A Framework for Password-Based Authenticated Key Exchange[*]

Rosario Gennaro   Yehuda Lindell

IBM T.J.Watson Research Center
19 Skyline Drive,
Hawthorne 10532, USA.
{rosario,lindell}@us.ibm.com

February 14, 2003

Abstract

In this paper we present a general framework for password-based authenticated key exchange protocols, in the common reference string model. Our protocol is actually an abstraction of the key exchange protocol of Katz et al. and is based on the recently introduced notion of smooth projective hashing by Cramer and Shoup. We gain a number of benefits from this abstraction. First, we obtain a modular protocol that can be described using just three high-level cryptographic tools. This allows a simple and intuitive understanding of its security. Second, our proof of security is significantly simpler and more modular. Third, we are able to derive analogues to the Katz et al. protocol under additional cryptographic assumptions. Specifically, in addition to the DDH assumption used by Katz et al., we obtain protocols under both the Quadratic and N-Residuosity assumptions. In order to achieve this, we construct new smooth projective hash functions.

1 Introduction

A central problem in cryptography is that of enabling parties to communicate secretly and reliably in the presence of an adversary. This is often achieved by having the parties run a protocol for generating a mutual and secret session key. This session key can then be used for secure communication using known techniques (e.g., applying encryption and message authentication codes to all communication). Two important parameters to define regarding this problem relate to the strength of the adversary and the initial setup for the parties.

Adversarial power. The problem of session-key generation was initially studied by Diffie and Hellman [13] who considered a passive adversary that can eavesdrop on the communication of the parties, but cannot actively modify messages on the communication line. Thus, the parties are assumed to be connected by reliable, albeit non-private, channels. Many efficient and secure protocols are known for this scenario. In contrast, in this paper, we consider a far more powerful adversary who can modify and delete messages sent between the parties, as well as insert messages of its own choice. Such an adversarial attack could be carried out by the owner of a routing server on the Internet, for example.

[*] An extended abstract of this paper will appear in the proceedings of EUROCRYPT'03.

Setup assumptions. In order to achieve authenticated key exchange, the parties 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). Thus, some initial setup assumption is required. Known setup assumptions range from the case that the parties share high entropy secret keys to the case that all they share are low entropy *passwords* that can be remembered and typed in by human users. Although many secure and efficient protocols exist for the high entropy case, our understanding of the low entropy case is far from satisfactory. This is despite the fact that the most common setup assumption used today in practice is that of passwords.

This paper focuses on the question of password-based key exchange in the face of a powerful, active adversary. Before proceeding further, we describe this setting in some more detail.

Password-based authenticated key-exchange. We consider a multi-party scenario where all pairs of parties share a password that is chosen uniformly from some small dictionary (the assumption of uniformity is made for simplicity only). The parties interact in a network in which an active adversary has full control over the communication lines. Essentially, this means that the parties cannot communicate directly with each other; rather, all communication is carried out via the adversary. Nevertheless, the parties attempt to generate session keys that they can then use to secretly and reliably communicate with each other. An immediate observation is that in this scenario it is impossible to guarantee that the adversary's success is negligible (where success means, for example, that it succeeds in learning the session key). This is because it can guess the password and then impersonate Bob while communicating with Alice. If its password guess was correct, then it clearly obtains the session key. The important point is that because the password dictionary may be small (i.e., polynomial in the security parameter), the success by the adversary in this naive attack may be quite high. This type of attack is called an on-line guessing attack and is inherent whenever security depends on low entropy passwords. The aim of password-based authenticated key exchange is thus to limit the adversary to such an attack only.

Prior related work. The first (unbroken) protocol suggested for password-based session-key generation was by Bellovin and Merritt [4]. This work was very influential and became the basis for much future work in this area [5, 26, 19, 22, 25, 27]. However, these protocols have not been proven secure and their conjectured security is based on heuristic arguments. Despite the strong need for secure password-based protocols, the problem was not treated rigorously until quite recently.

A first rigorous treatment of the problem was provided by Halevi and Krawczyk [18]. They actually considered an *asymmetric* hybrid model in which one party (the server) may hold a high entropy key and the other party (the human) may only hold a password. The human is also assumed to have secure access to a corresponding public-key of the server. The protocol of [18] provides a password-based solution; however, it requires additional setup assumptions beyond that of human passwords. The first (and only currently known) protocol to achieve security without *any* additional setup is that of Goldreich and Lindell [17]. 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 [17] protocol is not efficient and thus cannot be used in practice.

Recently, Katz, Ostrovsky and Yung (KOY) [21] presented a highly efficient protocol for the problem of password-based authenticated key-exchange in the common reference string model. In this model, it is assumed that all parties have access to a set of public parameters, chosen by some trusted third party. Although this is a stronger setup assumption than where only human passwords are shared, it is still significantly weaker than other models that have been studied (like, for example, the Halevi–Krawczyk model). Furthermore, in practice there are settings in which such a setup can reasonably be implemented at little cost.[1] The KOY protocol is based on the Decisional Diffie-Hellman (DDH) assumption and has complexity that is only 5–8 times the complexity of unauthenticated key-exchange protocols. We remark that our work was borne out of an abstraction of the KOY protocol.

We note that password-based authenticated key-exchange protocols in the password only setting have been presented in the random oracle model [1, 6]. In this model, all parties are assumed to have oracle access to a totally random (universal) function [2]. 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 [7]. Thus, the proofs of security of these protocols are actually heuristic in nature.

1.1 Our Contributions

In this paper, we present a framework for password-based authenticated key-exchange protocols in the common reference string model. Our construction is an abstraction of the KOY protocol [21] and uses non-malleable commitments [14], one-time signature schemes and the smooth projective hash functions of Cramer and Shoup [10]. The advantages of this abstraction are as follows:

1. The security of the resulting protocol can be intuitively understood. Our work can thus also be seen as an "explanation" of the KOY protocol (in a similar way to the fact that [10] can be seen as an explanation of [9]).

2. The proof of our protocol is significantly simpler than that of [21], although there are definite similarities in the high-level overview of the proof (see [20] for a full proof of the [21] protocol). Having abstracted out the building blocks of the protocol, the exact requirements on each element of the protocol also become clearer. One specific result of this is that by slightly modifying the [21] protocol, we are able to show that non-malleable commitments suffice (in contrast to [21] whose proof heavily relies on the fact that they use an encryption scheme that is secure against adaptive chosen-ciphertext attacks (CCA2)).

