Faster and Shorter Password-Authenticated Key Exchange

Rosario Gennaro
IBM T.J. Watson
19 Skyline Drive,
Hawthorne 10532, USA.
rosario@us.ibm.com

August 17, 2007

Abstract

This paper presents an improved password-based authenticated key exchange protocols in the common reference string model. Its security proof requires no idealized assumption (such as random oracles).

The protocol is based on the GL framework introduced by Gennaro and Lindell, which generalizes the KOY key exchange protocol of Katz et al. Both the KOY and the GL protocols use (one-time) signatures as a non-malleability tool in order to prevent a man-in-the-middle attack against the protocol. The efficiency of the resulting protocol is negatively affected, since if we use regular signatures, they require a large amount of computation (almost as much as the rest of the protocol) and further computational assumptions. If one-time signatures are used, they substantially increase the bandwidth requirement.

Our improvement avoids using digital signatures altogether, replacing them with faster and shorter message authentication codes. The crucial idea is to leverage as much as possible the non-malleability of the encryption scheme used in the protocol, by including various values into the ciphertexts as *labels*. As in the case of the GL framework, our protocol can be efficiently instantiated using either the DDH, Quadratic Residuosity or N-Residuosity Assumptions.

For typical security parameters our solution saves as much as 12 Kbytes of bandwidth if one-time signatures are implemented in GL with fast symmetric primitives. If we use number-theoretic signatures in the GL framework, our solution saves several large exponentiations (almost a third of the exponentiations computed in the GL protocol). The end result is that we bring provable security in the realm of password-authenticated key exchange one step closer to practical.

1 Introduction

The central problem of cryptography is to enable reliable and secure communication among parties in the presence of an adversary. In order to do this, parties must share a common secret key to secure communication using known techniques (e.g., applying encryption and message authentication codes to all messages).

A protocol that allows two parties to establish such a secret key is called a *key exchange* protocol. The key exchange problem was initially studied by Diffie and Hellman [16] who considered a passive adversary that can eavesdrop on the honest parties' communication, but cannot actively modify it. In other words, parties are assumed to be connected by reliable, albeit non-private, channels. Many efficient and secure protocols are known for this scenario. The more realistic scenario however is that of a far more powerful adversary who can modify and delete messages sent between the parties, as well as insert messages of its own choice. This is the scenario we consider in this paper.

Once we allow such a powerful adversary it becomes clear that in order to securely exchange a key, any two parties (call them Alice and Bob) must hold some secret information. Otherwise, there is nothing preventing an adversary from pretending to be Bob while communicating with Alice (and vice versa). The most common type of secret information considered are (i) parties already share a high entropy secret key; (ii) each party holds a secret key matching an authenticated public key (i.e. a public key securely associated with his identity) and (iii) the case we consider in this paper: parties share only a low entropy *password* that can be remembered and typed in by human users.

Cryptography has long been concerned with cases (i) and (ii), while the scenario of low entropy passwords (arguably the most commonly used case) has only recently received attention. This paper proposes a new and improved family of protocols (a framework) for password-based key exchange in the face of a powerful, active adversary.

Password-based authenticated key-exchange. Our model consists of a group of parties, with each pair of them sharing a password chosen uniformly at random from some small dictionary (the assumption of uniformity is made for simplicity only). The parties communicate over a network in the presence of an active adversary who has full control over the communication lines. In other words all communication between parties is basically carried out through the adversary. Nevertheless, the goal of the parties is to generate session keys in order to secretly and reliably communicate with each other.

An immediate observation is that the small size of the dictionary implies a non-negligible probability that the attacker will succeed in impersonating one of the parties, since the adversary can always guess Bob's password and pretend to be him while communicating with Alice. This type of attack is called an on-line guessing attack and is inherent whenever security depends on low entropy passwords. The severity of on-line guessing attacks can be limited with other mechanisms (such as locking an account after a number of failed attempts). A more dangerous attack is the *off-line guessing attack*, in which the adversary obtains a transcript of an execution of the key exchange protocol and is then able to check guesses for Bob's password against this transcript *off-line*. The aim of password-based authenticated key exchange is to limit the adversary only to on-line guessing attacks, and rule out possible off-line ones.

Prior related work. Bellovin and Merritt [4] proposed the first protocol for password-based session-key generation. Although the specific protocol of [4] can be attacked (see [28]), small modifications to the protocol prevent these attacks [28], and even allow one to prove it secure under the ideal cipher and random oracle models [2]. Bellovin and Merritt's work was very influential and was followed by many protocols (e.g. [5, 29, 23, 27, 28, 30]) which, however, have not been proven secure and their conjectured security is based on heuristic arguments.

A first rigorous treatment of the problem was provided by Halevi and Krawczyk [22]. They consider an *asymmetric* model in which some parties (called servers) hold certified public keys available to the all parties, including the clients who instead hold only passwords. In this model (which requires a public-key infrastructure) Halevi and Krawczyk provide a secure password-based key exchange. The first (and only currently known) protocol to achieve security without *any* additional setup is that of Goldreich and Lindell [20]. Their protocol is based on general assumptions (i.e., the existence of trapdoor permutations) and constitutes a proof that password-based authenticated key exchange can actually be obtained. Unfortunately, the protocol of [20] is not very efficient and thus cannot be used in practice.

Katz, Ostrovsky and Yung (KOY) [24] present an efficient and practical protocol for the problem of password-authenticated key-exchange in the common reference string model. In this model, the extra setup assumption is that all parties have access to some public parameters, chosen by some trusted third party. This assumption is clearly weaker than assuming a public-key infrastructure, and there are settings in which it can be implemented safely and efficiently (such as a corporation wanting to provide secure password login for its employees, and thus can be trusted to choose and distribute the common reference string). The KOY protocol is based on the security against chosen-ciphertext attack [17] of the original Cramer-Shoup encryption scheme [12]. This in turn can be reduced to the Decisional Diffie-Hellman (DDH) assumption. The complexity of the KOY protocol is only 5–8 times the complexity of a Diffie-Hellman unauthenticated key-exchange protocol.

The KOY protocol was generalized by Gennaro and Lindell [19], using generic building blocks instead of specific number-theoretic assumptions. More specifically, they use the notion of *projective hash functions* and the CCA-secure encryption schemes defined in [13]. The resulting protocol GL has a much more intuitive proof of security and can be proven secure under a variety of computational assumptions (such as Quadratic Residuosity and N-Residuosity).

We note that there are password-authenticated key-exchange protocols which are more efficient than KOY and GL , but whose proof holds in an idealized model of computation such as the ideal cipher and random oracle models [2, 7]. The common interpretation of such results is that security is *likely* to hold even if the random oracle is replaced by a ("reasonable") concrete function known explicitly to all parties (e.g., SHA-1). However, it has been shown that it is impossible to replace the random oracle in a generic manner with any concrete function [8]. Thus, the proofs of security of these protocols are actually heuristic in nature.

1.1 Our Contributions

We improve on the both the GL and KOY protocols, in particular by reducing the communication bandwidth required by the protocol.

Both the KOY and the GL protocol use (one-time) signatures as a non-malleability tool in order to prevent a man-in-the-middle attack against the protocol. This negatively affects the efficiency of the resulting protocol. Indeed in order to preserve provable security without use of the random oracle an implementation of the KOY or GL protocol is presented with two choices.

One-time signature schemes (i.e. signature schemes which are secure if the key is used to sign only one message) can be implemented from fast symmetric key primitives (such as one-way functions). However the length of the resulting keys and signatures is problematic and causes a substantial increase in the required bandwidth.

One could use "regular" signature schemes (i.e. secure for many messages) but then, if we require provable security in the standard model, the amount of computation would substantially increase. Moreover if we want to use the most efficient provably secure signature schemes in the literature (e.g. [18, 15, 25]) we would introduce new computational assumptions such as the Strong RSA assumption, on top of the ones required by the GL protocol.

Our improvement avoids using digital signatures altogether, replacing them with faster and shorter message authentication codes. The crucial idea is to leverage as much as possible the non-malleability of the encryption scheme used in the protocol, by including various values into the encryption as *labels*. For typical security parameters our improvement saves as much as 12 Kbytes of bandwidth in a protocol execution.

As in the case of the GL framework, our protocol can be efficiently instantiated using either the DDH, Quadratic Residuosity or N-Residuosity Assumption.

1.2 Our Construction in a Nutshell

Let us describe our construction informally. We start by first describing the tools that we are going to use and then describing the protocol.

Chosen-Ciphertext Secure Public-Key Encryption [17]: We use an encryption scheme $\mathcal{E}$ which is secure against chosen-ciphertext attack. The common reference string for our password protocol is simply the public key PK for such an encryption scheme. We stress that the corresponding secret key does not have to be known by any party[1].

Smooth projective hashing [13]: Let X be a set and $L \subset X$ a language. Loosely speaking, a hash function H_k that maps X to some set is projective if there exists a projection key that defines the action of H_k over the subset L of the domain X. That is, there exists a projection function $\alpha(\cdot)$ that maps keys k into their projections $s = \alpha(k)$. The projection key s is such that for every $x \in L$ it holds that the value of $H_k(x)$ is uniquely determined by s and x. In contrast, nothing is guaranteed for $x \notin L$, and it may not be possible to compute $H_k(x)$ from s and x. A smooth projective hash function has the additional property that for $x \notin L$, the projection key s actually says *nothing* about the value of $H_k(x)$. More specifically, given x and $s = \alpha(k)$, the value $H_k(x)$ is uniformly distributed (or statistically close) to a random element in the range of H_k.

What makes smooth projective hashing a powerful tool (in both our application and the original one in [13]) is that if L is an NP-language, then for every $x \in L$ it is possible to efficiently compute $H_k(x)$ using the projection key $s = \alpha(k)$ and a witness of the fact that $x \in L$. Alternatively, given k itself, it is possible to efficiently compute $H_k(x)$ even without knowing a witness. Gennaro and Lindell [19] also prove another important property of smooth projective hash functions that holds when L is a hard-on-the-average NP-language. For a random $x \in_R L$, given x and $s = \alpha(k)$ the value $H_k(x)$ is *computationally indistinguishable* from a random value in the range of $H_k(x)$. Thus, even if $x \in L$, the value $H_k(x)$ is pseudorandom, unless a witness is known.

