if in the above problem all $x_{(i,j)} = i$ then the problem can be solved fast (see [3, 4]). However, these fast algorithms do not solve the problem from Assumption 3.1, as stated in [14].

The local PR solution. Now we describe the protocol. In the first step the client prepares PR data: v and $\{s_1 - \mathfrak{g}_1(p_1), \ldots s_n - \mathfrak{g}_n(p_n)\}$, such that $\{(\mathfrak{h}_1(p_1), s_1), \ldots, (\mathfrak{h}_n(p_n), s_n)\}$ define a polynomial P of degree $t-1$, for which $P(0) = p$. Here, $\mathfrak{h}_i, \mathfrak{g}_i$ are hash functions (see Figure 2). Afterwards, the client forgets the password, and tries to recover it from $S = \{(\mathfrak{h}_1(p_1), s_1 - \mathfrak{g}_1(p_1) + \mathfrak{g}_1(p'_1)), \ldots, (\mathfrak{h}_n(p_n), s_n - \mathfrak{g}_n(p_n) + \mathfrak{g}_n(p'_n))\}$. If $p \approx_t p'$ then he obtains in S at least t proper points belonging to P, and can derive the password $P(0)$. Otherwise, informally speaking, the client needs to solve the problem from Assumption 3.1.

Theorem 3.2 (Local PR Security). *An adversary A attacking PR from Figure 2 first produces two passwords p_0, p_1, and sends them to an oracle. Then the oracle chooses $b \in_R \{0,1\}$, performs password registration for p_b, and sends the result back. Finally, A outputs his guess of b.*

A succeeds with some probability $\frac{1}{2} + a$. We denote his advantage as a. Working in ROM, no A having non-negligible advantage exits under Assumption 3.1.

Proof (sketch). Assume to the contrary that there exists an adversary A, that attacks our local PR with non-negligible advantage. Using A, we construct an adversary A^* that breaks Assumption 3.1. Firstly, A sends p_0, p_1 to A^*. A^* forwards them to an intraceability oracle (corresponding to Assumption 3.1). This oracle chooses $b \in_R \{0,1\}$, and answers with n subsets $S_i = \{(x_{(i,j)}, y_{(i,j)})\}$ sampled from $C^{t,p_b}_{n,|\mathbb{D}|}$. Now A^* sends to A: $v \in_R \{0,1\}^k$, and n random points in $\mathbb{F}$: $\{r_1, \ldots r_n\}$. A^* defines random oracles (representing $\mathfrak{h}_i$ and $\mathfrak{g}_i$) in the following way: for all $j \in \mathbb{D}$ and $i \in \{1, \ldots n\}$: $RO_{\mathfrak{h}_i}(j) = x_{(i,j)}$ and $RO_{\mathfrak{g}_i}(j) = y_{(i,j)} - r_i$. A^* outputs the result of A. Notice, the importance of the implicit random parameter v, which lets random oracles, for two different PR runs, have different outputs (even for the same password).

Because of working in ROM, the distribution of A's input, created in such a way by A^* for p_b, is identical to the distribution of the client's input created in password registration (from Figure 2) for p_b. Therefore, A^*'s advantage is equal to A's advantage, and Assumption 3.1 is broken. □

4 Equivocable Threshold Cryptosystem

In this section we define an equivocable threshold encryption (TE) scheme, and we present a slightly modified threshold ElGamal scheme (based on [17], and the "normal" ElGamal scheme [10]) that is equivocable. Subsequently, in Sections 6 and 7 we use this scheme to solve the PR problem.

In [8] a standard TE scheme consists of the following components. *A key generation algorithm KG* takes as input a security parameter k, the number of

decryption servers n, the threshold parameter t and randomness; it outputs a public key pk, a list $\alpha_1, \ldots \alpha_n$ of private keys, and a list $vk_1, \ldots vk_n$ of verification keys. *An encryption algorithm Enc* takes as input the public key pk, randomness and a plaintext m; it outputs a ciphertext c. *A share decryption algorithm SD* takes as input the public key pk, an index $i \in \{1, \ldots n\}$, the private key α_i and a ciphertext c; it outputs a decryption share c_i (called also partial decryption) and a proof of its validity pr_i. Finally, *a combining algorithm CM* takes as input the public key pk, a ciphertext c, a list $c_1, \ldots c_n$ of decryption shares, a list $vk_1, \ldots vk_n$ of verification keys, and a list $pr_1, \ldots pr_n$ of validity proofs. It performs decryption using any subset of $\{c_1, \ldots c_n\}$ of size t, for which the corresponding proofs are verified. If there is no such set then CM fails.

An equivocable TE scheme consists of the same components as above, but: KG does not produce verification keys, SD does not produce validity proofs, and validity proofs are not part of CM's input. Therefore, CM simply checks if a decryption is possible for any subset $c_{i_1}, \ldots c_{i_t}$ (that is $\binom{n}{t}$ checks).

A secure equivocable TE scheme should fulfill the standard TE security definition called threshold CPA [8]. Notice, that omitting validity proofs does not help a malicious combiner to decrypt, because he possesses less data than for standard TE. A secure equivocable TE scheme moreover has the following properties. After any number of CM invocations, a malicious combiner (which does not know any secret shares) gains no information about: (1) the plaintexts in unsuccessful runs (semantic security) and (2) the shares used in unsuccessful runs for producing partial decryptions. We formalize this intuition in Definition 4.1.

Definition 4.1 (Equivocable Security). *Define an oracle O. Firstly, O performs algorithm KG (for the parameters stated above). Then O can be accessed by the following procedures:*
$S(m)$; returns: an encryption c of m, and correct decryption shares $c_1, \ldots c_n$.
$I(m, i_1, \ldots i_{t-1})$, where $i_1, \ldots i_{t-1} \in \{1, \ldots n\}$ and $|\{i_1, \ldots i_{t-1}\}| = t - 1$; produces an encryption c of m, and $x_1, \ldots x_n$, where $x_i = c_i = SD(pk, i, \alpha_i, c)$ if $i \in \{i_1, \ldots i_{t-1}\}$, and $x_i = SD(pk, i, r_i, c)$ (where r_i is a random value) otherwise; returns $c, x_1, \ldots x_n$.
$F(m)$; returns $c, SD(pk, 1, r_1), \ldots SD(pk, n, r_n, c)$; every r_i is a random value.