3. The KOY protocol assumes the DDH assumption. We demonstrate additional instantiations of the framework and obtain password-based authenticated key-exchange protocols under both the Quadratic Residuosity and N-Residuosity assumptions. The resulting protocol for N-Residuosity is also highly efficient. In contrast, the protocol based on Quadratic Residuosity is less efficient, but has the advantage of being based on a more standard assumption.

Before presenting our protocol, we briefly (and informally) describe its components (we stress that the descriptions below are very high-level and thus are not accurate):

Non-interactive non-malleable commitments [14]: A non-malleable commitment scheme has the property that given a commitment to a value x, it is hard to generate a commitment to a *related* value y (with probability that is greater than the a priori probability). Non-interactive non-malleable commitments are known to exist in the common reference string model [11, 12]. We actually need these commitments to be *perfectly binding.* In the common reference string model, such schemes can be obtained from any public-key encryption scheme that is non-malleable

[1] An example of where it is reasonable to assume a common reference string is when a large organization wishes to implement secure login for its employees. In this case, the organization is trusted to choose the common reference string properly, and this string can then be hardwired into the software code.

against chosen-plaintext attacks [12]. The common reference string for the non-malleable commitments is taken as the common reference string for our password protocol.

Smooth projective hashing: [10] 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.

An interesting feature of smooth projective hashing is that if L is an NP-language, then for every $x \in L$ it is possible to compute $H_k(x)$ using the projection key $s = \alpha(k)$ and a witness of the fact that $x \in L$.

In this paper we 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. (Of course, as described above, for $x \notin L$ the value of $H_k(x)$ is statistically close to a random element in the range of H_k.)

Our protocol uses a very simple combination of the above tools. In particular, we define a hard-on-the-average NP-language $L = \{(c, m)\}$, where c is a non-malleable commitment to m. (Notice that for a randomly generated commitment, it is hard to know whether or not c is a commitment to m, even given m.) The basic idea behind the protocol is to have the parties exchange non-malleable commitments of the joint password and compute the session key by applying smooth projective hash functions to these commitments. The smooth projective hash functions that they use are based on the hard language L described above. That is, let w be the parties' joint password. Then, for a commitment c we have that $(c, w) \in L$ if and only if c is a commitment of the password w. An informal (and incomplete) description of the protocol appears in Figure 1. We stress that this protocol description is incomplete. In particular, a one-time signature scheme and an additional exchange of a projection key and hash value should also be included.

Security of the protocol framework: We now explain why the protocol is secure. First, consider the case that the adversary passively eavesdrops on a protocol execution between two parties. The security of the session key in this case is based on the above-described property of smooth projective hash functions over hard languages. Specifically, the adversary sees (c, s, c', s') where c and c' are randomly generated commitments of w; in other words, $(c, w), (c', w) \in_R L$. Therefore, $H_k(c, w)$ and $H_{k'}(c', w)$ are pseudorandom and the generated session key is secure. Next, assume that the adversary sends one of the parties a commitment c that it generated itself. If c is a commitment to some value w' which is not the correct password, then the statement (c, w) is *not* in the language L. Therefore, by the smoothness of H_k, the part of the key $H_k(c, w)$ is statistically close to uniform with respect to the adversary (who only sees s and not k). Thus, as above, the generated key meets the requirements for a secret session key. (That is, the adversary has only a negligible advantage in distinguishing the session key output by the parties from a uniformly distributed key that is chosen independently of the protocol.)

Password-Based Session-Key Exchange

- **Common reference string:** a common reference string ρ for a non-malleable non-interactive commitment scheme.
- **Common input:** a shared (low-entropy) password w.
- **The protocol:**
  1. Party P_1 computes a commitment $c = C_\rho(w)$ using common reference string ρ and sends it to party P_2.
  2. Party P_2 chooses a key k for the smooth projective hash function (for the language L described above), and sends its projection $s = \alpha(k)$ to P_1.

     In addition, P_2 computes another commitment $c' = C_\rho(w)$ and sends it to party P_1.
  3. Party P_1 chooses another key k' for the smooth projective hash function, and sends its projection $s' = \alpha(k')$ to P_2.
- **Session key definition:** Both parties compute the session key to be $H_k(c, w) \oplus H_{k'}(c', w)$.
  1. P_1 computes $H_k(c, w)$ using the projection s and its knowledge of a witness for the fact that c is a commitment to the password w (it knows a witness because it generated c). In contrast, it computes $H_{k'}(c', w)$ using its knowledge of k' (and without a witness).
  2. P_2 computes $H_k(c, w)$ using its knowledge of k, and $H_{k'}(c', w)$ using s' and its knowledge of a witness.

Figure 1: An informal and incomplete description of the protocol framework

The only "bad event" that can occur is if the adversary itself generates a commitment to the correct password w. In this case, we cannot say anything about the security of the session key. However, the adversary can succeed in generating such a commitment with only the same probability as guessing the password outright. This is due to the non-malleability of the commitment scheme which implies that all the "correct" commitments that the adversary sees throughout the executions do not help it in generating any new "correct" commitment. Thus, the adversary's success probability is essentially limited to its probability of guessing the password on-line.

Our constructions of smooth projective hash functions. One of the main contributions of this paper regards our use of the recently introduced notion of smooth projective hash functions [10]. First, we find a new and novel application of this notion to password-based key exchange. Second, we construct new smooth projective hash functions for languages not considered by [10]. Specifically, we construct smooth projective hash functions for the language of pairs (c, m) where c is an encryption of m by a CCA2-secure encryption scheme [14]. This suffices for our protocol framework since any CCA2-secure encryption scheme can be used as a non-interactive non-malleable commitment scheme (in the common reference string model). The KOY protocol implicitly contains one such construction for the encryption scheme of Cramer and Shoup based on the DDH assumption [9]. We prove this fact and then build new smooth projective hash functions for the recent CCA2-secure encryption schemes of Cramer and Shoup [10] that are based on the quadratic residuosity and N-residuosity assumptions. Our constructions of these smooth projective hash functions build on the work of [10]. However, it is important to notice the difference between their constructions and ours. Consider the case of quadratic residuosity, for example. The smooth projective hash function constructed by [10] is for the language of quadratic residues in Z_N^*. In contrast, our construction is for the far more involved language of pairs of plaintexts and ciphertexts for the entire encryption scheme of [10]. We remark that our constructions require a careful combination of ideas from both [10] and [21].