The basic idea behind the KOY and GL protocols is to have the parties exchange non-malleable encryptions of the joint password. The session key is the computed as the result of applying smooth projective hash functions to these encryptions (in this case the hard-on-the-average NP

[1]In the GL protocol the requirement is actually weaker, as all is needed is a non-interactive non-malleable (with respect to many commitments) commitment which in the common reference string can be built out of CCA-Secure Encryption. For simplicity we describe our protocol using encryption, but in the final version we show that we can also use commitments. In practice this does not make much difference since CCA encryption is the most efficient known implementation of for this type of non-malleable commitments.

language consists of correct ciphertext/message pairs). Figure 1 shows the basic layout of the protocol.

The basic problem with the protocol described in Figure 1 is that the projective hash function themselves can be malleable, and an adversary could manage to get information about the session key by playing man-in-the-middle. In order to avoid this attack, the GL and KOY protocols add a signature step. A verification key is chosen by party A in the first message and *bound* together with the first encryption, by including it as a *label*[2]. Then A signs the whole transcript in the third message. Party B accepts only if the signature is correct. Since the verification key cannot be changed (being protected by the non-malleability of the encryption in the first step), the adversary cannot modify the projection keys, unless it is able to produce a forgery.

In our protocol we expand the use of encryption labels. We protect the first projection s, by including it as a label in the second ciphertext[3] c'. Now the adversary is left with the possibility of manipulating the second projection key s'. But at this point the master key computed by party A is already pseudorandom for the adversary and thus it can be used as a key to MAC the projection key s' in order to prevent $\mathcal{A}$ from changing it.

A technical issue arises here, as party B has to use the same key that party A uses to compute the MAC, but party B has to yet finish the protocol and compute such key. Moreover the adversary can make B compute a different key from A, by modifying the projection s'. This issue can be solved by using sk_B as a MAC key since B already knows it. However the explicit use of only one component of the session key would allow an off-line attack from $\mathcal{A}$ (see Section 4).

The final solution is to MAC the transcript with sk_B and then use sk_A to "mask" the value of the MAC from the adversary. In the proof if B accepts after $\mathcal{A}$ modified s' he will be able to retrieve a forgery on the MAC keyed with sk_B. The protocol in full details is shown in Figure 2.

1.3 Efficiency gain.

The most efficient implementation of the KOY or GL protocols uses one-time signatures. These are based on symmetric primitives such as one-way functions. One example of such a signature is the Lamport signature [26]: to sign a single bit b the public key consists of two values y_0, y_1 and the secret key is x_0, x_1 where $y_i = F(x_i)$ for a one-way function F. To sign bit b the signer reveals x_b.

Assuming a security parameter of 128 (e.g. a one-way function applied to 128 bits input, and messages hashed to 256 bits using a collision resistant function), we have that transmitting the key and the signature requires about 12 KBytes. In contrast our solution requires only 256 bits for the MAC.

Number-Theoretic Signatures. Of course one could implement the signature step in the KOY or GL protocol using provably secure signature schemes such as Gennaro-Halevi-Rabin [18] or Cramer-Shoup [15] which are based on the Strong RSA Assumption: they not only introduce another computational assumption for the security of the scheme, but require several modular exponentiations and about 4 Kbit of bandwidth to transmit keys and signatures. A shorter alternative would be the Boneh-Boyen [6] which requires only 160 bit for the signature, but it would still require 2 Kbits to send the verification key. Moreover signature verification in the Boneh-Boyen scheme is particularly expensive since it requires the computation of a bilinear map.

[2] A *label* is a public string that accompanies a ciphertext and is an integral part of it. It must be submitted together with the ciphertext in order to obtain a decryption and the adversary should not be able to modify it. See Section 2.1 for details.

[3] Interestingly this is already done in the KOY protocol, but it is not used in the proof in any significant way. Indeed the GL proof shows that the use of digital signatures make this step unnecessary. We reinstate it exactly because we want to avoid using signatures.

Insecure Password-Authenticated Key Exchange

- **Common reference string:** The public key PK for a chosen-ciphertext secure encryption scheme $\mathcal{E}$. A description of a smooth projective hashing family H_k over the set X of ciphertext/password pairs (c, w).
- **Common input:** a shared (low-entropy) password w.
- **The protocol:**
  1. Party A computes an encryption $c = \mathcal{E}_{PK}(w)$ and sends it to party B.
  2. Party B chooses a key k for the smooth projective hash function, and computes its projection $s = \alpha(k)$. Also B computes the projective hash over (c, w), i.e. $sk_B = H_k(c, w)$.
     Finally B computes another encryption of the password i.e., $c' = \mathcal{E}_{PK}(w)$.
     B sends s, c' to party A.
  3. Party A chooses another key k' for the smooth projective hash function, and computes its projection $s' = \alpha(k')$. Also A computes the projective hash over (c', w), i.e. $sk_A = H_{k'}(c', w)$.
     A sends s', t to party B.
- **Session Key Definition:**
  1. Party B computes sk_A using the projection s' and its knowledge of a witness for the fact that c' is an encryption to the password w (it knows a witness because it generated c') and outputs the session key $sk = sk_A \oplus sk_B$.
  2. Party A also computes sk_B using the projection s and its knowledge of a witness for the fact that c is an encryption to the string w (it knows a witness because it generated c) and outputs $sk = sk_A \oplus sk_B$.

Figure 1: Common skeleton of KOY , GL and our protocol

The above signatures are secure against many messages. There are more efficient number-theoretic one-time signatures such as the one obtained by a chain of length two in the GMR scheme [21], or the one recently proposed in [11] based on chameleon hashing with two trapdoors. Still because of the computation of modular exponentiations and the transmission of verification key and signature, these options are much more expensive then sending a simple MAC. It is not hard to see that for each one of these options the reduction in the number of exponentiations is at least a third.

1.4 Organization

In section 2 we recall the cryptographic tools that we need: chosen-ciphertext secure public-key encryption, and message authentication codes. In Section 3 we review the notions of smooth projective hash functions. The protocol is then presented in Section 4 and some concluding remarks are presented in Section 5.

For lack of space we recall the formal definition of password-based authenticated key exchange, in Appendix A and the security proof of our protocol appears in Appendix B.

2 Cryptographic Tools

We denote by n the security parameter.

If S is a set, with $|S|$ we denote its cardinality. $|m|$ denotes the bit length of m, if m is a string or a number. If m is a $2n$-bit string we denote with $m^{(1)}$ and $m^{(2)}$ the first and second half of it respectively.

If $A(\cdot,\cdot,\cdots)$ is a probabilistic algorithm, then $x \in_R A(x_1, x_2, \cdots)$ denotes the experiment of running A on input $x_1, x_2, \cdots$ with x being the outcome. If S is a set, $x \in_R S$ denotes the experiment of choosing $x \in S$ uniformly at random. If X is a probability distribution over S then $x \in_R X$ denotes the experiment of choosing $x \in S$ according to the distribution X.

Finally, we denote statistical closeness of probability ensembles by $\stackrel{\mathrm{s}}{\equiv}$, and computational indistinguishability (with respect to non-uniform polynomial-time machines[4]) by $\stackrel{\mathrm{c}}{\equiv}$.

We say that a real-valued function $\epsilon(\cdot)$ defined over the integers is *negligible* if for every constant $c \geq 0$ there exists an integer n_c such that for all $n > n_c$ $\epsilon(n) < n^{-c}$.

2.1 Chosen-Ciphertext Secure Public-Key Encryption

A public key encryption scheme is a tuple of three algorithms $\mathsf{PKE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key generation algorithm $\mathcal{K}$ generates a pair $(PK, SK) \in_R \mathcal{K}(1^n)$, where PK is a public key and SK is a secret key.

We use *labeled* encryption, which means that the encryption algorithm $\mathcal{E}$ takes a public key PK a plaintext m, and a label ℓ and returns a ciphertext/label pair $c \in_R \mathcal{E}_{PK}(m, \ell)$. The decryption algorithm $\mathcal{D}$ takes a secret key SK, a ciphertext c and a label ℓ, and returns $\mathcal{D}_{SK}(c, \ell)$ which is either a message m or $reject$.

The adaptive chosen ciphertext attack (IND-CCA) game is defined similarly. A key pair is generated by the key generation algorithm: $(PK, SK) \in_R \mathcal{K}(1^n)$. Then a PPT adversary $\mathcal{A}$, on input the public key PK, queries a pair of equal length messages m_0 and m_1 and a label ℓ^* to an encryption oracle. The encryption oracle chooses $b \in_R \{0, 1\}$ and computes a challenge ciphertext $c^* \in_R \mathcal{E}_{PK}(m_b, \ell^*)$, which is given to $\mathcal{A}$. In the course of the game the adversary A is given access to a decryption oracle, $\mathcal{D}_{SK}(\cdot,\cdot)$ which A can query on any ciphertext/label pair except the challenge ciphertext/label pair c^*, ℓ^*. The game ends with the adversary outputting a bit $\tilde{b}$.

We say that the encryption scheme is secure against (adaptive) chosen-ciphertext attack if for any adversary $\mathcal{A}$, the probability that $b = \tilde{b}$ is negligible (in the security parameter n).

Notice that the adversary is allowed to query the decryption oracle on *any* ciphertext/label pair which is not the target pair. In particular this definition guarantees that the adversary will not get any information from querying a ciphertext with a label different from the one used when the ciphertext was created.

Notice that in our notation the first argument of the encryption algorithm is always the message, the second argument is the label, and the random coins are implicit.

2.2 Message Authentication Codes

A message authentication code MAC is a function

$$MAC : \{0, 1\}^n \times \{0, 1\}^* \longrightarrow \{0, 1\}^n.$$

The first input is the key $k \in \{0, 1\}^n$, and the second input is the message $m \in \{0, 1\}^*$. The output is called a "tag" $t = MAC_k(m)$.