First game (corresponds to property 1):

1. *O invokes KG, and sends a public key to a malicious combiner C_1.*
2. *C_1 sends a message m to the oracle O, which returns $S(m)$. This step is repeated as many times as the combiner wishes.*
3. *C_1 chooses m_0, m_1 and sends them to the oracle.*
4. *C_1 chooses $i_1, \ldots i_{t-1} \in \{1, \ldots n\}$, and sends them to O, which chooses $b \in_R \{0, 1\}$. Then O sends back $I(m_b, i_1, \ldots i_{t-1})$. This step is repeated as many times as the combiner wishes.*
5. *C_1 repeats Step 2, and finally, outputs his guess of b.*

No polynomial time adversary C_1 guesses b with a non-negligible advantage.

Second game (corresponds to property 2):

1. *O invokes KG, and sends a public key to a malicious combiner C_2.*
2. *The same like Step 2 of C_1.*

3. *C_2 chooses m and sends it to the oracle.*
4. *C_2 chooses $i_1, \ldots i_{t-1} \in \{1, \ldots n\}$, and sends them to O, which chooses $b \in_R \{0,1\}$. Then O sends back $I(m, i_1, \ldots i_{t-1})$ if $b = 0$, and $F(m)$ otherwise. This step is repeated as many times as the combiner wishes.*
5. *C_2 repeats Step 2, and finally, outputs his guess of b.*

No polynomial time adversary C_2 guesses b with a non-negligible advantage.

4.1 ElGamal Equivocable TE Scheme

In this section we introduce our version of the ElGamal scheme and prove that this version is securely equivocable.

Let $\mathbb{G} = <g>$ denote a finite cyclic (multiplicative) group of prime order q for which the Decision Diffie-Hellman (DDH) problem is assumed to be infeasible: given $g^\alpha, g^\beta, g^\gamma$, where either $g^\gamma \in_R \mathbb{G}$ ($\in_R$ means that a value is chosen uniformly at random from a set) or $\alpha\beta = \gamma \mod q$, it is infeasible to decide whether $\alpha\beta = \gamma \mod q$. This implies that the computation Diffie-Hellman problem, which is to compute $g^{\alpha\beta}$ given $g^\alpha, g^\beta \in_R \mathbb{G}$, is infeasible as well. In turn, this implies that the Discrete Log problem, which is to compute $\log_g h = \alpha$ given $g^\alpha \in_R \mathbb{G}$, is infeasible. We use the group $\mathbb{G}$ defined as the subgroup of quadratic residues modulo a prime p, where $q = (p-1)/2$ is also a large prime. This group is believed to have the above properties.

In the ElGamal scheme the public key consists of q, a generator g of $\mathbb{G}$, and $h = g^\alpha$, while the private key is $\alpha \in \{0, \ldots q-1\}$. For this public key, a message $m \in \mathbb{G}$ is encrypted as a pair $(a, b) = (g^r, mh^r)$, with $r \in_R \mathbb{Z}_q$. Encryption is multiplicatively homomorphic: given encryptions (a, b), (a', b') of messages m, m', respectively, an encryption of $m * m'$ is obtained as $(a, b) * (a', b') = (aa', bb') = (g^{r+r'}, m * m' * h^{r+r'})$. Given the private key $\alpha = \log_g h$, decryption of $(a, b) = (g^r, mh^r)$ is performed by calculating $b/a^\alpha = m$.

ElGamal semantic security can be defined using the following game. An oracle first sends $pk = (q, g, h)$ to an adversary. Then the adversary sends plaintexts $m_0, m_1 \in \mathbb{G}$ to the oracle, which answers, for $b \in_R \{0, 1\}$, with $(g^r, m_b h^r)$. Finally, the adversary guesses b. The scheme is semantically secure if the adversary's advantage is negligible. The ElGamal scheme achieves semantic security under the DDH assumption.

In this paper we use a (t, n)-threshold ElGamal cryptosystem based on [17], in which encryptions are computed using a public key $pk = (q, g, h)$, while decryptions are done using a joint protocol between n parties. The ith party holds a share $\alpha_i \in \mathbb{Z}_q$ of the secret key $\alpha = log_g h$, where the corresponding $h_i = g^{\alpha_i}$ can be made public. As long as at least t parties take part, decryption succeeds, whereas less than t parties are not able to decrypt.

We set the shares as follows: the dealer makes the polynomial $f(x) = \sum_{i=0}^{t-1} a_i x^i \mod q$, by picking $a_i \in_R \mathbb{Z}_q$ (for $0 < i < t$) and $a_0 = f(0) = \alpha$. In the original scheme, the ith share is $\alpha_i = f(i)$, while in our scheme $\alpha_i = f(x_i)$, and each $x_i \in_R \mathbb{Z}_q$ is made public. The schemes security is based on linear secret sharing [18]: t points of a polynomial of degree $t-1$ are sufficient to recover the polynomial and less points give no knowledge about $f(0)$.

The reconstruction of plaintext can be performed in the following way. For some $c = (g^r, mh^r)$, it is required to have t proper partial decryptions $g^{r\alpha_i}$ and x_i, which can be combined to compute (for any x_0):

$$g^{rf(x_0)} = \prod_{i \in S} g^{r\alpha_i \lambda^S_{x_0,i}} \mod p \text{ where } \lambda^S_{x_0,i} = \prod_{i' \in S \setminus i} \frac{x_0 - x_i}{x_i - x_{i'}} \in \mathbb{Z}_q \qquad (1)$$

Hence, because $g^{rf(0)}$ can be computed, c can be decrypted as follows: $mh^r / g^{r\alpha} = m$. Equation 1 describes a polynomial interpolation in the exponent.

We now show that our TE scheme is equivocable with respect to Definition 4.1 under the DDH assumption. For simplicity, we assume that the combiner receives only the data from unsuccessful invocations. However, the successful ones can be handled in a similar way to the security proof of [17]. We prove some lemmas, and then based on them we show that our scheme is equivocable.