In addition to providing new constructions, we also make some modifications to the definition of smooth projective hash functions, as presented in [10]. We both strengthen and weaken their definition (in different ways) so that it suffices for our application. This different use may also open the door to further applications.

1.2 Organization

In Section 2 we present the definitions of password-based authenticated key exchange. Then, in Sections 3 and 4 we define the notions of smooth projective hash functions and non-malleable commitments. The protocol framework and its proof of security are presented in Sections 5 and 6. The remainder of the paper contains our constructions of smooth projective hash functions for CCA2-secure encryption schemes (which also constitute non-malleable commitments in the common reference string model). That is, in Section 7 we present a warm-up example of smooth projective hash functions for the El-Gamal encryption scheme and then describe the KOY smooth projective hash function for the Cramer-Shoup encryption scheme based on the DDH assumption. Then, in Section 8, a relaxed version of smooth projective hash functions that suffices for our protocol is defined, and constructions of relaxed smooth projective hash functions for the Cramer-Shoup encryption schemes based on the N-residuosity and quadratic residuosity assumptions are presented. Finally, in Section 8.6, we compare the efficiency of the three different schemes.

2 Password-Based Authenticated Key Exchange

In this section, we briefly recall the formal security model for password key exchange protocols as presented in [1] (which is in turn based on [3]). For more details and motivation, we refer the reader to [1]. 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.[2] The passwords for all pairs of parties are uniformly (and independently) chosen from a fixed dictionary $\mathcal{D}$.[3] 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.

[2] 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.

[3] 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 $1/|\mathcal{D}|$.

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. In addition, the adversary is given control over a subset of the oracles. (This actually makes no difference in the password-only setting and is ignored from here on.) 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:

- $\mathsf{Execute}(i, l_i, j, l_j)$: 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 comprised of two components: correctness and privacy. We begin by stating the correctness requirement:

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 [8] for more discussion on this issue.) Formally, we say that the adversary $\mathsf{succeeds}$ 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:

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

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 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 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 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 $Q_{\text{send}}/|\mathcal{D}|$.

3 Smooth Projective Hash Functions

A central element of our new framework for password-based key exchange is the recently introduced notion of *smooth projective hashing* of Cramer and Shoup [10]. However, their precise definition actually does not suffice for our application. Rather, we use a variant of their definition. We begin by recalling the original notion of smooth projective hashing (as defined in [10]).

Notation. The security parameter is denoted by n. For a distribution D, $x \leftarrow D$ denotes the action of choosing x according to D. We denote by $x \in_R S$ the action of uniformly choosing an element from the set S. Finally, we denote statistical closeness of probability ensembles by $\overset{\text{s}}{\equiv}$, and computational indistinguishability (with respect to non-uniform polynomial-time machines[4]) by $\overset{\text{c}}{\equiv}$.

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

3.1 The Cramer-Shoup Definition[5]

We begin by defining hard subset membership problems which form the basis for the smooth projective hash functions that are of interest to us.

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 $X_n, L_n \subseteq \{0,1\}^{\text{poly}(n)}$ 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 $W_n \subseteq \{0,1\}^{\text{poly}(n)}$ 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 2 (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*

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

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 being sampled according to the security parameter n. We remark that our definition here differs from the one in [10] in the following way. The definition in [10] relates only to the uniform distributions over X and L, whereas we allow the hard problem to be for any samplable distributions over L and $X \backslash L$. This is needed for our application of smooth hash functions.

[5] We note that our presentation here contains a number of minor differences to the presentation of Cramer-Shoup [10]. Where important, these differences are mentioned.

Smooth projective hash functions. We present the notion of smooth projective hash functions in the context of hard subset membership problems. Thus, the sets X and L mentioned below should be thought of those derived from such a problem.[6] 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 $X \backslash L$ to G. In fact, given the projection key, the distribution over the mapping of $X \backslash L$ 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 $\alpha : K \to S$, 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 $s = \alpha(k)$ uniquely determines $H_k(x)$. (Ignoring issues of efficiency, this means that $H_k(x)$ can be computed given only s and $x \in L$.) We stress that the projection key $s = \alpha(k)$ is only guaranteed to determine $H_k(x)$ for $x \in L$, and nothing is guaranteed for $x \notin L$. Formally,

Definition 3 (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* $\alpha(k)$ *and* x*.*

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

Definition 4 (smooth projective hash functions [10]): *Let* $(\mathcal{H}, K, X, L, G, S, \alpha)$ *be a projective hash family. Then, let* $V(x, \alpha(k), H_k(x))$ *be the following random variable: choose* $x \in X \backslash L$ *according to* $D(X \backslash L)$*,* $k \in_R K$ *and output* $(x, \alpha(k), H_k(x))$*. Similarly, define* $V(x, \alpha(k), g)$ *as follows: choose* $x \in X \backslash L$ *according to* $D(X \backslash L)$*,* $k \in_R K$*,* $g \in_R G$ *and output* $(x, \alpha(k), g)$*. Then, the projective hash family* $(\mathcal{H}, K, X, L, G, S, \alpha)$ *is* smooth *if*

$$\Big\{V(x, \alpha(k), H_k(x))\Big\}_{n \in \mathsf{N}} \overset{\text{s}}{\equiv} \Big\{V(x, \alpha(k), g)\Big\}_{n \in \mathsf{N}}$$

To summarize, a smooth projective hash function has the property that a projection of a key may be computed which enables the computation of $H_k(x)$ for $x \in L$, but gives almost no information about the value of $H_k(x)$ for a random $x \notin L$.

Efficient smooth projective hash functions. We say that a smooth projective hash family is efficient if the following algorithms exist:[7]

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

[6] This is different to the presentation by [10]; they present the notion of smooth projective hash functions independently of hard subset membership problems.

[7] Cramer and Shoup [10] call this structure a smooth projective hash *proof system*, since it serve the role of a *proof* in their construction. We divert from the Cramer-Shoup terminology, and call such families "efficient".

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

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, $\alpha(k)$ (for some $k \in K$), $x \in L$ and a witness w such that $(x, w) \in R$, 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 of the key, and a witness for x (as in item 4 above). This property plays a central role in our password-based protocol.

3.2 A Variant of Smooth Projective Hash Functions

As we have mentioned, the precise definition of smooth hashing presented in [10] does not suffice for our application. Rather, we define a variant which is stronger in one way and weaker in another.