The chosen message attack (CMA) game is defined as follows. A key is selected uniformly at random $k \in_R \{0, 1\}^n$. The adversary $\mathcal{A}$ is given $t^* = MAC_k(m^*)$ for many adaptively adversarially

[4] All of our results also hold with respect to uniform adversaries.

chosen m^*, after which the adversary outputs a pair (m,t). We say that (m,t) is a *forgery* if $m \neq m^*$ for all the queried m^* and $t = MAC_k(m)$.

We that a MAC is secure if for every adversary $\mathcal{A}$ the probability of computing a forgery is negligible.

3 Smooth Projective Hash Functions

Following Gennaro and Lindell [19] we use a modified version of the notion of *smooth projective hashing* introduced by Cramer and Shoup [13]. We recall the definition from [19] which is needed here and refer the reader to [19] for a description of the differences between this definition and the original one from [13].

Subset membership problems. Intuitively, a hard subset membership problem is a problem for which "hard instances" can be efficiently sampled. More formally, a subset membership problem $\mathcal{I}$ specifies a collection $\{I_n\}_{n\in\mathsf{N}}$ such that for every n, I_n is a probability distribution over *problem instance descriptions* Λ. A problem instance description defines a set and a hard language for that set. Formally, each instance description Λ specifies the following:

1. Finite, non-empty sets $X_n, L_n \subseteq \{0,1\}^{\mathrm{poly}(n)}$ such that $L_n \subset X_n$, and distributions $D(L_n)$ over L_n and $D(X_n \backslash L_n)$ over $X_n \backslash L_n$.
2. A witness set $W_n \subseteq \{0,1\}^{\mathrm{poly}(n)}$ and an NP-relation $R_n \subseteq X_n \times W_n$. R_n and W_n must have the property that $x \in L_n$ if and only if there exists $w \in W_n$ such that $(x,w) \in R_n$.

We are interested in subset membership problems $\mathcal{I}$ which are efficiently samplable. That is, the following algorithms must exist:

1. *Problem instance samplability:* a probabilistic polynomial-time algorithm that upon input 1^n, samples an instance $\Lambda = (X_n, D(X_n\backslash L_n), L_n, D(L_n), W_n, R_n)$ from I_n.
2. *Instance member samplability:* a probabilistic polynomial-time algorithm that upon input 1^n and an instance $(X_n, D(X_n\backslash L_n), L_n, D(L_n), W_n, R_n)$, samples $x \in L_n$ according to distribution $D(L_n)$, together with a witness w for which $(x,w) \in R_n$.
3. *Instance non-member samplability:* a probabilistic polynomial-time algorithm that upon input 1^n and an instance $(X_n, D(X_n\backslash L_n), L_n, D(L_n), W_n, R_n)$, samples $x \in X_n\backslash L_n$ according to distribution $D(X_n\backslash L_n)$.

We are now ready to define hard subset membership problems:

Definition 3.1 (hard subset membership problems): *Let $V(L_n)$ be the following random variable: Choose a problem instance Λ according to I_n, a value $x \in L_n$ according to $D(L_n)$ (as specified in Λ), and then output (Λ, x). Similarly, define $V(X_n\backslash L_n)$ as follows: Choose a problem instance Λ according to I_n, a value $x \in X_n\backslash L_n$ according to $D(X_n\backslash L_n)$ (as specified in Λ) and then output (Λ, x). Then, we say that a subset membership problem $\mathcal{I}$ is* hard *if*

$$\left\{V(L_n)\right\}_{n\in\mathsf{N}} \stackrel{\mathsf{c}}{\equiv} \left\{V(X_n\backslash L_n)\right\}_{n\in\mathsf{N}}.$$

In other words, $\mathcal{I}$ is hard if random members of L_n cannot be distinguished from random non-members. In order to simplify notation, from here on we drop the subscript of n from all sets. However, all mention of sets X and L etc., should be understood as having been sampled according to the security parameter n.

Smooth projective hash functions. Loosely speaking a smooth projective hash function is a function with two keys. The first key maps the entire set X to some set G. The second key (called the projection key) is such that it can be used to correctly compute the mapping of L to G. However, it gives no information about the mapping of $X \setminus L$ to G. In fact, given the projection key, the distribution over the mapping of $X \setminus L$ to G is statistically close to uniform (or "smooth"). We now present the formal definition.

Let X and G be finite, non-empty sets and let $\mathcal{H} = \{H_k\}_{k \in K}$ be a collection of hash functions from X to G. We call K the key space of the family. Now, let L be a non-empty, proper subset of X (i.e., L is a language). Then, we define a *key projection* function $\alpha : K \times X \to S$, where S is the space of key projections. Informally, the above system defines a projective hash system if for $x \in L$, the projection key $s_x = \alpha(k, x)$ uniquely determines $H_k(x)$. (Ignoring issues of efficiency, this means that $H_k(x)$ can be computed given only s_x and $x \in L$.) We stress that the projection key $s_x = \alpha(k, x)$ is only guaranteed to determine $H_k(x)$ for $x \in L$, and nothing is guaranteed for $x' \neq x$. Formally,

Definition 3.2 (projective hash functions): *The family $(\mathcal{H}, K, X, L, G, S, \alpha)$ is a* projective hash family *if for all $k \in K$ and $x \in L$, it holds that the value of $H_k(x)$ is* uniquely determined *by $\alpha(k, x)$ and x.*

Of course, projective hash functions can always be defined by taking $\alpha(\cdot, \cdot)$ to be the identity function. However, we will be interested in *smooth* projective hash functions which have the property that for every $x \notin L$, the projection key $s_x = \alpha(k, x)$ reveals (almost) nothing about $H_k(x)$. More exactly, for every $x \notin L$, the distribution of $H_k(x)$ given $\alpha(k, x)$ should be statistically close to uniform. Formally,

Definition 3.3 (smooth projective hash functions [13]): *Let $(\mathcal{H}, K, X, L, G, S, \alpha)$ be a projective hash family. Then, for every $x \in X \setminus L$ define the random variable $V(x, \alpha(k), H_k(x))$ by choosing $k \in_R K$ and output $(x, \alpha(k, x), H_k(x))$. Similarly, define $V(x, \alpha(k, x), g)$ as follows: choose $k \in_R K$, $g \in_R G$ and output $(x, \alpha(k, x), g)$. Then, the projective hash family $(\mathcal{H}, K, X, L, G, S, \alpha)$ is* smooth *if for every $x \in X \setminus L$:*

$$\left\{V(x, \alpha(k), H_k(x))\right\}_{n \in \mathsf{N}} \stackrel{s}{\equiv} \left\{V(x, \alpha(k), g)\right\}_{n \in \mathsf{N}}.$$

Efficient smooth projective hash functions. We say that a smooth projective hash family is efficient if the following algorithms exist:

1. *Key sampling:* a probabilistic polynomial-time algorithm that upon input 1^n samples $k \in K$ uniformly at random.

2. *Projection computation:* a deterministic polynomial-time algorithm that upon input 1^n, $k \in K$ and $x \in X$ outputs $s = \alpha(k, x)$.

3. *Efficient hashing from key:* a deterministic polynomial-time algorithm that upon input 1^n, $k \in K$ and $x \in X$, outputs $H_k(x)$.

4. *Efficient hashing from projection key and witness:* a deterministic polynomial-time algorithm that upon input 1^n, $x \in L$ with a witness w such that $(x, w) \in R$, and $\alpha(k, x)$ (for some $k \in K$), computes $H_k(x)$.

We note an interesting and important property of such hash functions. For $x \in L$, it is possible to compute $H_k(x)$ in two ways: either by knowing the key k (as in item 3 above) or by knowing the projection s_x of the key, and a witness for x (as in item 4 above). This property plays a central role in our password-based protocol.

Another interesting property formalized by Gennaro and Lindell in [19] is that these are the only ways to compute $H_k(x)$. Specifically, for $x \in_R D(L)$ (where an appropriate witness w is not known), the value $H_k(x)$ is *computationally* indistinguishable from random, given the projection s_x.

Since we use smooth projective hashing in our password protocol, it is necessary to prove the above statement even when the adversary sees many tuples $(x, s_x, H_k(x))$ with $x \in_R D(L)$. Let M be a (non-uniform) polynomial-time oracle machine. Define the following two experiments.

$\mathsf{Expt\text{-}Hash}(M)$: An instance $\Lambda = (X, D(X \setminus L), L, D(L), W, R)$ of a hard subset membership problem is chosen from I_n. Then, the machine M is given access to two oracles: Ω_L and $\mathsf{Hash}(\cdot)$. The Ω_L oracle receives an empty input and returns $x \in L$ chosen according to the distribution $D(L)$. The Hash oracle receives an input x. It first checks that x was previously output by the Ω_L oracle. If no, then it returns nothing. Otherwise, it chooses a key $k \in_R K$ and returns the pair $(\alpha(k, x), H_k(x))$. We stress that the Hash oracle *only* answers for inputs x that were generated by Ω_L. The output of the experiment is whatever machine M outputs.

$\mathsf{Expt\text{-}Unif}(M)$: This experiment is defined exactly as above except that the Hash oracle is replaced by the following Unif oracle. On input x, Unif first checks that x was previously output by the Ω_L oracle. If no, it returns nothing. Otherwise, it chooses a key $k \in_R K$ and a random element $g \in_R G$, and returns the pair $(\alpha(k, x), g)$. As above, the output of the experiment is whatever M outputs.

In [19] it is proven that no efficient M can distinguish between the experiments. In other words, when $x \in_R D(L)$, the value $H_k(x)$ is pseudorandom in G, even given $\alpha(k, x)$. This lemma is used a number of times in the proof of our password protocol.