Lemma 4.2 (Run Independence). *We define the following game. Firstly, an adversary A gets from an oracle a public key $pk = (q, g, g^\alpha)$, and parameters t, n. Secondly, the oracle: chooses $b \in_R \{0,1\}$, prepares a list of shares $\{(x_1, \alpha_1), \ldots (x_n, \alpha_n)\}$ with secret key α, and sends $x_1, \ldots x_n$ to A. Then, A chooses two plaintexts p_0 and p_1, and sends them to the oracle. Now, A repeats as many times as he wishes the following step: A chooses any $i_1, \ldots i_{t-1} \in \{1, \ldots n\}$ and sends them to an oracle, which returns: $g^r, p_b * g^{r\alpha}, g^{r\alpha_{i_1}}, \ldots g^{r\alpha_{i_{t-1}}}$, where $r \in_R \mathbb{Z}_q$ is chosen by the oracle. Finally, A outputs his guess of b.*

No polynomial adversary A guesses b with non-negligible advantage under the DDH assumption.

Proof (sketch). Assume that A asks the oracle for partial decryptions at most d times (where d is polynomial in k). For simplicity, we assume here that $n = t = 2$ and $d = 2$. The proof for greater n, t, and d can be made similarly.

Assume to the contrary that there exists an A, that wins the game with a non-negligible advantage a. Using A we construct an adversary A^* that breaks the ElGamal semantic security. Firstly, A^* receives a public key $pk = (q, g, g^\alpha)$ from a "semantic security" oracle, and forwards it to A. A^* also generates $x_1, x_2 \in_R \mathbb{Z}_q$ and sends them to A. Then A chooses plaintexts p_0, p_1, and sends them to A^*. Subsequently, A^* forwards them to the oracle, which answers with $g^{r_1}, p_b g^{r_1\alpha}$. Now, A^* chooses $j \in_R \{0,1\}$ and $\alpha_j \in_R \mathbb{Z}_q$. A^* computes, using Equation 1, such $g^{\alpha_{j \oplus 1}}$ that points: $\{(0, \alpha), (x_1, \alpha_1), (x_2, \alpha_2)\}$ define a polynomial of degree 1. Then A^* chooses $b' \in_R \{0,1\}$, and a random permutation $\pi : \{1,2\} \to \{1,2\}$.

Subsequently, A asks for partial decryptions. When A asks eth time (1st or 2nd time) and $\pi(e) = 1 \wedge i_1 = j$ then A^* answers: $g^{r_1}, p_b * g^{r_1\alpha}, g^{r_1\alpha_j}$. If $\pi(e) = 1$ and $i_1 \neq j$ then A^* halts and outputs a random bit. Eventually, if $\pi(e) \neq 1$ then A^* sends to A (for $r \in_R \mathbb{Z}_q$): $g^r, p_{b'} * g^{r\alpha}, g^{r\alpha_2}$. Finally, A^* returns A's output.

Notice that in the case $\pi(e) = 1$, the probability that $i_1 \neq j$ (and the attack stops with a random output) is $\frac{1}{2}$. Assume that it does not happen. Note, that if $b' = b$ then A's input is well constructed and the probability that A outputs b is $\frac{1}{2} + a$. Otherwise, because of the random permutation π, A's input is distributed independently of b (even if the adversary asks less than $d = 2$ times). Thus,

the probability of A guessing correctly is $\frac{1}{2}$ in this case. Therefore, the A^*'s advantage is $a/4$. □

The proof for greater n and t is easy: A^* can simply produce more data α_i. In the case of $d > 2$, the proof is modified as follows. A^* chooses randomly $t-1$ indexes and the corresponding shares. Then A^* chooses $b' \in_R \{0, \dots d-1\}$, and constructs the answer to the eth question of A ($1 \leq e \leq d$) as follows. If $\pi(e) = 1$ (π is a random permutation of set $\{1, \dots d\}$) then, if A^* knows $\alpha_{i_1}, \dots \alpha_{i_{t-1}}$, then A^* answers with $g^r, p_b * g^{r\alpha}, g^{r\alpha_{i_1}}, \dots g^{r\alpha_{i_{t-1}}}$. If $\pi(e) = 1$ and A^* does not have corresponding shares then A^* finishes and outputs a random bit. Otherwise ($\pi(e) > 1$), A^* answers (using Equation 1) with:

$$g^r, p_x * g^{r\alpha}, g^{r\alpha_{i_1}}, \dots g^{r\alpha_{i_{t-1}}} \begin{cases} x = 0 & \text{if } \pi(e) - 1 \leq b' \\ x = 1 & \text{otherwise} \end{cases}$$

Finally, A's result is returned by A^*.

This construction ensures that A's input is either well constructed or, because of the permutation π, is produced independently of b. The probability of not returning a random bit (when $\pi(e) = 1$) is $1/\binom{n}{t-1}$, and is non-negligible in k. Details of this constructions are quite straightforward, and we omit them here.

Lemma 4.3 (Run Indistinguishability). *We define the following game. Firstly, an adversary A gets from an oracle a public key $pk = (q, g, g^\alpha)$, and parameters t, n. Secondly, the oracle: chooses $b \in_R \{0, 1\}$, prepares a list of shares $\{(x_1, \alpha_1), \dots (x_n, \alpha_n)\}$ with a secret key α, and sends $x_1, \dots x_n$ to A. Now, A repeats as many times as he wishes the following step. A chooses a set $I = \{i_1, \dots i_{t-1}\}$ (where each $i_f \in \{1, \dots n\}$ and $|I| = t-1$) and sends it to the oracle. If $b = 0$ then the oracle chooses $r \in_R \mathbb{Z}_q$ and answers with: $g^r, g^{r\alpha}, g^{r\alpha_{i_1}}, \dots g^{r\alpha_{i_t}}$. Otherwise the oracle chooses $r, r_1, \dots r_{t-1} \in_R \mathbb{Z}_q$ and answers with: $g^r, g^{r\alpha}, g^{rr_2}, \dots g^{rr_{t-1}}$. Finally, A outputs his guess of b.*

No polynomial adversary A guesses b with non-negligible advantage under the DDH assumption.

The proof sketch of this lemma is in the Appendix B.