Weakening the projection function. The first difference relates to the definition of the *projection function* α. We do not require the existence of a projection function $\alpha : K \rightarrow S$ such that $s = \alpha(k)$ uniquely defines the value of $H_k(x)$ for *every* $x \in L$. Rather we allow the restriction to be element based. That is, the restriction function receives a key k *and* an element $x \in X$ and outputs a projection $s_x = \alpha(k, x)$. The requirement is that if $x \in L$, then the value $H_k(x)$ is uniquely determined given s_x. However, we do not have any requirement regarding the value of $H_k(x')$ for $x' \neq x$ given s_x, even if $x' \in L$. In other words, the projection only guarantees unique determinability for the element x upon which it was computed. Notice that this is a weaker requirement because smooth hashing of the original definition always satisfies this condition (i.e., define $\alpha(k, x) = \alpha(k)$ for all x).

Strengthening the smoothness property. The smoothness property required in Definition 4 relates to the case that the element x is randomly chosen in $X \backslash L$ (according to the distribution $D(X \backslash L)$). However, for our application, we need to deal with the case that x is *adversarially chosen*. We therefore require smoothness with respect to *every* $x \in X \backslash L$. That is, redefine $V(x, \alpha(k, x), H_k(x))$ to be a random variable where x is *fixed*, $k \in_R K$ is chosen at random and $(x, \alpha(k, x), H_k(x))$ is output. Similarly, redefine $V(x, \alpha(k, x), g)$ where x is fixed, $k \in_R K, g \in_R G$ are chosen at random and $(x, \alpha(k, x), g)$ is output. Then, we require that for *every* $x \in X \backslash L$:

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

This is clearly a strengthening of the requirement. (We note that the notion of ϵ-universal hashing in [10] does relate to the case of an arbitrary $x \in X \backslash L$; however, their notion of smooth hashing does not.)

We remark that the weakening of the projection requirement actually makes it harder to satisfy the stronger smoothness property. This is because we must rule out the case that the adversary finds a "bad" x such that s_x reveals more than it should. Indeed, this technical difficulty arises in our constructions of smooth hash functions under the N-Residuosity and Quadratic Residuosity assumptions. Fortunately, it can be shown that these "bad" x's are hard to find and this will suffice. See Section 8 for more details. From here on, when we refer to smooth projective hashing, we mean the variant defined here.

3.3 A Technical Lemma

By the definition of projective hash functions, in order to compute $H_k(x)$ it suffices to know either the key k, or the projection $s_x = \alpha(k, x)$ and a witness for x. We now prove that for smooth projective hash functions (based on hard subset membership problems), these are actually the only ways to compute $H_k(x)$. Specifically, we show that for $x \leftarrow D(L)$ (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 $x \leftarrow D(L)$. Let M be a (non-uniform) polynomial-time oracle machine. Define the following two experiments.

$\mathsf{Expt\text{-}Hash}(M)$: An instance $\Lambda = (X, D(X\backslash L), L, D(L), W, R)$ 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.

We now prove that no efficient M can distinguish between the experiments. In other words, when $x \leftarrow D(L)$, 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 3.1 *Assume that $\mathcal{I}$ is a hard subset membership problem. Then, for every* (*non-uniform*) *polynomial-time oracle machine M it holds that,*

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

Proof: We begin by proving a preliminary claim. Let $\Omega_{X\backslash L}$ be an oracle that when queried on an empty input returns $x \in X\backslash L$ distributed according to $D(X\backslash L)$. Consider the experiment $\mathsf{Expt\text{-}Hash}_{X\backslash L}$ (resp. $\mathsf{Expt\text{-}Unif}_{X\backslash L}$) which is identical to $\mathsf{Expt\text{-}Hash}$ (resp. $\mathsf{Expt\text{-}Unif}$), except that the oracle Ω_L is replaced with $\Omega_{X\backslash L}$. Then, we prove the following claim.

Claim 3.2 *Assume that $\mathcal{I}$ is a hard subset membership problem. Then, for every* (*non-uniform*) *polynomial-time oracle machine M, the following two equations hold:*

$$\left|\Pr[\mathsf{Expt\text{-}Hash}_{X\backslash L}(M) = 1] - \Pr[\mathsf{Expt\text{-}Hash}(M) = 1]\right| < \mathsf{negl}(n) \quad (1)$$

$$\left|\Pr[\mathsf{Expt\text{-}Unif}_{X\backslash L}(M) = 1] - \Pr[\mathsf{Expt\text{-}Unif}(M) = 1]\right| < \mathsf{negl}(n) \quad (2)$$

Proof: This claim follows from the fact that the Hash and Unif oracles do not need to know whether or not $x \in L$. Now, let M' be a machine who has oracle access to either Ω_L or $\Omega_{X \backslash L}$ and attempts to distinguish these cases. M' receives x values from its oracle, and can perfectly emulate the Hash and Unif oracles itself (without knowing whether or not $x \in L$). This implies that if M' has oracle access to Ω_L then it perfectly emulates $\mathsf{Expt\text{-}Hash}$ (resp., $\mathsf{Expt\text{-}Unif}$). Likewise, if M' has oracle access to $\Omega_{X \backslash L}$ then it perfectly emulates $\mathsf{Expt\text{-}Hash}_{X \backslash L}$ (resp., $\mathsf{Expt\text{-}Unif}_{X \backslash L}$). We conclude that if there exists an efficient machine M that can distinguish the experiments in Eq. (1) or Eq. (2), then M' can distinguish between Ω_L and $\Omega_{X \backslash L}$. This contradicts the hardness of the subset membership problem. (The full details are straightforward and are therefore omitted.) ■

We now resume the proof of the lemma. The quantity we wish to bound can be written as follows:

$$\begin{aligned}
&\Big|\Pr[\mathsf{Expt\text{-}Hash}(D) = 1] - \Pr[\mathsf{Expt\text{-}Unif}(D) = 1]\Big| \\
&\qquad \leq \Big|\Pr[\mathsf{Expt\text{-}Hash}(D) = 1] - \Pr[\mathsf{Expt\text{-}Hash}_{X \backslash L}(D) = 1]\Big| && (3) \\
&\qquad + \Big|\Pr[\mathsf{Expt\text{-}Hash}_{X \backslash L}(D) = 1] - \Pr[\mathsf{Expt\text{-}Unif}_{X \backslash L}(D) = 1]\Big| && (4) \\
&\qquad + \Big|\Pr[\mathsf{Expt\text{-}Unif}_{X \backslash L}(D) = 1] - \Pr[\mathsf{Expt\text{-}Unif}(D) = 1]\Big| && (5)
\end{aligned}$$

Equations (3) and (5) are both negligible by Claim 3.2. Furthermore, the fact that Eq. (4) is negligible immediately follows from the definition of smooth hashing (the experiments in this equation are actually statistically close). This completes the proof of Lemma 3.1. ■

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 partitioned subset membership problem if for *every* i, it is hard to distinguish $x \leftarrow D(L(i))$ from $x \leftarrow D(X(i) \backslash L(i))$. (In the notation of Definition 2, 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.