Lemma 3.4 *Assume that $\mathcal{I}$ is a hard subset membership problem. Then, for every (non-uniform) polynomial-time oracle machine M it holds that,*

$$\Big|\Pr[\mathsf{Expt\text{-}Hash}(M) = 1] - \Pr[\mathsf{Expt\text{-}Unif}(M) = 1]\Big| < \mathsf{negl}(n).$$

Hard partitioned subset membership problems. We now consider a variant of hard subset membership problems, where the set X can be *partitioned* into disjoint subsets of hard problems. That is, assume that the set X contains pairs of the form (i, x), where $i \in \{1, \ldots, \ell\}$ is an index. We denote by $X(i)$ the subset of pairs in X of the form (i, x). Furthermore, we denote by $L(i)$ the subset of pairs in the language L of the form (i, x). (We also associate sampling distributions $D(L(i))$ and $D(X(i) \setminus L(i))$ to each partition.) Then, such a problem constitutes a hard $\mathsf{partitioned}$ subset membership problem if for *every* i, it is hard to distinguish $x \in_R D(L(i))$ from $x \in_R D(X(i) \setminus L(i))$. (In the notation of Definition 3.1, we require that for every i, the ensembles $\{V(L(i))\}$

and $\{V(X(i)\backslash L(i))\}$ are computationally indistinguishable.) We stress that the definition of smooth projective hashing is unchanged when considered in the context of hard partitioned subset problems. That is, the smoothness is required to hold with respect to the entire sets X and L, and not with respect to individual partitions.

Lemma 3.4 also holds for hard partitioned subset membership problems (see [19]. Specifically, the definitions of the oracles in the experiments are modified as follows. The Ω_L oracle is modified so that instead of receiving the empty input, it is queried with an index i, and returns $x \in_R D(L(i))$. Likewise, $\Omega_{X \backslash L}$ receives an index i and returns an element $x \in_R D(X(i)\backslash L(i))$. Notice that in this scenario, the distinguishing machine M is given some control over the choice of x. Specifically, M can choose the index i that determines from which partition an element x is sampled.

Corollary 3.5 *Assume that $\mathcal{I}$ is a family of hard* partitioned *subset membership problem. Then, for every (non-uniform) polynomial-time oracle machine M*

$$|\Pr[\mathsf{Expt\text{-}Hash}(M) = 1] - \Pr[\mathsf{Expt\text{-}Unif}(M) = 1]| < \mathsf{negl}(n).$$

4 The Protocol

Our protocol uses a chosen-ciphertext secure public-key labeled encryption scheme $\mathcal{E}$. The common reference string for the protocol is a public key PK for $\mathcal{E}$.

We then use a family of smooth projective functions $\mathcal{H} = \{H_k\}$ such that for every k in the key space K, $H_k : C_{PK} \times M \to \{0,1\}^{2n}$, where M is the message space, C_{PK} is an efficiently recognizable superset of the ciphertext space. Notice that we are assuming that the projective hash function outputs $2n$-bit strings[5].

Finally, we assume that there is a mechanism that enables the parties to differentiate between different concurrent executions and to identify who they are interacting with. This can easily be implemented by having P_i choose a sufficiently long random string r and send the pair (i, r) to P_j along with its first message. P_i and P_j will then include r in any future messages of the protocol. We stress that the security of the protocol does not rest on the fact that these values are not modified by the adversary. Rather, this just ensures correct communication for protocols that are not under attack. The protocol appears in Figure 2.

Intuitive Security Proof. First notice that both A and B can compute the session key as instructed. Specifically, A can compute $H_k(c, w, A \circ B)$ because it has the projection key s and the witness (coins) for c. Furthermore, it can compute $H_{k'}(c', w, c \circ s)$ because it has the key k' (and therefore does not need the witness for c'). Likewise, B can also correctly compute both the hash values (and thus the session key). Second, when both parties A and B see the same messages (c, s, c', s', t) the session keys that they compute are the same. This is because the same hash value is obtained when using the hash keys (k and k') and when using the projection keys (s and s'). This implies that the correctness property holds for the protocol.

We now proceed to motivate why the adversary cannot distinguish a session key from a random key with probability greater that $Q_{\text{send}}/|\mathcal{D}|$, where Q_{send} equals the number of Send oracle calls made by the adversary to different protocol instances and $\mathcal{D}$ is the password dictionary. In order to see this, notice that if A, for example, receives c' that is not an encryption to w with label

[5]We note that the constructions in [19] output values in a large algebraic group G. It is a standard application of randomness extraction (e.g. via universal hashing) to map such elements into $2n$-bit strings, assuming the group G is large enough.

G-PaKE

- **Common reference string:** The public key PK for a chosen-ciphertext secure encryption scheme $\mathcal{E}$. A description of a smooth projective hashing family H_k over the set X of ciphertext/password pairs (c, w). The NP language L is composed of the tuples (c, m, ℓ) where $c = \mathcal{E}_{PK}(m, \ell)$ i.e. c is an encryption of m with label ℓ under PK. A message authentication code MAC.
- **Common input:** a shared (low-entropy) password w.
- **The protocol:**
  1. Party A computes an encryption $c = \mathcal{E}_{PK}(w, A \circ B)$ and sends it to party B.
  2. Party B chooses a key k for the smooth projective hash function (for the language L described above), and computes its projection $s = \alpha(k, c)$. Also B computes the projective hash over $(c, w, A \circ B)$, i.e. $sk_B = H_k(c, w, A \circ B)$.

     Finally B computes another encryption of the password with label $c \circ s$ i.e., $c' = \mathcal{E}_{PK}(w, c \circ s)$.

     B sends s, c' to party A.
  3. Party A chooses another key k' for the smooth projective hash function (for the language L described above), and computes its projection $s' = \alpha(k', c')$. Also A computes the projective hash over $(c', w, c \circ s)$, i.e. $sk_A = H_{k'}(c', w, c \circ s)$.

     A also computes sk_B using the projection s and its knowledge of a witness for the fact that c is an encryption to the string w with label $A \circ B$ (it knows a witness because it generated c).

     Set $t = MAC_{sk_B^{(1)}}(c, s, c', s') \oplus sk_A^{(1)}$.

     A sends s', t to party B.
- **Session Key Definition:**
  1. Party B computes sk_A using the projection s' and its knowledge of a witness for the fact that c' is an encryption to the password w with label $c \circ s$ (it knows a witness because it generated c')

     It tests if $t = MAC_{sk_B^{(1)}}(c, s, c', s') \oplus sk_A^{(1)}$. If the test fails it outputs an error message, otherwise it outputs $sk = sk_A^{(2)} \oplus sk_B^{(2)}$.
  2. Party A outputs $sk = sk_A^{(2)} \oplus sk_B^{(2)}$.

Session-Identifier Definition: Both parties take the series of messages (c, s, c', s') to be their session identifiers.

Figure 2: Improved Password-Based Session-Key Exchange

$c \circ s$ under PK, then A's component of the session key sk_A will be statistically close to uniform. This is because A computes $H_{k'}(c', w, c \circ s)$ for $c' \notin \mathcal{E}_{PK}(w, c \circ s)$ i.e. on an input outside the language. Therefore, by the definition of smooth projective hashing, $\{c', w, \alpha(k, c'), H_k(c', w, c \circ s)\}$ is statistically close to $\{c', w, \alpha(k, c'), r\}$, where r is a random $2n$-bit string.

The same argument holds if B receives c that is not a encryption of w with label $A \circ B$. It therefore follows that if the adversary is to distinguish the session key from a random element, it must hand the parties encryptions of the valid messages (and in particular containing the correct passwords). One way for the adversary to do this is to copy (valid) commitments that are sent by the honest parties in the protocol executions. However, in this case, the adversary does not know the random coins used in generating the commitment, and once again the result of the projective hash function is a pseudorandom $2n$-bit string (see Lemma 3.4). This means that the only option left to the adversary is to come up with valid commitments that were not previously sent by honest parties. However, by the non-malleability of the encryption scheme, the adversary cannot succeed in doing this with probability non-negligibly greater than just a priori guessing the password. Thus, its success probability is limited to $Q_{\text{send}}/|\mathcal{D}| + \mathsf{negl}(n)$.

This intuitive explanation of the security of the protocol is not complete. Indeed it does not address the use of message authentication codes in the protocol. The MAC is needed to prevent further malleability attacks. Indeed the combination of projective hash function and semantically secure symmetric encryption alone could be malleable, and by recycling messages from previous executions the adversary could gain some knowledge about a session key. For example, it is possible that for some smooth projective hash family it holds that for every k, $H_{2k}(x) = 2H_k(x)$. If this were the case, in the basic protocol described in Figure 1 an adversary could cause two instances to accept with *different* session identifiers and *related* session-keys. By requesting a Reveal for one of the instances, it could then distinguish the other instance's session-key from random, in contradiction to the security requirements.

Notice that the projection s is protected by malleability attacks because is incorporated as a label in the ciphertext c'. The surprising thing is that at this point the value sk_B is already pseudo-random to the eyes of the adversary, and known to both parties A and B. The most intuitive thing would be to use it to MAC the other projection s'.

But if A were to send $t = MAC_{sk_B}(c, s, c', s')$ the adversary could perform the following off-line attack. The adversary would start a session with A pretending to be B and obtain the commitment c. Next, the adversary $\mathcal{A}$ chooses k and returns $s = \alpha(k, c)$ to A together with an incorrect encryption c'. The response from A is s' and t computed as above with $sk_B = H_k(c, w, A \circ B)$. Now the adversary can traverse the entire dictionary $\mathcal{D}$ and for all possible w's compute $sk_B = H_k(c, w, A \circ B)$ (it can do this because it knows k and so can compute $H_k(c, w)$ without a witness for c). The right password is the one for which sk_B verifies the above MAC t.

The final solution is then to "mask" the MAC, using sk_A which is not known to the adversary, and cannot even be computed off-line by traversing the dictionary (because the adversary does not know the coins used to produce c'). If the adversary modifies s' and makes B accept then it must produce a MAC forgery with key sk_B. Of course this is just an intuition and the proof in the Appendix works out all the details[6].

Theorem 4.1 *Assume that $\mathcal{E}$ is a public-key encryption secure against adaptive chosen ciphertext attack, MAC is a secure message authentication code and $\mathcal{H}$ is a family of smooth projective hash functions. Then, Protocol* G-PaKE *in Figure* 2 *is a secure password-based session-key generation protocol.*

5 Extension and Conclusions

The reader is referred to [19] to see examples of efficient chosen-ciphertext secure encryption schemes that admit the type of projective hash functions needed in this protocol. They are based on the encryption schemes proposed by Cramer and Shoup in [12, 13], and can be based on the DDH, Quadratic Residuosity and N-Residuosity Assumptions. Labels are easily incorporated into these schemes, without impacting their overall efficiency. See Appendix C for a discussion of this point.