Corollary 4.4. *We define the following game. Firstly, an oracle: chooses $b \in_R \{0, 1\}$, generates a public key $pk = (q, g, g^\alpha)$, and a list of random elements (in $\mathbb{Z}_q$): $\{(x_1, \alpha_1), \dots (x_n, \alpha_l)\}$. Secondly, the oracle sends l, pk, and $x_1, \dots x_l$ to an adversary A. The following action is repeated as many times as A wishes: if $b = 0$ then the oracle chooses $r \in_R \mathbb{Z}_q$ and sends to A: $g^r, g^{r\alpha}, g^{r\alpha_1}, \dots g^{r\alpha_l}$. Otherwise the oracle chooses $r, r_1, \dots r_l \in_R \mathbb{Z}_q$ and sends: $g^r, g^{r\alpha}, g^{rr_1}, \dots g^{rr_l}$. Finally, A outputs his guess of b.*

No polynomial adversary A that guesses b with non-negligible advantage exists under the DDH assumption.

Proof. Follows directly from Lemma 4.3 for parameters $t = l$ and $n = l + 1$. □

Now based on Lemmas 4.2, 4.3, we show that our TE scheme is equivocable.

Theorem 4.5 (ElGamal Equivocable TE Scheme). *The ElGamal TE scheme described above in Section 4.1 is equivocable with respect to Definition 4.1 under the DDH assumption.*

Proof. Successful combining invocations can be handled like in the security proof from [17]. This theorem, for unsuccessful invocations, follows directly from Lemma 4.2 for the first game, and from Lemma 4.3 for the second game. □

5 Problems with Using Robust Fuzzy Extractors and Secure Sketches for Client-Server PR

In this section we show the main problems of using secure sketches or fuzzy extractors solving client-server PR in our strongly secure model. Secure sketches and fuzzy extractors (see [6]) can be used for turning noisy information into cryptographic keys and securely authenticating biometric data.

Now, let's define secure sketches and fuzzy extractors. Let $\mathbb{F}$ be a field, $n \in \mathbb{N}$, and Δ a Hamming distance function in $\mathbb{F}^n$. An $(\mathbb{F}^n, m, m', \tau)$-secure sketch is a pair of procedures, "sketch" (SS) and "recover" (Rec), with the following properties. Firstly, SS on input $w \in \mathbb{F}^n$ returns a bit string $s \in \{0,1\}^*$. Secondly, the procedure Rec takes an element $w' \in \mathbb{F}^n$ and a bit string $s \in \{0,1\}^*$. The correctness property guarantees that if $\Delta(w, w') \leq \tau$, then $Rec(w', SS(w))$ equals w. The security property guarantees that for any distribution W over $\mathbb{F}^n$ with min-entropy m, the value of W can be recovered by the adversary who observes s, with probability no greater than $2^{-m'}$.

An $(\mathbb{F}^n, m, l, \tau, \epsilon)$-fuzzy extractor is a pair of procedures, "generate" (Gen) and "reproduce" (Rep), with the following properties. Firstly, the procedure Gen on input $w \in \mathbb{F}^n$ outputs an extracted string $R \in \{0,1\}^l$ and a helper string $P \in \{0,1\}^*$. Secondly, Rep takes an element $w' \in \mathbb{F}^n$ and a string $P \in \{0,1\}^*$ as inputs. The correctness property guarantees that if $\Delta(w, w') \leq \tau$ and P were generated by $(R, P)=Gen(w)$ then $Rep(w', P)=R$. The security property guarantees that for any distribution W over $\mathbb{F}^n$ with min-entropy m, the string R is nearly uniform even for those who observe P. A robust version of fuzzy extractor additionally detects whether the value P got modified by an adversary (which is essential in the biometric authentication).

Secure sketches can be used to solve local PR (Section 3) and client-server PR from Section 6. Roughly speaking, the first case is close to the approach from [9]. Let's consider the second case. The client produces $s=SS(p)$ of his password p and sends it to the server, who stores s. When the client invokes the PR routine by sending p' then the server runs $p''=Rec(p', s)$ and if $p' \approx_t p''$ then the server sends back p''. This solution is sound and secure, i.e, the server can guess p with probability no greater than $2^{-m'}$. However, we do not see a way to transform this solution to the challenge response model, because in this model the server is not allowed to see the password's guesses. We leave finding the transformation of this solution to the challenge response model as a future work.

It would appear that Robust Fuzzy Extractors (RFE) can be used to overcome this problem in, for example, the following way. First the client produces

(R, P)=$Gen(p)$ and $E_R(p)$ (where E is a symmetric encryption scheme, e.g., AES), and he sends P and $E_R(p)$ to the server, who stores them. When the client invokes the PR routine, then the server sends the relevant $P, E_R(p)$ to the client. Now, the client can recover R'=$Rep(p', P)$, and try to decrypt: $Dec_{R'}(E_R(p))$. This solution is sound and seems secure. However, in our security model this protocol gives too much information to the adversary impersonating the client, because it allows an offline dictionary attack. We remind, that the adversary is computationally bounded by k but not $n \log |\mathbb{D}|$. Therefore, the adversary can simply guess l bits (notice that practically always $l < m \leq n \log |\mathbb{D}|$), and break the protocol. Other solutions based on RFE seem to suffer to the same problem.

6 Password Recovery for the Hash based PA System

In this section we present solutions that work for the most widely used PA system. We present first a simple and secure PR scheme, that has a functional drawback: the server's time complexity is too high for many scenarios. Secondly, we show the solution that eliminates this drawback.

6.1 Simple PR System for the Hash based PA System

In the simple PR system the server performs all important security actions. During the registration the client sends to the server the login, and the password p. The server generates the local PR data, like in Section 3. Later, if the client wants to recover p, he sends a perturbed password p' to the server, who runs the local PR routine (Section 3). If the recovery was successful then p is sent to the client and the request is rejected otherwise. The correctness and the security of this protocol follows directly from the corresponding local PR properties.

Notice that the client's privacy is not protected during protocols run (the server even knows the result of PR). Furthermore, there are two significant drawbacks: $\binom{n}{t}$ checks on the server side, and we do not foresee any way to transform this protocol to work in the securer, challenge-response model. These problems are solved in Section 6.2.