We now show that Lemma 3.1 also holds for hard partitioned subset membership problems. 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 $x \leftarrow D(L(i))$. Likewise, $\Omega_{X \backslash L}$ receives an index i and returns an element $x \leftarrow D(X(i) \setminus L(i))$. 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.

It is easy to verify that Claim 3.2 still holds and thus the proof of Lemma 3.1 goes through. We therefore have the following corollary:

Corollary 3.3 *Assume that $\mathcal{I}$ is a family of hard* partitioned *subset membership problem. Then, for every (non-uniform) polynomial-time oracle machine M*

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

3.4 Our Usage of Smooth Projective Hashing

As we have mentioned, for our password protocol, we construct smooth projective hash functions for a specific family of hard (partitioned) subset membership problems. In this section we describe this family.

Let $\mathcal{C}$ be a non-interactive non-malleable perfectly-binding commitment scheme; such schemes are known to exist in the common reference string model (see Section 4). We denote by $C_\rho(m;r)$ a commitment to m using random-coins r and common reference string ρ. Such a commitment scheme is the basis for our hard problem. Let C_ρ denote the space of all strings that may be output by the commitment scheme $\mathcal{C}$ when the CRS is ρ, and let M denote the message space. We note that actually, C_ρ and M must be supersets of these spaces that are *efficiently recognizable*; the actual definition of the supersets depends on the specific commitment scheme used; see Sections 7 and 8. Next, define the following sets:

- $X_\rho = C_\rho \times M$.
- $L_\rho = \{(c,m) \mid \exists r \; c = C_\rho(m;r)\}$

Furthermore, define the *partitioning* of X_ρ to be by index m (i.e., consider $X_\rho(m) = C_\rho \times m$ and $L_\rho(m) = \{(c,m) \mid \exists r \; c = C_\rho(m;r)\}$ (i.e., $L_\rho(m)$ is the language of all commitments to m using ρ). The distribution $D(L_\rho(m))$ is defined by choosing a random r and outputting $(C_\rho(m;r),m)$. In contrast, the distribution $D(X_\rho(m)\backslash L_\rho(m))$ is defined by choosing a random r and outputting $(C_\rho(0^{|m|};r),m)$. Clearly, by the hiding property of C, it holds that for every m, random elements chosen from $D(L_\rho(m))$ are computationally indistinguishable from random elements chosen from $D(X_\rho(m)\backslash L_\rho(m))$. This therefore constitutes a hard partitioned subset membership problem. (The witness set W_ρ and NP-relation R_ρ are defined in the natural way.)

To summarize, the problem instance sampler for this problem uniformly chooses a common reference string ρ from CRS. This then defines the sets X_ρ, L_ρ, W_ρ and the NP-relation R_ρ. Taking the distributions $D(L_\rho(m))$ and $D(X_\rho(m) \backslash L_\rho(m))$ as defined above, we have that $\Lambda = (X_\rho, D(X_\rho \backslash L_\rho), L_\rho, D(L_\rho), W_\rho, R_\rho)$ is a hard partitioned subset membership problem.[8]

Our password-based key exchange protocol assumes the existence of a smooth projective hash family for the problem Λ. This hash family $\mathcal{H}$ is indexed by the key space K and it holds that for every $k \in K$,

$$H_k : C_\rho \times M \to G$$

where G is a group of *super-polynomial* size. Let α be the key projection function. We require that α be a function of the hash key and the commitment *only*. That is,

$$\alpha : K \times C_\rho \to S$$

In Section 3.2, we allowed α to depend on the element x, which in this case is a pair (c,m). In our application, m will contain the secret password and we require that α can be computed without

[8] We remark on an important subtlety here. On the one hand, X_ρ is defined over strings that may contain a c that is not a valid commitment to any value (recall that C_ρ is actually a *superset* of the commitments using ρ). On the other hand, the distribution $D(X_\rho(m) \backslash L_\rho(m))$ always outputs c that is a valid commitment to something (specifically to $0^{|m|}$). We define the sets and distributions this way for the following reason. First, the distribution $D(X_\rho(m) \backslash L_\rho(m))$ is not defined over invalid commitments is because we cannot claim that an invalid commitment is indistinguishable from a valid one. (For example, if $D(X_\rho(m) \backslash L_\rho(m))$ were to uniformly choose an element in $X_\rho(m)\backslash L_\rho(m)$, then the resulting problem is not necessarily hard.) In contrast, $X_\rho \setminus L_\rho$ *does* include "everything" that is not a valid (commitment,message) pair. This is because we need the smooth projective hash function to send *everything* that is not a valid pair to a random point (even strings that are not a valid commitment to anything). Of course, if we can efficiently verify that c is not a valid commitment to any message, then this can be excluded from X_ρ. We therefore take X_ρ to be an efficiently recognizable superset of $C_\rho \times M$.

knowledge of the password. Therefore, α is a function of k and c, rather than a function of k and (c, m). This restriction is crucial for the security of the protocol and is used in the proof of Claim 6.7. We remark that our specific constructions of projective hash functions fulfill this additional requirement. For the sake of clarity, we restate some properties of smooth projective hash functions in the context of our specific usage of them:

1. *Efficient hashing from key:* Given any pair $(c, m) \in X_\rho$ and a key k, it is possible to efficiently compute $H_k(c, m)$.

2. *Efficient hashing from projection and witness:* Given a pair $(c, m) \in L_\rho$, the random coins r such that $c = C_\rho(m; r)$ and the projection key $\alpha(k, c)$, it is possible to efficiently compute $H_k(c, m)$.

3. *Smoothness:* For every pair $(c, m) \in X_\rho \backslash L_\rho$ (i.e., for every pair in which c is not a commitment to m), it holds that
$$\Big\{V((c,m), \alpha(k,c), H_k(c,m))\Big\} \overset{\text{s}}{\equiv} \Big\{V((c,m), \alpha(k,c), g)\Big\}$$

4. *Application of Lemma 3.1:* Let ρ be a randomly chosen CRS and let m be a message. Then, for $k \in_R K$ and uniformly chosen coins r, it holds that $\{c = C(m; r), m, \alpha(k, c), H_k(c, m)\}$ is computationally indistinguishable from $\{c = C(m; r), m, \alpha(k, c), g\}$, where $g \in_R G$. That is, for a randomly generated commitment c of m, we have that $H_k(c, m)$ is pseudorandom. Of course, this holds even when seeing many such commitments (as stated in Lemma 3.1).