For simplicity we have presented the protocol using chosen-ciphertext secure encryption. It is possible using techniques used in [19] to prove the protocol assuming that $\mathcal{E}$ is non-malleable commitment scheme, which admits a projective hash function. The protocol needs to be modified

[6]One final technicality: in in order to protect the semantic security of the final session key we use the fist half of sk_A and sk_B to perform the MAC test during the protocol and the second half to compute the actual session key.

and the proof is more complicated. However in practice it does not make much of a difference, as the only known efficient implementations of such commitment schemes are the ones mentioned above (i.e. based on chosen-ciphertext secure encryption and described by [19]).

Canetti et al. extend the KOY and GL protocol to the Universal Composability framework in [9]. Their protocol also uses one-time signatures to prevent malleability attacks and our modification is applicable to their protocol as well.

Conclusions. We have shown an improvement of the KOY and GL protocols, which does not require one-time signatures. Our protocol works in the common reference string and its proof does not require idealized assumptions such as the random oracle. For typical security parameters our protocol saves about 12 Kbytes of bandwidth, thus bringing provable security in the realm of password-authenticated key exchange one step closer to practical.

References

[1] M. Abe, R. Gennaro and K. Kurosawa. . Tag-KEM/DEM: A New Framework for Hybrid Encryption. To appear in the *Journal of Cryptology.* Available at `http://eprint.iacr.org/2005/027`.

[2] M. Bellare, D. Pointcheval and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Eurocrypt 2000*, Springer-Verlag (LNCS 1807), pp.139–155, 2000.

[3] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *CRYPTO'93*, Springer-Verlag (LNCS 773), pp.232–249, 1994.

[4] S.M. Bellovin and M. Merritt. Encrypted Key Exchange: Password Based Protocols Secure Against Dictionary Attacks. In *Proceedings 1992 IEEE Symposium on Research in Security and Privacy*, pp.72–84. IEEE Computer Society, 1992.

[5] S.M. Bellovin and M. Merritt. Augmented Encrypted Key Exchange: A Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise. In *Proceedings of the 1st ACM Conference on Computer and Communication Security*, pp.244–250, 1993.

[6] D. Boneh and X. Boyen. *Short Signatures Without Random Oracles.* EUROCRYPT'04, LNCS 3027, pp.56-73

[7] V. Boyko, P. MacKenzie and S. Patel. Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In *Eurocrypt 2000*, Springer-Verlag (LNCS 1807), pp.156–171, 2000.

[8] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. *Journal of the ACM*, 51(4):557–594, 2004.

[9] R. Canetti, S. Halevi, J. Katz, Y. Lindell and P. MacKenzie. *Universally Composable Password-Based Key Exchange.* EUROCRYPT 2005, LNCS Vol.3494, pp.404–421, 2005.

[10] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Eurocrypt 2001*, Springer-Verlag (LNCS 2045), pp.453–474, 2001.

[11] D. Catalano, M. Di Raimondo, D. Fiore and R. Gennaro. *Some Theoretical and Experimental Results about Off-Line/On-Line Signatures.* Manuscript. Available from the authors.

[12] R. Cramer and V. Shoup. A Practical Public-Key Cryptosystem Secure Against Adaptive Chosen Ciphertexts Attacks. In *CRYPTO'98*, Springer-Verlag (LNCS 1462), pp.13–25, 1998. Full version in [14].

[13] R. Cramer and V. Shoup. Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. In *Eurocrypt 2002*, Springer-Verlag (LNCS 2332), pp.45–64, 2002. Full version in [14].

[14] R. Cramer and V. Shoup. Design and Analysis of Practical Public-Key Encryption Schemes Secure Against Adaptive Chosen Ciphertext Attack. *SIAM Journal of Computing*, 33:167-226, 2003.

[15] R. Cramer and V. Shoup. *Signature schemes based on the strong RSA assumption.* ACM Trans. Inf. Syst. Secur. 3(3): 161-185 (2000)

[16] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Trans. on Inf. Theory*, IT-22, pp.644–654, Nov. 1976.

[17] D. Dolev, C. Dwork and M. Naor. Non-Malleable Cryptography. *SIAM Journal of Computing*, 30(2):391–437, 2000.

[18] R. Gennaro, S. Halevi and T. Rabin. Secure Hash-and-Sign Signatures Without the Random Oracle. EUROCRYPT 1999, LNCS Vol.1592, pp.123-139, Springer 1999.

[19] R. Gennaro and Y. Lindell. *A framework for password-based authenticated key exchange.* ACM Transactions on Information and System Security (TISSEC), 9(2):181–234, ACM Press, May 2006.

[20] O. Goldreich and Y. Lindell. Session Key Generation using Human Passwords Only. In *CRYPTO 2001*, Springer-Verlag (LNCS 2139), pp.408–432, 2001.

[21] S. Goldwasser, S. Micali and R.L. Rivest, *A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks.* SICOMP, 17(2):281–308, 1988.

[22] S. Halevi and H. Krawczyk. Public-Key Cryptography and Password Protocols. *ACM Transactions on Information and System Security (TISSEC)*, 2(3):230–268, 1999.

[23] D.P. Jablon. Strong Password-Only Authenticated Key Exchange. *SIGCOMM Computer Communication Review*, 26(5):5–26, 1996.

[24] J. Katz, R. Ostrovsky and M. Yung. Practical Password-Authenticated Key Exchange Provably Secure under Standard Assumptions. In *Eurocrypt 2001*, Springer-Verlag (LNCS 2045), pp.475–494, 2001.

[25] K. Kurosawa and K. Schmidt-Samoa. *New Online/Offline Signature Schemes Without Random Oracles.* Public Key Cryptography 2006, LNCS Vol.3958, pp.330-346, Springer.

[26] L. Lamport. *Constructing digital signatures from a one-way function.* Technical Report CSL-98, SRI International, Oct. 1979.

[27] S. Lucks. Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys. In *Proceedings of the Workshop on Security Protocols*, Springer-Verlag (LNCS 1361), pp.79–90, Ecole Normale Superieure, 1997.

[28] S. Patel. Number Theoretic Attacks on Secure Password Schemes. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pp.236–247, 1997.

[29] M. Steiner, G. Tsudik and M. Waidner. Refinement and Extension of Encrypted Key Exchange. *ACM SIGOPS Oper. Syst. Rev.*, 29(3):22–30, 1995.

[30] T. Wu. The Secure Remote Password Protocol. In *1998 Internet Society Symposium on Network and Distributed System Security*, pp.97–111, 1998.

A Password-Based Authenticated Key Exchange

In this section, we briefly recall the formal security model for password key exchange protocols as presented in [2] (which is in turn based on [3]). For more details and motivation, we refer the reader to [2]. A protocol for password key exchange assumes that there is a set of *principals* which are the clients and the servers who will engage in the protocol. Each client stores its own private password w, while each server stores all the passwords of the clients who have access to that server.[7] The passwords for all pairs of parties are uniformly (and independently) chosen from a fixed dictionary $\mathcal{D}$.[8] Each principal may start various executions of the protocol, with different partners. Thus we denote with Π_i^l the l^{th} instance that user P_i runs. The adversary is given oracle access to these instances and may also control some of the instances itself. We remark that unlike the standard notion of an "oracle", in this model instances maintain state which is updated as the protocol progresses. In particular the state of an instance $\Pi_i^{l_i}$ includes the following variables (initialized as *null*):

- $\mathsf{sid}_i^{l_i}$: the *session identifier* of this particular instance;
- $\mathsf{pid}_i^{l_i}$: the *partner identifier* which is the name of the principal with whom $\Pi_i^{l_i}$ believes it is interacting (we note that $\mathsf{pid}_i^{l_i}$ can never equal i);
- $\mathsf{acc}_i^{l_i}$: a boolean variable denoting whether $\Pi_i^{l_i}$ accepts or rejects at the end of the execution.

Partnering. We say that two instances $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ are *partnered* if the following properties hold: (1) $\mathsf{pid}_i^{l_i} = j$ and $\mathsf{pid}_j^{l_j} = i$; and (2) $\mathsf{sid}_i^{l_i} = \mathsf{sid}_j^{l_j} \neq null$. The notion of partnering is important for defining security, as we will see.

The adversarial model. The adversary is given total control of the external network (i.e., the network connecting clients to servers). In particular we assume that the adversary has the ability to not only listen to the messages exchanged by players, but also to interject messages of its choice and modify or delete messages sent by the parties.[9] The above-described adversarial power is modeled by giving the adversary oracle access to the instances of the protocol that are run by the principals. Notice that this means that the parties actually only communicate through the adversary. The oracles provided to the adversary are as follows:

[7] We actually make no use of the asymmetry between clients and servers. Indeed, the protocol is stated for a general scenario where arbitrary pairs of parties share secret passwords.

[8] This uniformity requirement is made for simplicity and can be easily removed by adjusting the security of an individual password to be the min-entropy of the distribution, instead of $1/|\mathcal{D}|$.

[9] In principle, the adversary should also be given control over a subset of the oracles, modeling the case of an "inside attacker". In the password-only setting, this actually makes no difference because the adversary can internally simulate these oracles by itself. We therefore leave out this requirement in our formal definition and analysis.

- $\mathsf{Execute}(i, l_i, j, l_j)$: When this oracle is called, a complete protocol execution between instances $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ takes place. The oracle-output is the protocol transcript (i.e., the complete series of messages exchanged by the instances throughout the execution). These oracle calls reflect the adversary's ability to passively eavesdrop on protocol executions. As we shall see, the adversary should learn nothing from such oracle calls.

- $\mathsf{Send}(i, l_i, M)$: This call sends the message M to the instance $\Pi_i^{l_i}$. The output of the oracle is whatever message the instance $\Pi_i^{l_i}$ would send after receiving the message M (given its current state). This oracle allows the adversary to carry out an active man-in-the-middle attack on the protocol executions.

- $\mathsf{Reveal}(i, l_i)$: This call outputs the secret key $sk_i^{l_i}$ which resulted from instance $\Pi_i^{l_i}$. This oracle allows the adversary to learn session keys from previous and concurrent executions, modeling improper exposure of past session keys and insuring independence of different session keys in different executions.