6.2 Improved PR System for the Hash based PA System

We improve the simple PR scheme by combining the equivocable TE scheme (Section 4) with local PR. In this solution, the client checks whether the password recovery is possible. Therefore, the server's time complexity is efficient. The improved PR system is presented in Figure 3.

During registration the client first produces a public key (q, g, g^α) of the equivocable TE scheme, with the corresponding secret key α and computes an encryption c of the password p. Subsequently, he generates the PR data: secret values v_1, v_2 (they have the same meaning as v in local PR) and points $\{(\mathfrak{h}_i(p_i), \alpha_i - \mathfrak{g}_i(p_i)) | i \in \{1, \ldots n\}\}$. All the points $\{(\mathfrak{h}_i(p_i), \alpha_i)\}$ together with $(0, \alpha)$ define the polynomial of degree $t-1$. This construction is very similar to the local PR registration. The client also produces the login and the hash of the password for the PA system. Then all these data are stored on the server. Intuitively,

PASSWORD REGISTRATION: The client's input is: *login* and $p = p_1, \ldots p_n$ ($p_i \in \mathbb{D}$); the server's input is his database.

1. The client chooses $v_1, v_2 \in_R \{0,1\}^k$ and
2. generates a public key of the (t,n)-TE scheme (Section 4): $pk = (q, g, h{=}g^{\alpha})$. Then he generates shares: $(x_1, \alpha_1), \ldots (x_n, \alpha_n) \in {\mathbb{Z}_q}^2$ of the secret key α, where $x_i = \mathfrak{h}_i(p_i)$. $\mathfrak{h}$ is MAC (described in Section 2.2) with implicit parameter v_1.
3. The client computes encryption of the password p: $c = (g^r, p * h^r)$, and
4. produces $PR{=}(pk, v_1, v_2, c, \{\alpha_1 - \mathfrak{g}_1(p_1), \ldots \alpha_n - \mathfrak{g}_n(p_n)\})$; $\mathfrak{g}$ is MAC with implicit parameter v_2. Then he sends $(login, H(p), PR)$ (H is from the PA system).
5. The server stores $(login, H(p), PR)$ in his database.

LOG IN: The client sends his *login*, and p to the the server, which accepts the client if $H(p)$ is equal to the corresponding value from the database.

PASSWORD RECOVERY: The client's input is: *login* and $p' = p'_1, \ldots p'_n$ ($p'_i \in \mathbb{D}$); the server's input is his database.

1. The client sends $(login, p')$ to the server.
2. The server performs:
   (a) finds $PR{=}(pk, v_1, v_2, c, \{y_1, \ldots y_n\})$ corresponding to *login* in the database.
   (b) re-randomizes $c = (a, b)$, by $c' = (a', b') = (a{*}g^{r'}, b{*}h^{r'})$. This step ensures that data from different PR runs are independent of each other.
   (c) produces n potential partial decryptions of c': $\forall_{i \in \{1,\ldots n\}} c'_i = {a'}^{y_i + \mathfrak{g}_i(p'_i)}$.
   (d) sends v_1, pk, c', and the partial decryptions $\{c'_1, \ldots c'_n\}$ to the client.
3. Using $\{(\mathfrak{h}_1(p_1), c'_1), \ldots (\mathfrak{h}_n(p_n), c'_n)\}$, the client performs a CM invocation from Section 4. If a decryption p'' matches p' then the client outputs p''.

Fig. 3. An improved PR for the Hash based PA system

the server cannot recover more than in local PR, because he stores the local PR data and an encryption of the password under the secret of the local PR data.

If the client forgets the password then he invokes the PR routine by sending the login and a guess p'. Subsequently, the server produces, using the homomorphic property, a new encryption c' of p. Afterwards, the potential partial decryptions $\{c'_i = {c'}^{y_i + \mathfrak{h}_g(p'_i)} | i \in \{1, \ldots n\}\}$ are produced. Notice, that if $p'_i = p_i$ then $(\mathfrak{h}_i(p_i), c'_i)$ is a proper partial decryption of c'. Later on, the server sends v_1 (so the client can compute $\mathfrak{h}$), c', and $c'_1, \ldots c'_n$. If $p' \approx_t p$, then the client can easily obtain p, because he has at least t proper decryptions. Otherwise, the client does not have enough correct decryptions to obtain p. Moreover, because of the equivocable property of the TE scheme, the client cannot recognize which partial decryptions are correct from the data from many unsuccessful PR runs.

v_1 and v_2 are implicit parameters for $\mathfrak{h}$ and $\mathfrak{g}$, respectively, that are used to make different local PR data indistinguishable. v_1 is public (it is send to the client before any authentication), while v_2 is not revealed to the client, so he cannot locally compute $\mathfrak{g}$.

Correctness and Security. Correctness of the PR phase is straightforward: if $p \approx_t p'$ then at least t partial decryptions are correct and thus, the client can decrypt c'. Otherwise, the client does not have enough partial decryptions of c'.

Theorem 6.1 (The privacy of the client). *An adversary A attacking the privacy of the client from Figure 3 produces two passwords p_0, p_1, and sends them to an oracle. Then the oracle, chooses $b \in_R \{0,1\}$, performs the registration for p_b, and sends the result back. Finally, A outputs his guess of b.*

Working in ROM, no A having non-negligible advantage exits under the DDH assumption and Assumption 3.1.

Proof (sketch). Assuming that the DDH assumptions holds (and thus, the ElGamal is semantically secure), A can break the scheme only by gaining the secret of the local PR data. Following Theorem 3.2, if the local PR security is broken then Assumption 3.1 does not hold.

Theorem 6.2 (The privacy of the server). *Define an ideal situation to be one, in which an adversary tries PR by sending his guess p' of the password p to the server, who returns p if $p' \approx_t p$, and the empty string otherwise. Now, define a simulator as an algorithm that works in the ideal situation, and acts as a server to an adversary A attacking the privacy of the server.*

In ROM and under the DDH assumption, there exists a simulator I such that no adversary A can distinguish between I and the real server (from Figure 3) with non-negligible advantage.

The proof sketch of this lemma is in the Appendix C.