4 Non-Malleable Commitment Schemes

Our protocol uses non-interactive non-malleable perfectly-binding commitment schemes. In this section we will describe such schemes, as well as a subtle issue that arises regarding these schemes. Loosely speaking, a non-malleable string commitment scheme is a commitment scheme with the additional requirement that a commitment c to a value α is of no help in generating a commitment to a related value β. We note that most standard commitment schemes are easily malleable. The concept of non-malleability was introduced by Dolev et al. in [14], where they also provide a perfectly binding, (interactive) non-malleable commitment scheme based on any one-way function.

We now present the definition of non-interactive non-malleable commitment schemes. Let $\mathcal{A}$ be an adversary who receives a commitment to a value α chosen from some efficiently samplable distribution D, and attempts to output a commitment to a value β that is related to α. (More accurately, $\mathcal{A}$ outputs a commitment to a vector $\bar{\beta}$ that it hopes is related to α.) Then, we say that the commitment scheme is non-malleable if there exists a simulation machine $\mathcal{A}'$ who *does not* receive a commitment to α, yet succeeds in outputting a commitment to a related $\bar{\beta}$ with probability that is at most negligibly smaller than for $\mathcal{A}$ (who *does* receive a commitment to α). Thus, seeing a commitment to α is essentially of no help in generating a commitment to a related value $\bar{\beta}$. This definition has the flavor of semantic security for encryption, in that the success of the adversary is compared to the a priori success of finding a related commitment.

Formally, let C be a perfectly-binding commitment scheme and denote by C_n the scheme when the security parameter n is used. We write $C_n(\alpha)$ to mean a commitment to α generated with uniformly distributed coin tosses. Finally, we denote by $\mathsf{decommit}(c)$ the value committed to in c (i.e., $\mathsf{decommit}(C_n(\alpha)) = \alpha$) and likewise by $\mathsf{decommit}(\bar{c})$ the vector of values committed to in $\bar{c}$. We remark that since C is perfectly binding, every commitment c defines at most one value $\mathsf{decommit}(c)$. We are now ready to define non-malleable commitments:

Definition 5 (non-interactive non-malleable perfectly-binding commitments): *A non-interactive perfectly-binding commitment scheme C is* non-malleable *if for every efficiently samplable distribution D, every polynomial-time relation R, every auxiliary-information function h and every adversary $\mathcal{A}$ that outputs a vector of commitments, there exists an adversarial simulator $\mathcal{A}'$ such that*

$$\Pr_{\alpha \leftarrow D}[(\alpha, \mathsf{decommit}(\mathcal{A}(C_n(\alpha), h(\alpha)))) \in R] < \Pr_{\alpha \leftarrow D}[(\alpha, \mathsf{decommit}(\mathcal{A}'(h(\alpha))) \in R] + \mathsf{negl}(n)$$

The auxiliary-information function h represents partial information on α that may have been obtained. This information is given to both $\mathcal{A}$ and $\mathcal{A}'$.

Non-interactive non-malleable commitment schemes are known to exist in the common reference string model [11, 12]. Perfectly binding schemes of this type can be constructed from non-malleable public-key encryption as follows. Fix the common reference string to be a randomly chosen public-key and commit to a string by encrypting it. Decommitment is carried out in the canonical way by sending the value committed to along with the random coins. The fact that this is a secure perfectly-binding commitment scheme follows immediately from the properties of any (chosen plaintext attack secure) encryption scheme. Non-malleability of the commitment likewise follows from the non-malleability of the encryption scheme. We conclude that any public-key encryption scheme that is non-malleable under chosen plaintext attack can be used as a non-interactive non-malleable perfectly-binding commitment scheme in the common reference string model. This was formally proven in [12].

Non-malleability for multiple commitments. Observe that in the above definition, the adversary is given only a single commitment which it then attempts to maul (we call this non-malleability for a single commitment). However, we actually need a stronger definition in which the adversary is given a *vector* of commitments, rather than a single commitment. Formally,

Definition 6 (non-malleability for multiple commitments): *Let $\ell(\cdot)$ be a polynomial. Then, a non-interactive perfectly-binding commitment scheme C is* non-malleable for ℓ multiple commitments *if for every efficiently samplable distribution D (outputting vectors $\bar{\alpha}$ of $\ell(n)$ elements), every polynomial-time 2ℓ-ary relation R, every auxiliary-information function h and every adversary $\mathcal{A}$, there exists an adversarial simulator $\mathcal{A}'$ such that*

$$\Pr_{\bar{\alpha} \leftarrow D}[(\bar{\alpha}, \mathsf{decommit}(\mathcal{A}(C_n(\bar{\alpha}), h(\bar{\alpha})))) \in R] < \Pr_{\bar{\alpha} \leftarrow D}[(\bar{\alpha}, \mathsf{decommit}(\mathcal{A}'(h(\bar{\alpha}))) \in R] + \mathsf{negl}(n)$$

Achieving non-malleability for multiple commitments. We first note that (perhaps surprisingly) non-malleability for a single commitment does *not* imply non-malleability for multiple commitments. In particular, the DDN-Lite [15] public-key encryption scheme that is non-malleable under chosen plaintext attack (and thus constitutes a non-malleable commitment scheme in the common reference string model) is *malleable* under chosen-plaintext attack in the case of multiple encryptions. (This is in contrast to semantic security for public-key encryption in which security for a single encryption implies security for multiple encryptions.) We remark that in order to obtain a non-malleable commitment scheme of this type from a public-key encryption scheme, one would need to explicitly prove security for multiple encryptions. Another possibility is to use any *adaptive* chosen-ciphertext (CCA2) secure scheme (for which non-malleability and security for multiple encryptions are both implied).