- $\mathsf{Test}(i, l_i)$: This call is needed for the definition of security and does not model any real adversarial ability. The adversary is only allowed to query it once, and the output is either the private session key of $\Pi_i^{l_i}$, denoted $sk_i^{l_i}$, or a random key sk that is chosen independently of the protocol executions (each case happens with probability 1/2). The adversary's aim is to distinguish these two cases.

The security of key exchange protocols is composed of two components: correctness and privacy. We begin by stating the correctness requirement:

Non-triviality: If two instances $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ that hold each other's partner identifier communicate without adversarial interference (as in an Execute call), then $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ are partnered and they both accept.

Correctness: If two *partnered* instances $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ accept (i.e., $\mathsf{acc}_i^{l_i} = \mathsf{acc}_j^{l_j} = 1$), then they must both conclude with the same session key (i.e., $sk_i^{l_i} = sk_j^{l_j}$).

Privacy: We now define what it means for a protocol to be private. Intuitively, a protocol achieves privacy if the adversary cannot distinguish real session keys from random ones. (This then implies that the parties can use their generated session keys in order to establish secure channels; see [10] for more discussion on this issue.) Formally, we say that the adversary succeeds if it correctly guesses the bit determining whether it received the real session key or a random session key in the Test oracle query. Of course, the adversary can always correctly guess the bit in a $\mathsf{Test}(i, l_i)$ query if it queried $\mathsf{Reveal}(i, l_i)$ or $\mathsf{Reveal}(j, l_j)$ when $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ are partnered. Therefore, $\mathcal{A}$ is only said to have succeeded if these oracles were not queried. Now, the adversary's advantage is formally defined by:

$$\mathsf{Adv}(\mathcal{A}) = |2 \cdot \text{Prob}[\mathcal{A} \text{ succeeds }] - 1| .$$

We reiterate that an adversary is only considered to have succeeded if it correctly guesses the bit used by the $\mathsf{Test}(i, l_i)$ oracle and it did *not* query $\mathsf{Reveal}(i, l_i)$ or $\mathsf{Reveal}(j, l_j)$ when $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ are partnered.

As we have mentioned, when low entropy passwords are used, the adversary can always gain a non-negligible advantage (by just guessing passwords in an on-line guessing attack). A protocol is therefore called private if it is limited to such an on-line guessing attack. Notice that in $\mathsf{Execute}$ oracle calls, the adversary is passive (and thus is not carrying out any on-line attack). Therefore, we only count password guesses when the adversary queries the Send oracle. Furthermore, we also only count *one* Send query for each protocol instance (i.e., if the adversary sends two Send oracle queries to a single protocol instance, it should still only count as a single password guess). Formally, a protocol is said to be $\mathsf{private}$ if the advantage of the adversary is at most negligibly more than $Q_{\text{send}}/|\mathcal{D}|$, where Q_{send} is the number of Send oracle queries made by the adversary to different protocol instances and $\mathcal{D}$ is the dictionary. In summary,

Definition A.1 (password-based key exchange): *A password-based authenticated key exchange protocol is said to be* secure *if for every dictionary* $\mathcal{D}$ *and every (non-uniform) polynomial-time adversary* $\mathcal{A}$ *that makes at most* Q_{send} *queries of type* Send *to different protocol instances,*

$$\mathsf{Adv}(\mathcal{A}) < \frac{Q_{\text{send}}}{|\mathcal{D}|} + \mathsf{negl}(n).$$

Furthermore, the probability that the correctness requirement is violated is at most negligible in the security parameter n*.*

We note that the bound of $Q_{\text{send}}/|\mathcal{D}|$ for $\mathcal{A}$'s advantage is actually optimal. That is, one can always construct an adversary who obtains this exact advantage by guessing a password, and then playing $\Pi_i^{l_i}$'s role in a protocol execution with $\Pi_j^{l_j}$, using the guessed password. It is easily verified that $\mathcal{A}$'s advantage in this attack is exactly $Q_{\text{send}}/|\mathcal{D}|$.

B Proof of Security

Our proof of security significantly differs from the proof of [24, 19] since it has to work without using digital signatures. It is however structured in the same way, i.e. through a series of hybrid arguments.

We begin by proving the *correctness* requirement. Notice that the session identifiers are defined to be (c, s, c', s', t) as seen by the parties in the execution. Since the session key is fully defined by these values, it follows that parties with the same identifiers must always have the same session key. Thus, correctness holds.

We now proceed to prove the *privacy* requirement through a series of hybrid experiment. In the following we refer to sk_A and sk_B as the *master keys* computed by the players. In each new hybrid, we modify the way these master keys (and thus the session key) are chosen for certain protocol instances. We begin by choosing random master keys for protocol instances for which the $\mathsf{Execute}$ oracle is called. We then proceed to choose random master keys for instances that receive Send oracle calls. These instances are gradually changed over six hybrid experiments, depending on specific sub-cases. In the last hybrid experiment, all the master keys are (almost) uniform $2n$-bit strings. Thus, the adversary clearly cannot distinguish them from random. We denote these hybrid experiments by $H_0, \ldots, H_6$ and by $\mathsf{Adv}(\mathcal{A}, H_i)$ the advantage of $\mathcal{A}$ when participating in experiment H_i. The real adversarial attack is denoted by H_0 and in this experiment all the oracles are as defined in the protocol. Thus $\mathsf{Adv}(\mathcal{A}, H_0) = \mathsf{Adv}(\mathcal{A})$ and we wish to bound this advantage by $Q_{\text{send}}/|\mathcal{D}| + \mathsf{negl}(n)$, where Q_{send} denotes the number of Send queries that $\mathcal{A}$ makes to different protocol instances during its attack.

Hybrid experiment H_1: In this experiment, the Execute oracle is modified so that the master keys of protocol instances for which Execute is called are all chosen uniformly as random $2n$-bit strings, rather than being computed using the projective hash. We prove that the view of the adversary in H_1 is indistinguishable from its view in a real execution. Intuitively, this is due to Lemma 3.4 which states that without knowing k or the randomness used to generate an encryption, the distribution $\{s, w, i \circ j, H_k(c, w, i \circ j)\}$ is computationally indistinguishable from $\{s, w, i \circ j, \rho\}$, where $\rho \in_R \{0,1\}^{2n}$. We now formally prove that modifying the oracles in this way can make at most a negligible difference. That is,

Claim B.1 *For every non-uniform polynomial-time adversary $\mathcal{A}$,*

$$|\mathsf{Adv}(\mathcal{A}, H_1) - \mathsf{Adv}(\mathcal{A}, H_0)| < \mathsf{negl}(n).$$

Proof: The proof of this claim works by showing how any advantage that $\mathcal{A}$ has in distinguishing H_1 from H_0 can be used to construct a polynomial-time machine that distinguishes between Expt-Hash and Expt-Unif as defined in Section 3. We recall the definition of these two experiments, specified to our setting:

Expt-Hash(D): A public key PK is chosen and given to the machine D. D can then query two oracles: Encrypt$(\cdot,\cdot)$ and Hash$(\cdot,\cdot,\cdot)$. The Encrypt oracle receives a message m and a label ℓ and returns an encryption $c \in_R \mathcal{E}_{PK}(m, c, \ell)$ (generated using random coins r). The Hash oracle receives an input c, ℓ where c was output by the Encrypt oracle on input m, ℓ. It chooses a key $k \in_R K$ and computes the projection $s = \alpha(k, c)$. It then outputs $(s, H_k(c, m, \ell))$. We stress that the Hash oracle *only* answers on inputs c, ℓ where c was generated by Encrypt on input m, ℓ. The output of the experiment is whatever the machine D outputs.

Expt-Unif(D): This experiment is defined exactly as above except that the Hash oracle works as follows. Upon input c, ℓ, where c was generated by Encrypt on input m, ℓ, it chooses a key $k \in_R K$ and computes the projection $s = \alpha(k, c)$. It then chooses a random string $\rho \in_R \{0,1\}^n$ and outputs (s, ρ).

In Corollary 3.5, we showed that for every (non-uniform) probabilistic polynomial-time machine D,

$$|\Pr[\mathsf{Expt\text{-}Hash}(D) = 1] - \Pr[\mathsf{Expt\text{-}Unif}(D) = 1]| < \mathsf{negl}(n).$$

Indeed we can see this as a family of partitioned hard subset membership problems where X is partitioned by the message/label pair m, ℓ. In this context, the Encrypt oracle plays the role of Ω_L.

We now show that $|\Pr[\mathsf{Expt\text{-}Hash} = 1] - \Pr[\mathsf{Expt\text{-}Unif} = 1]|$ upper bounds the difference between an adversary $\mathcal{A}$'s advantage in H_0 and H_1. Loosely speaking, this is shown as follows. For all instances involved in Execute calls, the Encrypt and Hash oracles from the Expt-Hash and Expt-Unif experiments are used to obtain the commitments and projection keys, and to compute the master key mk. Then, if Expt-Hash is being used, the result is identical to the protocol specification, and therefore H_0. In contrast, if Expt-Unif is being used, then the master keys are chosen at random for all Execute calls (and everything else remains the same as in the protocol specification). The result is therefore exactly H_1. We conclude that any distinguisher for H_0 and H_1 can be used to distinguish between Expt-Hash and Expt-Unif, with the same probability.

Formally, let $\mathcal{A}$ be an adversary interacting in either H_0 or H_1. Then, we construct a machine D for the Expt-Hash and Expt-Unif experiments as follows. D receives a randomly chosen public key PK, and chooses random passwords $w_1, w_2, \ldots$ for all the pairs of parties. Then, D emulates the H_0/H_1 experiment exactly as prescribed. The only difference relates to the emulation of the Execute oracles. Rather than computing c, s, c', s', t as prescribed by the protocol, D queries its Encrypt oracle with $(w, i \circ j)$ and $(w, c \circ s)$ obtaining c and c' respectively. Then, the Hash oracle is queried with $(c, i \circ j)$ and $(c', c \circ s)$, returning (s, h) and (s', h') respectively. The projection keys are included in the transcript and the master keys for these instances are chosen as $sk = h + h'$. Everything else in the emulation remains the same. At the conclusion of the emulation, D outputs whatever $\mathcal{A}$ does.