Complexity. During the registration the client sends a public key, two secret values (of length k), the login, the hash of the password, an encryption of the password, and n perturbed shares. The complexity of this phase can be bound by $O(nk)$ bits. In the PR phase the server sends the public key, an encryption of password, and n potential partial decryptions. This totals to $O(nk)$ bits.

The registration is performed efficiently by the participants. In the PR phase the server's performance is fast (main load is n exponentiations), while the client's time complexity involves $\binom{n}{t}$ polynomial interpolations (Step 3).

7 Password Recovery for the Challenge-Response System

In this section we present a PR solution for challenge response login system, where the password or the guess of the password is never sent to the server. We combine the protocol from Section 6.2 with OT_l^n oblivious transfer (see below). The challenge-response PR protocol is shown in Figure 4.

There are two participants in the OT protocol: Receiver, who wants to obtain some information from a remote database and Sender that owns the database. OT can be formalized as follows. During a 2-party 1-out-of-n OT protocol for l-bit strings (OT_l^n), Receiver fetches $S[q]$ from the Sender's database $S = (S[1], \ldots S[n])$, $S[j] \in \{0,1\}^l$, so that a computationally bounded Sender does not know which entry Receiver is learning. Moreover, we assume information-theoretically privacy of Sender (it means that Receiver obtains only desired $S[q]$ and nothing more). Such OT_l^n scheme is presented in [13]. This OT protocol works in bit communication $O(k \log^2 n + l \log n)$, low degree polylogarithmic Receiver's time computation and linear time Sender's computation. This is the fastest oblivious transfer protocol to the best of our knowledge.

PASSWORD REG.: like in Fig. 3, but instead of $H(p)$, values $g, g^{H(p)}$ are sent.
LOGGING IN: like in the challenge-response PA system (Figure 1).
PASSWORD RECOVERY: The client's input is: *login* and $p = p'_1, \ldots p'_n$; $p'_i \in \mathbb{D}$; the server's input is the database.

1. The client sends *login* to the server.
2. The server, using *login*, finds $PR{=}(pk, v_1, v_2, c, \{y_1, \ldots y_n\})$ in the database. Then he re-randomizes $c = (a, b)$: $c' = (a', b') = (a * g^{r'}, b * h^{r'})$ and sends v_1, pk, c'.
3. For $i \in \{1, \ldots n\}$, the client and the server performs OT^b_m protocol, where $|\mathbb{D}|{=}m$ and b is a partial decryption's bit size. The server acts as Sender with the database:
$$S[j] = a'^{y_i + \mathfrak{g}_i(j)}, \text{ for all } j \in \mathbb{D}$$
and the client acts as Receiver with index $q = p_i$. The client's output is $S[q]$.
4. The same like Step 3 in PR from Figure 3.

Fig. 4. A PR protocol for the challenge-response PA system

This system is very similar to the one from Section 6.2. However, the log in routine is different (i.e., the challenge-response one is used), and the PR routine is a bit modified. The client does not send the guess $p'{=}p'_1, \ldots p'_n$ directly to the server. Instead, he obtains partial decryptions corresponding to p' in an oblivious way, as follows. For each $i \in \{1, \ldots n\}$, the server prepares a potential partial decryption c'_i for all possible |D| letters (Step 3). Then the client asks for partial decryptions for guess $p'{=}p'_1, \ldots p'_n$ by performing oblivious transfer n times: for every letter p'_i separately. In this way, the server does not gain information about p', and the client cannot ask for more than one partial decryption per OT protocol. The protocol's security follows from the security of OT and the security properties of the scheme from Section 6.2.

7.1 Correctness and Security

We give an informal intuition about the theorems and the proofs. The proof of the correctness and the privacy of the client outside the protocol runs are the same as for the system from Figure 3. The proof of the privacy of the server is the same as the one for PR from Figure 3, assuming that the OT is secure. The privacy of the client during PR runs is maintained by using OT (the server cannot gain any information about the client guess $p'_1, \ldots p'_n$).

7.2 Complexity

Only the PR phase is significantly different from the system from Figure 3. The major payload comes from n runs of $OT^{O(k)}_{|\mathbb{D}|}$ protocols. This can be bound by $O(n(k \log^2 |\mathbb{D}| + k \log |\mathbb{D}|)) = O(nk \log^2 |\mathbb{D}|)$ bits. The bit complexity of this PR, although greater than the one from Figure 3, is still efficient.

In the PR protocol the time complexity of the client is relatively high and follows from $\binom{n}{t}$ polynomial interpolations. The main drawback of this protocol is the time complexity of the server, who acts as Sender in OT, using $O(n * |\mathbb{D}|)$ operations. However, for the relatively small domain of letters $\mathbb{D}$, and due to

the fact that PR is performed rarely, this solution is still quite feasible. This drawback might be of greater impact if we use this protocol in the personal entropy setting (i.e., the question-answer setting), where $|\mathbb{D}|$ might be larger.

8 Conclusions

In this paper we have presented secure and efficient solutions for password recovery, where the recovery data is stored securely at the server side. Our solutions apply to all common types of password authentication systems, without significantly lowering their security. We have introduced a variant of threshold encryption, called equivocable, that serves as a building block to our solutions, and that may be of independent interest as well.

Further research could be aimed at alternative definitions of password similarity, that also include reordering of password letters (which is a common mistake). Other issues that can be improved are the $\binom{n}{t}$ time complexity at the client side, and the server's time complexity in the challenge-response protocol (Section 7).

References

[1] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *CCS '93: Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73, New York, NY, USA, 1993. ACM.

[2] G.R. Blakley. Safeguarding cryptographic keys. In *AFIPS Conference Proceedings*, volume 48, pages 313–317, June 1979.

[3] Daniel Bleichenbacher and Phong Q. Nguyen. Noisy polynomial interpolation and noisy chinese remaindering. In *EUROCRYPT*, pages 53–69, 2000.

[4] Dan Boneh. Finding smooth integers in short intervals using crt decoding. *J. Comput. Syst. Sci.*, 64(4):768–784, 2002.

[5] Ivan Damgard, M. Jurik, and J. Nielsen. A generalization of paillier's public-key system with applications to electronic voting, 2003.