5 The Protocol

Our protocol below uses a non-interactive and non-malleable perfectly-binding commitment scheme. The only known such schemes are in the common reference string (CRS) model, and we therefore write the protocol accordingly. Let ρ denote the CRS and let $C_\rho(m; r)$ denote a commitment to the message m using random coin tosses r and the CRS ρ. We also denote by $C_\rho(m)$ a commitment to m using unspecified random coins. As described in Section 3.4, we use a family of smooth projective functions $\mathcal{H} = \{H_k\}$ such that for every k in the key space K, $H_k : C_\rho \times M \to G$, where M is the message space, C_ρ is an efficiently recognizable superset of $\{C_\rho(m; r) \mid m \in M \ \& \ r \in \{0,1\}^*\}$, and G is a group of super-polynomial size. Recall that the key projection function α is defined as a function of K and C_ρ. See Section 3.4 for more details.

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.

Motivation for the protocol. First notice that both P_i and P_j can compute the session key as instructed. Specifically, P_i can compute $H_k(c, VK \circ w \circ i \circ j)$ because it has the projection key s and the witness r for c. Furthermore, it can compute $H_{k'}(c', w)$ because it has the key k' (and therefore does not need the witness r' for c'). Likewise, P_j can also correctly compute both the hash values (and thus the session key). Second, when both parties P_i and P_j see the same messages (c, s, c', s'), 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 P_i, for example, receives c' that is not a commitment to w by CRS ρ, then P_i's session key will be statistically close to uniform. This is because P_i computes $H_{k'}(c', w)$, and for $c' \notin C_\rho(w)$ we have that the statement (c', w) is not in the language L_ρ defined for $\mathcal{H}$ (see Section 3.4). Therefore, by the definition of smooth projective hashing, $\{c', w, \alpha(k, c'), H_k(c', w)\}$ is statistically close to $\{c', w, \alpha(k, c'), g\}$, where $g \in_R G$ is a random element. The same argument holds if P_j receives c that is not a commitment to $VK \circ w \circ i \circ j$. It therefore follows that if the adversary is to distinguish the session key from a random element, it must hand the parties commitments 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 hash function is pseudorandom in G (see Section 3.3). 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 commitment 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 $Q_{\text{send}}/|\mathcal{D}| + \mathsf{negl}(n)$. We remark that the value h_{test} constitutes a test that c is a valid commitment; this is included to ensure that P_j rejects in case it receives an invalid commitment c.

F-PaKE

Common reference string: A common reference string ρ for a non-interactive and non-malleable perfectly-binding commitment scheme C.

Common private input: A password $w = w_{i,j}$.

Messages:

1. Party P_i chooses a key-pair (VK, SK) from a one-time signature scheme, generates a commitment $c = C_\rho(VK \circ w \circ i \circ j; r)$, and sends (VK, c) to P_j.
2. Party P_j receives (VK, c), and checks that c is of the proper format (i.e., $c \in C_\rho$ as defined in Section 3.4). Then, P_j does the following:

   (a) Choose two keys k and k_{test} for the smooth projective hash function family $\mathcal{H}$ defined in Section 3.4 and compute the projections $s = \alpha(k, c)$ and $s_{\text{test}} = \alpha(k_{\text{test}}, c)$.

   (b) Generate a commitment $c' = C_\rho(w; r')$.

   P_j sends (s, c', s_{test}) to P_i.
3. P_i receives (s, c', s_{test}), checks that $c' \in C_\rho$, and does the following:

   (a) Choose a key k' for $\mathcal{H}$ and compute the projection $s' = \alpha(k', c')$.

   (b) Compute the signature $\sigma = \mathsf{Sign}_{SK}(s, s')$.

   (c) Compute the hash $h_{\text{test}} = H_{k_{\text{test}}}(c, VK \circ w \circ i \circ j)$ using the projection key s_{test} and the witness r.

   P_i then sends $(s', \sigma, h_{\text{test}})$ to P_j.

Session-Key Definition:

- P_i computes $sk = H_k(c, VK \circ w \circ i \circ j) + H_{k'}(c', w)$, where addition is in the group G. (Note that P_i computes the first element using the projection key s and the witness r.)
- P_j checks that the following two conditions hold:

  1. $h_{\text{test}} = H_{k_{\text{test}}}(c, VK \circ w \circ i \circ j)$
  2. $\mathsf{Verify}_{VK}((s, s'), \sigma) = 1$.

  If at least one of the above conditions does not hold, then P_j aborts (setting $\mathsf{acc} = 0$). Otherwise, P_j computes $sk = H_k(c, VK \circ w \circ i \circ j) + H_{k'}(c', w)$. (Note that the second element is computed using the projection key s' and the witness r').

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

Figure 2: A framework for Password Based Key Exchange

This is needed in the proof.

We note one more important point regarding the protocol. Based on the above explanation, one may wonder why it does not suffice to exchange c and s only, without the second exchange of c' and s'. The need for this additional exchange is best demonstrated with the following "attack". Assume that the parties only exchange c and s. Then, the adversary $\mathcal{A}$ can interact with P_i and obtain the commitment c. Next, $\mathcal{A}$ chooses k and returns $s = \alpha(k, c)$ to P_i. Now, if the session key was defined to be $H_k(c, w)$, then $\mathcal{A}$ can distinguish this from random as follows. By traversing the dictionary $\mathcal{D}$ with all possible w's, $\mathcal{A}$ can compile a list of all $|\mathcal{D}|$ possible candidates for the session key ($\mathcal{A}$ can do this because it knows k and so can compute $H_k(c, w)$ without a witness for c). Since $\mathcal{D}$ may be small, this enables $\mathcal{A}$ to easily distinguish the session key from random. This problem is overcome, however, by having the parties repeat the commitment/projection exchange in both directions.

Using CCA2-encryption instead of non-malleable commitments. If a CCA2-encryption scheme is used to implement the non-malleable commitments, then the s_{test} and h_{test} values can be removed from the protocol, improving the efficiency. A full proof of the security of the protocol with this modification is available upon request from the authors. (The proof in this case is actually very different from our proof here and is quite similar to the proof of [21].) This version of the protocol has the advantage of being slightly more efficient.

Protocol F-PaKE and the protocol of KOY [21]. Consider the above-mentioned modification of the protocol that relies on CCA2 encryption. Then, a protocol almost identical to that of [21] is obtained by using the CCA2-encryption scheme of Cramer and Shoup [9] that is based on the DDH assumption. Indeed, as we have mentioned, our protocol framework was obtained through an abstraction of [21].

6 Proof of Security

Our proof of security differs from the proof of [21] in a number of ways. First, our proof is based on more generic primitives and is therefore more modular. Second, our protocol uses a non-malleable commitment, whereas the [21] protocol is based on CCA2-encryption. In particular, their proof uses the capability of decryption in an essential way, whereas ours does not.