Now, if D interacts in Expt-Hash, then the emulation carried out by D is exactly that of H_0. In contrast, if D interacts in Expt-Unif, then the master keys of protocol instances for which Execute is invoked are all random strings. Thus, the emulation by D is exactly that of H_1. We conclude that D distinguishes between Expt-Hash and Expt-Unif with probability exactly $|\mathsf{Adv}(\mathcal{A}, H_0) - \mathsf{Adv}(\mathcal{A}, H_1)|$. Thus, by Corollary 3.5, this value is negligible. This completes the proof of Claim B.1. ∎

At this point, the Execute oracles provide almost no advantage to the adversary $\mathcal{A}$ because the session keys are n-bit strings chosen uniformly at random. We now proceed to show that the Send oracles are also not of "too much" help to the adversary. We divide the Send oracles into four different types:

- $\mathsf{Send}_0(i, l_i)$: this oracle returns the first message (c) as sent by the initiator P_i in protocol instance $\Pi_i^{l_i}$.
- $\mathsf{Send}_1(j, l_j, c)$: this oracle returns the reply (s, c') of P_j upon receiving the message c in protocol instance $\Pi_j^{l_j}$.
- $\mathsf{Send}_2(i, l_i, s, c')$: this oracle returns the final message (s', t) sent by P_i upon receiving the message (s, c') in protocol instance $\Pi_i^{l_i}$.
- $\mathsf{Send}_3(j, l_j, s', t)$: this oracle returns nothing, but it updates the state of $\Pi_j^{l_j}$ by feeding it the message (s', t).

Terminology. Before proceeding, we introduce the following terminology: an encryption c is said to have been oracle-generated if it was output by a Send_0 or Send_1 oracle (otherwise, it is said to have been adversarially-generated). Likewise, if an encryption c was generated by oracle $\Pi_i^{l_i}$, then we say that it was $\Pi_i^{l_i}$-oracle-generated. We say that an encryption c is valid for a protocol instance if it is of the correct format. Thus, c is valid for $\Pi_j^{l_j}$ when received in a Send_1 oracle query if and only if $c = \mathcal{E}_{PK}(w, i \circ j)$ where c is received by $\Pi_j^{l_j}$, w is P_i and P_j's shared password and $\mathsf{pid}_j^{l_j} = i$. Likewise, if c' is received by $\Pi_i^{l_i}$ in a Send_2 oracle query, then it is valid if and only if $c' = \mathcal{E}_{PK}(w, c \circ s)$ where s is the projection key sent along with c' and c is the encryption output by the instance $\Pi_i^{l_i}$. Note that an encryption may be valid for one instance and not for another (by default, when we say that an instance $\Pi_j^{l_j}$ receives a valid or invalid encryption c, the validity refers to $\Pi_j^{l_j}$).

Hybrid experiment H_2: In this hybrid experiment we allow the simulator to choose the common reference string by running the key generation algorithm for the encryption algorithm. So the common reference string is set to PK but the simulator knows SK the matching secret key. Clearly the distribution seen by the adversary is the same.

Now the simulator uses the secret key to determine if any adversarially-generated encryption is *valid* (in particular it contains the correct password). If that happens, the simulator sets the session key of the party receiving the message to a special symbol †. Also the adversary is given a success if it ever queries Reveal or Test on a key set to †. Otherwise the experiment proceeds as in H_1. It clearly holds that:

Claim B.2 *For every non-uniform polynomial-time adversary $\mathcal{A}$,* $\mathsf{Adv}(\mathcal{A}, H_1) \leq \mathsf{Adv}(\mathcal{A}, H_2)$.

Hybrid experiment H_3: In this experiment, if an instance $\Pi_i^{l_i}$ receives a $\mathsf{Send}_2(s, c')$ oracle call in which c' is invalid, then $\Pi_i^{l_i}$ master keys are chosen at random. The rest of the emulation is identical to H_2.

Claim B.3 *For every non-uniform polynomial-time adversary $\mathcal{A}$, For every non-uniform polynomial-time adversary $\mathcal{A}$*

$$|\mathsf{Adv}(\mathcal{A}, H_3) - \mathsf{Adv}(\mathcal{A}, H_2)| < \mathsf{negl}(n).$$

Proof: The proof of this Claim follows immediately from the definition of projective hash function. If c' is invalid i.e. it is not an encryption of w with label $c \circ s$ under public key PK where c is the encryption output in the first message by $\Pi_i^{l_i}$), then the n-bit string $H_{k'}(c', w, c \circ s)$ is distributed almost uniformly. We note that this Claim holds information-theoretically.

We also note that this is the step of the proof where the masking of the MAC with sk_A is needed. Notice that in this experiment we are setting *both* of $\Pi_i^{l_i}$ master keys to random. But the above argument proves that only sk_i (i.e. the one computed hashing the commitment c prepared by $\Pi_i^{l_i}$) is random. If $\Pi_i^{l_i}$ answered with the MAC computed only using sk_j (the other master key), then the adversary could distinguish between the view in this hybrid and its view in H_2. But because the MAC is masked by the randomness of sk_i the view of the adversary remains the same between these two hybrid experiments. ■

Before defining the next hybrid experiment, we prove a lemma stating that if $\mathcal{A}$ forwards the first two messages c and c', s of an execution unmodified, then it cannot modify the projection key s' sent between the parties during the execution, without the party receiving c eventually aborting. Formally, denote by $\mathsf{main}(\Pi_i^{l_i})$ the projection keys and commitments (c, s, c', s') seen by $\Pi_i^{l_i}$ in the execution. The above-mentioned lemma is formally stated as follows:

Lemma B.4 *Denote by* $\mathsf{Send_{01}\text{-}unmodified}(\Pi_i^{l_i}, \Pi_j^{l_j})$ *the event of* $\Pi_j^{l_j}$ *receiving on a* Send_1 *call the exact message output by* $\Pi_i^{l_i}$ *upon a* Send_0 *oracle call, and* $\Pi_i^{l_i}$ *receiving on a* Send_2 *call the exact message output by* $\Pi_j^{l_j}$ *in the previous* Send_1 *call. Then,*

$$\Pr\left[\mathsf{Send_{01}\text{-}unmodified}(\Pi_i^{l_i}, \Pi_j^{l_j}) \;\&\; \mathsf{main}(\Pi_i^{l_i}) \neq \mathsf{main}(\Pi_j^{l_j}) \;\&\; \mathsf{acc}_j^{l_j} = 1\right] < \mathsf{negl}(n).$$

Proof: The proof of this lemma is based on the security of the MAC function and the pseudorandomness of the result of the projective hash function.

By the assumption in the lemma, $\Pi_j^{l_j}$ received the same first message c sent by $\Pi_i^{l_i}$, and $\Pi_i^{l_i}$ received the same message that $\Pi_j^{l_j}$ sent in response. In other words this is "almost" an Execute call, except that the adversary has the power to modify the last message of the protocol. It is not hard to see that it follows from the proof of Claim B.1 that the master key computer by $\Pi_i^{l_i}$ is pseudorandom and thus can be replaced in this execution with a random value.

Now assume that $\mathcal{A}$ is able to modify s' and make $\Pi_j^{l_j}$ accept. This means that we can use $\mathcal{A}$ to forge the MAC, as follows. We are given access to a MAC oracle that on input a message m produces $t_m = MAC_\kappa(m)$ where κ is a random n-bit key. Set up the whole emulation of the entire network, and guess an execution in which the adversary will not modify the first two messages, but will modify the last and make $\Pi_j^{l_j}$ accept.

In this execution "replace" $sk_B^{(1)}$ with κ, i.e. use the MAC oracle to compute the last message for $\Pi_i^{l_i}$, i.e. choose s' and ask the MAC oracle for $\tau = MAC_\kappa(c, s, c', s')$ and then send $t = \tau \oplus sk_A^{(1)}$. Because we have already observed that sk_B and sk_A are pseudo-random to the adversary, the view of the adversary after this change is indistinguishable from the real one.

Now, the adversary will send $\hat{s}', \hat{t}$ to $\Pi_j^{l_j}$ which accepts. The simulator now computes (for $\Pi_j^{l_j}$) the projective hash function over the encryption c' with the projection $\hat{s}'$ that the adversary provided him. It can do that because it has the coins used to generate c'. Let σ be the result. Since $\Pi_j^{l_j}$ accepts it must be that $t \oplus \sigma = MAC_\kappa(c, s, c', \hat{s}')$ and this constitutes a forgery since $s' \neq \hat{s}'$. ■

Hybrid experiment H_4: In this experiment, if $\mathsf{Send}_3(s', t)$ is called on a protocol instance $\Pi_j^{l_j}$, which accepts, then the simulator examines the first message c received by instance $\Pi_j^{l_j}$. If c is oracle-generated, then the session key of $\Pi_j^{l_j}$ is chosen at random. If there exists an instance $\Pi_i^{l_i}$ such that $\mathsf{sid}_i^{l_i} = \mathsf{sid}_j^{l_j}$ then this instance is given the same session key as $\Pi_j^{l_j}$. The rest of the emulation is identical to experiment H_3.

Claim B.5 *For every non-uniform polynomial-time adversary $\mathcal{A}$,*

$$|\mathsf{Adv}(\mathcal{A}, H_4) - \mathsf{Adv}(\mathcal{A}, H_3)| < \mathsf{negl}(n).$$

Proof: The proof of this Claim uses Lemma B.4 in a crucial way.