[6] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *CoRR*, abs/cs/0602007, 2006.

[7] Carl Ellison, Chris Hall, Randy Milbert, and Bruce Schneier. Protecting secret keys with personal entropy. *Future Generation Computer Systems*, 16(4):311–318, 2000.

[8] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *FC '00: Proceedings of the 4th International Conference on Financial Cryptography*, pages 90–104, 2001.

[9] Niklas Frykholm and Ari Juels. Error-tolerant password recovery. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 1–9, New York, NY, USA, 2001. ACM.

[10] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[11] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[12] Bhavana Kanukurthi and Leonid Reyzin. An improved robust fuzzy extractor. In *SCN*, pages 156–171, 2008.
[13] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328. Springer, 2005.
[14] Moni Naor and Benny Pinkas. Oblivious polynomial evaluation. *SIAM J. Comput.*, 35(5):1254–1281, 2006.
[15] Naom Nisan and Amnon Ta-Shma. Extracting randomness: a survey and new constructions. *J. Comput. Syst. Sci.*, 58(1):148–173, 1999.
[16] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT*, pages 223–238, May 1999.
[17] Berry Schoenmakers and Pim Tuyls. Practical two-party computation based on the conditional gate. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 119–136. Springer, 2004.
[18] Adi Shamir. How to share a secret. In *Communications of the ACM, vol. 22, n.11*, pages 612–613, November 1979.

A Simple Substring-Knowledge Password Recovery in the Challenge-Response Setting

In this appendix we present a simple and efficient substring-knowledge challenge-response PR scheme that uses an additively homomorphic encryption scheme. In order for a client to recover a password it needs to prove to the server that he remembers a substring of the original password.

Let $[\cdot]_K$ denote a homomorphic encryption function with a public key K. The homomorphic cryptosystem supports the following two operations, which can be performed without knowledge of the private key. Firstly, given the encryptions $[a]_K$ and $[b]_K$ of a and b, one can efficiently compute the encryption of $a+b$, denoted $[a+b]_K := [a]_K +_h [b]_K$. Secondly, given a constant c and the encryption $[a]_K$ of a, one can efficiently compute the encryption of $c \cdot a$, denoted $[a \cdot c]_K := [a]_K \cdot_h c$. These properties hold for suitable operations $+_h$ and $\cdot_h$ defined over the range of encryption function. An example of such an encryption scheme is Paillier's cryptosystem [16].

In the registration phase the client sends, besides data necessary for logging in, $h_1(p_{1,t}), h_2(p_{2,t+1}), h_{n-t+1}(p_{n-t+1,n})$ (for simplicity, we denote $p_{i,w} = p_i, \ldots p_w$) and $E_{H_1(p_{1,t})}(p), \ldots E_{H_{n-t+1}(p_{n-t+1,n})}(p)$, where $E_{sk}(.)$ is a symmetric encryption scheme and H_i, h_i are hash functions. Notice, that to recover the password p, it is necessary to derive some $h_i(p_i, \ldots p_{i+t-1})$ or $H_i(p_i, \ldots p_{i+t-1})$, and it (assuming ROM) is only possible by obtaining any substring $p_i, \ldots p_{i+t-1}$.

Later on, in the PR phase, the client produces a public key K of the homomorphic encryption scheme, and sends it to the server together with $[h_1(p'_{1,t})]_K, \ldots [h_{n-t+1}(p_{n-t+1,n})]_K$. Then the server computes: $\{[(h_i(p'_{i,i+t-1}) - h_i(p_{i,i+t-1})) * r_i + E_{H_i(p_{i,i+t-1})}(p)]_K | i \in \{1, \ldots n-t+1\}\}$, (where r_i are random values), and sends this set to the client. The client decrypts the values from the received set and checks if he can decrypt these values with any $H_1(p_{i,t}), H_2(p_{2,t+1}), H_{n-t+1}(p_{n-t+1,n})$ (then he derives p).

The scheme is correct, since if $h_i(p'_{i,i+t-1}) = h_i(p_{i,i+t-1})$ then the client obtains $E_{H_i(p_{i,i+t-1})}(p)$, and he can easily decrypt it. Otherwise, the value received is random (because r_i are random) and therefore, the client cannot successfully decrypt. The privacy is protected by the security of the encryption schemes.

B Proof Sketch of Lemma 4.3

Notice that this game can be rephrased as follows. The oracle's first answer is always proper, i.e.: $g^r, g^{r\alpha}, g^{r\alpha_{i_1}}, \ldots g^{r\alpha_{i_{t-1}}}$. Only the following answers are constructed either always properly (if $b = 0$), or always randomly. It follows from the fact that t random values (in the first oracle's answer) always define a polynomial of degree at most $t - 1$.

Proof (sketch). Assume that A asks the oracle for partial decryptions at most d times (where d is polynomial in k). For simplicity, we assume that $n = t = 3$ and $d = 2$. The proof for greater n, t, and d can be made similarly.

Assume to the contrary that A winning the game with non-negligible advantage a, exists. Using A we construct an adversary A^* that breaks the ElGamal security. Firstly, A^* receives a public key (q, g, g^α) from a "semantic security" oracle. Secondly, A^* generates $x_1, x_2, x_3 \in_R \mathbb{Z}_q$ and sends them to A. Then A^* sends plaintexts p_0=1 and $p_1 \in_R \mathbb{G}$ to the oracle, which answers with $g^{r_1}, p_b g^{r_1\alpha}$.

Now, A^* chooses a random permutation $\pi : \{1, 2, 3\} \to \{1, 2, 3\}$ (we denote $j_f = \pi(f)$), and picks $\alpha_{j_1}, \alpha_{j_2} \in_R \mathbb{Z}_q$. Then A^* computes (using Equation 1), such $g^{\alpha_{j_3}}$ that points: $\{(0, \alpha_{j_1}), (x_{j_1}, \alpha), (x_{j_2}, \alpha_{j_2}), (x_{j_3}, \alpha_{j_3})\}$ define a polynomial of degree 2. We denote (for $1 \le i \le 3$): $\alpha'_i = \alpha$ if $i = j$, and $\alpha'_i = \alpha_i$ otherwise. A^* sends a public key $pk' = (q, g, g^{\alpha_{j_1}})$ to A.

When A asks the first time (for partial decryptions) with i_1, i_2 then A^* answers (for $r \in_R \mathbb{Z}_q$) with: $g^r, g^{r\alpha_j}, g^{r\alpha'_{i_1}}, g^{r\alpha'_{i_2}}$. For the second A's question i'_1, i'_2, A^* firstly checks whether $\{i'_1, i'_2\} \neq \{j_1, j_2\}$. If it holds then A^* halts and outputs a random bit. Otherwise A^* first sends $g^{r_1}, g^{r_1\alpha_j}$. Then A^* chooses $b' \in_R \{0, 1\}$, and for every $1 \le e \le 2$, acts as follows. If $i'_e = j_1$ then A^* sends $p_b g^{r_1\alpha}$ to A. If $i'_e = j_2$ and $b' = 0$ then A^* sends $g^{r_1\alpha_{j_2}}$. Otherwise ($i'_e = j_2$ and $b' = 1$): $g^{r_1 x}$ (for $x \in_R \mathbb{Z}_q$) is sent. Finally, A^* returns the A's output.

Notice that the probability that $\{i'_1, i'_2\} \neq \{j_1, j_2\}$ (and that A^* halts with a random output) is $1 - 1/\binom{3}{2}$. Assume that it does not happen. If $b = b'$ then A's input is well constructed and the probability that A outputs b is $\frac{1}{2} + a$. Otherwise, because of the random permutation π, A's input is distributed independently of b. Hence, the probability of A guessing correctly is $\frac{1}{2}$ in this case. Therefore, A^*'s advantage is $a/(2\binom{3}{2})$, and is non-negligible. □

The full proof for this lemma is similar, but complex, and we omit it here due to the space constraints (the proof for $d > 2$ uses similar techniques as in the proof of Lemma 4.2).

C Proof Sketch of Lemma 6.2

Proof (sketch). We construct I that works only for unsuccessful PR invocations. The proof for a successful A's invocation can be made similarly.

Firstly, I generates $v_1 \in_R \{0,1\}^k$, a public key $pk = (q, g, g^\alpha)$, $y \in_R \mathbb{D}^n$, and shares $(x_1, \alpha_1), \ldots (x_n, \alpha_n) \in {\mathbb{Z}_q}^2$ of the equivocable TE scheme (Section 4), such that $x_i = \mathfrak{h}_i(p_i)$. Later, when A sends his ith guess p^i, then I forwards it to the "ideal" oracle. If the oracle's answer equals p then I halts. Otherwise I chooses $r, r_1, \ldots r_n \in_R \mathbb{Z}_q$, and sends: v_1, pk, $c{=}(g^r, yg^{r\alpha})$, $\{c_1{=}g^{rr_1}, \ldots c_n{=}g^{rr_n}\}$ to A.

A can only submit the proper guess of the password (otherwise the server would recognize it). Therefore, A cannot break the protocol by disobeying the PR routine. Hence, now we only need to show that the A's view send by I is indistinguishable from the corresponding view in the real situation.

Let's now consider A in the real situation (Figure 3). Notice that, because A works in ROM, the data received by A in any d unsuccessful PR runs corresponds to the data from d unsuccessful invocations of the algorithm CM in the equivocable TE scheme. The difference is that, here, A does not know which value $\mathfrak{h}_i(p_i)$ (for any $p_i \in \mathbb{D}$) is a part of a share (i.e., equals x_i), while CM correctly knows all x_i. However, Lemmas 4.2, 4.3, and Corollary 4.4 can be applied in A's case, because A has actually less information than the combiner CM.

In every invocation A receives at most $t-1$ correct partial decryptions. Incorrect partial decryptions are created using values independent of α and α_i, because if $p_j \neq p'_j$ then $\alpha_j - \mathfrak{g}_j(p_j) + \mathfrak{g}_j(p'_j)$ is random in $\mathbb{Z}_q$ (in ROM). Therefore, based on Lemma 4.2, A cannot recognize encryptions received in the real situation from encryptions received from I.

Let p be any password from D^n encoded in $\mathbb{G}$, and $p^1, \ldots p^d$ is any list of passwords not similar to p. Consider the following probability distributions of instances of the adversary's view:

- $S_{0,0}$: A receives properly constructed data from the the PR routine (Figure 3) for his guesses $p^1, \ldots p^d$, and for the password p.
- $S_{0,1}$: For every guess p^i, A receives proper v_1, pk, an encryption of p: c, and n values: if $p^i_j = p_j$ $(1 \leq j \leq n)$ then a correct partial decryption c'_i, and $c'_j \in_R \mathbb{G}$ otherwise.
- $S_{1,1}$: similar to $S_{0,1}$, but all $c'_j \in_R \mathbb{G}$ (for every guess); $S_{1,1}$ corresponds to the view sent by I.

We show that no algorithm $\mathfrak{D}(p, p^1, \ldots p^d)$ can distinguish between an input sampled from $S_{0,0}$ and an input sampled from $S_{1,1}$ (under the DDH assumption). Define $\mathfrak{D}_{0,0}$ as the probability that the output of $\mathfrak{D}$ is 1 given an input sampled from $S_{0,0}$. Similarly, we define $\mathfrak{D}_{1,1}$, $\mathfrak{D}_{0,1}$. It holds that

$$|\mathfrak{D}_{0,0} - \mathfrak{D}_{1,1}| \leq |\mathfrak{D}_{0,0} - \mathfrak{D}_{0,1}| + |\mathfrak{D}_{0,1} - \mathfrak{D}_{1,1}|.$$

Assume to the contrary that $|\mathfrak{D}_{0,0} - \mathfrak{D}_{1,1}|$ is non-negligible. Then, either $|\mathfrak{D}_{0,0} - \mathfrak{D}_{0,1}|$ is non-negligible or $|\mathfrak{D}_{0,1} - \mathfrak{D}_{1,1}|$ is non-negligible. In the first case, Corollary 4.4 does not hold. In the second case, Lemma 4.3 does not hold. Therefore, A cannot distinguish I from the real server under the DDH assumption. □