Assume that the adversary submits a $\mathsf{Send}_3(s', t)$ oracle call to an instance $\Pi_j^{l_j}$, which accepts (i.e. t is the correct MAC). Let c be the encryption that $\Pi_j^{l_j}$ received in the first message, and s, c' be its response. Assume c was generated by oracle $\Pi_i^{l_i}$ (i.e. the adversary did not modify the first message). Then there can be only two cases:

1. The adversary submitted $\mathsf{Send}_2(s, c')$ to instance $\Pi_i^{l_i}$. In this case, since $\Pi_j^{l_j}$ accepts, Lemma B.4 tells us that $\Pi_i^{l_i}$ must have answered s', t. That is $\mathsf{sid}_i^{l_i} = \mathsf{sid}_j^{l_j}$, i.e. the adversary did not modify any messages between $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$;

2. The adversary submitted $\mathsf{Send}_2(\hat{s}, \hat{c}')$ to instance $\Pi_i^{l_i}$ where $[\hat{s}, \hat{c}'] \neq [s, c']$. In this case $\hat{c}'$ is either adversarially-generated or invalid. Indeed remember that in order to be valid for $\Pi_i^{l_i}$, $\hat{c}'$ must be an encryption of w with label $c \circ \hat{s}$ where c is the first message output by $\Pi_i^{l_i}$ and $\hat{s}$ is the label that accompanies $\hat{c}'$. All oracle-generated $\hat{c}'$ (except one) come from different instances and thus for a different c. The only possibility is c', but c' is valid for label s, and if the adversary submits c' it must change the label to $\hat{s} \neq s$ (otherwise we fall in case 1).

In case 1, the master keys for both $\Pi_j^{l_j}$ and $\Pi_i^{l_i}$ can be chosen at random. The proof that the adversary cannot distinguish follows from Claim B.1, since this session is basically an $\mathsf{Execute}$ call, in which the adversary does not modify messages.

In case 2, in the real protocol instance $\Pi_j^{l_j}$ ends with master keys $H_k(c, w, i \circ j)$ and $H_{k'}(c', w, c \circ s)$. Let us focus on the first component. This component is pseudorandom for the adversary since $\mathcal{A}$ does not know the coins used to generate c (here there is a reduction to the pseudorandomness of the projective hash function output). Moreover this component is independent from the key established by the instance $\Pi_i^{l_i}$ which outputted c. Indeed in its Send_2 oracle call $\Pi_i^{l_i}$ received either a valid adversarially-generated commitment and its key is set to † or $\Pi_i^{l_i}$ received an invalid encryption and thus its key is already random. So the session key of $\Pi_j^{l_j}$ can be safely replaced with a random one, making at most a negligible difference in the probability of success of the adversary. ■

Hybrid experiment H_5: In this experiment, if a protocol instance $\Pi_i^{l_i}$ receives an oracle-generated encryption c' in a Send_2 oracle call, then its session key is chosen at random. We stress that if $\Pi_i^{l_i}$ is such that its session key is already chosen at random, then nothing different is done. The indistinguishability of H_4 and H_5 is due to the fact that for oracle-generated encryptions, $\mathcal{A}$ cannot predict the hash function value.

Claim B.6 *For every non-uniform polynomial-time adversary $\mathcal{A}$,*

$$|\mathsf{Adv}(\mathcal{A}, H_5) - \mathsf{Adv}(\mathcal{A}, H_4)| < \mathsf{negl}(n).$$

Proof: Let $\Pi_i^{l_i}$ be a protocol instance that receives an oracle-generated encryption c' in a Send_2 oracle call. We differentiate between the case that c' is valid or not.

1. *c' is not valid:* This case was already handled in H_3 and the key set to random.

2. *c' is valid:* This proof uses the same tools as the proof of Claim B.1. See that proof for the definition of the experiments $\mathsf{Expt\text{-}Hash}$ and $\mathsf{Expt\text{-}Unif}$, and the oracles Hash and $\mathsf{Encrypt}$.

   We show that this case contributes at most a negligible difference to $\mathcal{A}$'s advantage by constructing a machine D to distinguish between $\mathsf{Expt\text{-}Hash}$ and $\mathsf{Expt\text{-}Unif}$ with probability negligibly close to $|\mathsf{Adv}(\mathcal{A}, H_5) - \mathsf{Adv}(\mathcal{A}, H_4)|$. Machine D receives a randomly chosen common public key PK, and chooses random passwords $w_1, w_2, \ldots$ for all the pairs of parties. Then, D emulates the H_4/H_5 experiment exactly as prescribed. The only difference relates to the emulation of the Send_1 and Send_2 oracles. Specifically, let $\Pi_j^{l_j}$ be a protocol instance who was queried with a Send_1 oracle call. Then, rather than computing the encryption c' by itself, D queries its $\mathsf{Encrypt}$ oracle with the appropriate password. Furthermore, if this c' is received unmodified in a Send_2 oracle call by any protocol instance $\Pi_i^{l_i}$, then machine D queries its Hash oracle with c' obtaining (s', h'). The reply of $\Pi_i^{l_i}$ then includes s' and the appropriate portion of the master key is set to be h'. We stress that D only modifies the choice of the master key for instances in which the master key is not already chosen randomly in H_4. If the key is already chosen at random (as defined in H_4 when $\mathsf{sid}_i^{l_i} = \mathsf{sid}_j^{l_j}$), then D leaves it this way. Everything else in the emulation remains unchanged. At the conclusion of the emulation, D outputs whatever $\mathcal{A}$ does. (We note an important point that facilitates the above emulation. In H_4, protocol instances $\Pi_j^{l_j}$ who receive oracle generated Send_1 messages *always* choose their master keys at random, rather than using the projective hash functions. This is crucial because in the emulation, the encryption c' sent by such an instance is obtained by D from the $\mathsf{Encrypt}$ oracle. Therefore, D does not know the random coins used to generate c' and cannot compute the portion of the session key $H_{k'}(c', w, c \circ s)$ when given the projection key s' as chosen by the adversary.)

Now, on the one hand, if D interacts in $\mathsf{Expt\text{-}Hash}$, then the emulation carried out by D is exactly that of H_4. On the other hand, if D interacts in $\mathsf{Expt\text{-}Unif}$, then the emulation is exactly that of H_5 because the session keys of instances $\Pi_i^{l_i}$ receiving oracle-generated c' encryptions are chosen uniformly at random. Thus D distinguishes between $\mathsf{Expt\text{-}Hash}$ and $\mathsf{Expt\text{-}Unif}$ with probability $|\mathsf{Adv}(\mathcal{A}, H_5) - \mathsf{Adv}(\mathcal{A}, H_4)|$. By Corollary 3.5, we have that $\mathcal{A}$'s advantage in H_5 is negligibly close to its advantage in H_4, as required.

This completes the proof of Claim B.6 ■

Hybrid experiment H_6: In this experiment, if a protocol instance $\Pi_j^{l_j}$ receives an *invalid* encryption c' in a Send_1 oracle call, then its session key is chosen at random. This makes at most a negligible difference because when c is not valid, $\Pi_j^{l_j}$'s session key is anyway statistically close to uniform due to the property of projective hash functions. We note that the following Claim holds information-theoretically.

Claim B.7 *For every non-uniform polynomial-time adversary* $\mathcal{A}$

$$|\mathsf{Adv}(\mathcal{A}, H_6) - \mathsf{Adv}(\mathcal{A}, H_5)| < \mathsf{negl}(n).$$

We are now ready to conclude the proof by bounding the advantage of the adversary in experiment H_6. First, consider the case that all the adversarially-generated commitments are *not valid*. Then, in experiment H_6, all master keys are actually chosen independently and uniformly at random in G. Thus the probability that the adversary distinguishes between a real key and a random one is exactly 1/2.

If the adversary generates a valid encryption then in H_6 we say that it succeeds. Due to the non-malleability of the encryption scheme $\mathcal{E}$ the probability that $\mathcal{A}$ outputs a valid encryption must be negligibly close to the probability of guessing the password outright, that is $Q_{send}/|D|$.

Concluding the probability of the adversary succeeding in H_6 is:

$$\frac{1}{2} + \frac{Q_{send}}{|D|} + \mathsf{negl}(n) \tag{1}$$

as desired.

By combining Claims B.1 to B.7, and Equation (1) we obtain that $\mathcal{A}$'s advantage in a real execution is at most $Q_{\mathrm{send}}/|\mathcal{D}| + \mathsf{negl}(n)$, as required. This completes the proof of Theorem 4.1.

C Labels in Cramer-Shoup Encryption Schemes

Consider the original Cramer-Shoup scheme based on the DDH Assumption:

Parameters A cyclic group G of prime order q; two random generators g_1, g_2. A collision resistant hash function H which outputs elements in Z_q.

Key Generation The secret key consists of five random values $z, x_1, x_2, y_1, y_2 \in_R Z_q$. The public key are the group elements $h = g_1^z$, $c = g_1^{x_1} g_2^{x_2}$ and $d = g_1^{y_1} g_2^{y_2}$.

Encryption To encrypt a message $m \in G$, the sender chooses a random value $r \in_R Z_q$ and computes $u_1 = g_1^r$, $u_2 = g_2^r$, $e = h^r \cdot m$, the hash value $a = H(u_1, u_2, e)$ and finally the group element $v = (cd^\alpha)^r$. The ciphertext is (u_1, u_2, e, v).

Decryption The receiver on input (u_1, u_2, e, v), checks that these are indeed elements of G, then computes $a = H(u_1, u_2, e)$ and checks if $v = g_1^{x_1+\alpha y 1} g_2^{x_2+\alpha y_2}$. If the check fails it outputs an error message otherwise it outputs $m = e \cdot u_1^{-z}$.

Cramer and Shoup in [12] prove that the above scheme is secure if the DDH Assumption holds over the group G and H is a target collision resistant (e.g. universal one-way) hash function.

As discussed in [1] to incorporate labels in the above ciphertext it is sufficient to add them to the input of the hash function H. In other words if the sender is encrypting a message m with label ℓ, then the only step that must be modified in the encryption and decryption procedures is the computation of a as $a = H(u_1, u_2, e, \ell)$. Notice that this modification has virtually no impact on the efficiency of the scheme. However, as pointed out in [1], the proof of security requires H to be a fully collision resistant function (actually in [1] a slightly weaker condition is assumed).

It also important to notice that even with this modification there exists a simple way to define what a correct ciphertext is: given a public key $PK = (g_1, g_2, h, c, d)$, a message m, a label ℓ and a ciphertext $C = (u_1, u_2, e, v)$, we have that C is a correct encryption of m with label ℓ under public key PK if and only if

$$\log_{g_1} u_1 = \log_{g_2} u_2 = \log_h \frac{e}{m} = \log_{cd^a} v$$

where $a = H(u_1, u_2, e, \ell)$. This simple property of ciphertext is what allows us to construct efficient projective hash functions for them (as described in [19] following [24]).