Theorem 7 *Assume that C is a non-interactive and non-malleable perfectly-binding commitment scheme that is secure under multiple commitments, and assume that $\mathcal{H}$ is a family of smooth projective hash functions as defined in Section* 3.2*. Then, Protocol* F-PaKE *in Figure* 2 *is a secure password-based session-key generation protocol.*

Proof: We begin by proving the *correctness* requirement. Notice that the session identifiers are defined to be (c, s, c', s') 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 each new hybrid, we modify the way session keys are chosen for certain protocol instances. We begin by choosing random session keys for protocol instances for which the Execute oracle is called. We then proceed to choose random session key for instances that receive Send oracle calls. These instances are gradually changed over five hybrid experiments, depending on specific sub-cases.

In the last hybrid experiment, all the session keys are chosen uniformly at random. Thus, the adversary clearly cannot distinguish them from random (because they *are* random). We denote these hybrid experiments by $H_0, \dots, H_5$ 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 session keys of protocol instances for which Execute is called are all chosen uniformly at random, rather than being computed as a hash of the commitments. Intuitively, this follows from Lemma 3.1 which states that without knowing k or the randomness used to generate a commitment, the distribution $\{s, VK \circ w \circ i \circ j, H_k(c, VK \circ w \circ i \circ j)\}$ is computationally indistinguishable from $\{s, VK \circ w \circ i \circ j, g\}$, where $g \in_R G$. We now formally prove that modifying the oracles in this way can make at most a negligible difference. That is,

Claim 6.1 *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 common reference string ρ is chosen and given to the machine D. D can then query two oracles: Commit($\cdot$) and Hash($\cdot, \cdot$). The Commit oracle receives a message m and returns a commitment $c = C_\rho(m; r)$ (generated using random coins r). The Hash oracle receives an input c that was output by the Commit oracle on input m. It chooses a key $k \in_R K$ and computes the projection $s = \alpha(k, c)$. It then outputs $(s, H_k(c, m))$. We stress that the Hash oracle *only* answers on inputs c which were generated by Commit on input m. 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, which was output by the Commit oracle, it chooses a key $k \in_R K$ and computes the projection $s = \alpha(k, c)$. It then chooses a random element $g \in_R G$ and outputs (s, g).

In Corollary 3.3, 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 problem where X is partitioned by the message m (see Section 3.4). In this context, the Commit 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 Commit and Hash oracles from the Expt-Hash and Expt-Unif experiments are used to obtain the commitments and projection keys, and to compute the session keys. Then, if $\mathsf{Expt\text{-}Hash}$ is being used, the result is identical to the protocol specification, and therefore H_0. In contrast, if $\mathsf{Expt\text{-}Unif}$ is being used, then the session keys are chosen at random for all $\mathsf{Execute}$ 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 $\mathsf{Expt\text{-}Hash}$ and $\mathsf{Expt\text{-}Unif}$, 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 $\mathsf{Expt\text{-}Hash}$ and $\mathsf{Expt\text{-}Unif}$ experiments as follows. D receives a randomly chosen common reference string ρ, 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 $\mathsf{Execute}$ oracles. Rather than computing c, s, c', s' as prescribed by the protocol, D queries its Commit oracle with $VK \circ w \circ i \circ j$ and w obtaining c and c' respectively. Then, the Hash oracle is queried with c and c', returning (s, h) and (s', h') respectively. The projection keys are included in the transcript and the session keys for these instances are chosen as $h + h'$. (We note that D can compute k_{test} and h_{test} itself, without using the Hash oracle and that the distribution is the same in H_0 and H_1.) 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 $\mathsf{Expt\text{-}Hash}$, then the emulation carried out by D is exactly that of H_0. In contrast, if D interacts in $\mathsf{Expt\text{-}Unif}$, then the session keys of protocol instances for which $\mathsf{Execute}$ is invoked are all chosen at random. Thus, the emulation by D is exactly that of H_1. We conclude that D distinguishes between $\mathsf{Expt\text{-}Hash}$ and $\mathsf{Expt\text{-}Unif}$ with probability exactly $|\mathsf{Adv}(\mathcal{A}, H_0) - \mathsf{Adv}(\mathcal{A}, H_1)|$. Thus, by Corollary 3.3, this value is negligible.

This completes the proof of Claim 6.1. ■

At this point, the $\mathsf{Execute}$ oracles provide no advantage to the adversary $\mathcal{A}$ because the session keys are 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 $\Pi_i^{l_i}$.
- $\mathsf{Send}_1(j, l_j, VK, c)$: this oracle returns the reply (s, c', s_{test}) of P_j upon receiving the message (VK, c) in protocol instance $\Pi_j^{l_j}$.
- $\mathsf{Send}_2(i, l_i, s, c', s_{\text{test}})$: this oracle returns the final message $(s', \sigma, h_{\text{test}})$ sent by P_i upon receiving the message (s, c', s_{test}) in protocol instance $\Pi_i^{l_i}$.
- $\mathsf{Send}_3(j, l_j, s', \sigma, h_{\text{test}})$: this oracle returns nothing, but it updates the state of $\Pi_j^{l_j}$ by feeding it the message $(s', \sigma, h_{\text{test}})$.

Before defining the next hybrid experiment, we prove a lemma stating that if $\mathcal{A}$ forwards the first message (VK, c) of an execution unmodified, then it "cannot" modify the projection keys s and s' sent between the parties during the execution. Formally, denote by $\mathsf{proj\text{-}keys}(\Pi_i^{l_i})$ the projection keys (s, s') seen by $\Pi_i^{l_i}$ in the execution (we do not include s_{test} with these keys). The above-mentioned lemma is formally stated as follows:

Lemma 6.2 *Let $\Pi_i^{l_i}$ be a protocol instance with $\mathsf{pid}_i^{l_i} = j$ and denote by $\mathsf{Send}_0\text{-}\mathsf{unmodified}(\Pi_i^{l_i}, \Pi_j^{l_j})$ the event of $\Pi_j^{l_j}$ receiving the exact message output by $\Pi_i^{l_i}$ upon a Send_0 oracle call. Then,*

$$\Pr\left[\mathsf{Send}_0\text{-}\mathsf{unmodified}(\Pi_i^{l_i}, \Pi_j^{l_j}) \ \& \ \mathsf{proj\text{-}keys}(\Pi_i^{l_i}) \neq \mathsf{proj\text{-}keys}(\Pi_j^{l_j}) \ \& \ \mathsf{acc}_j^{l_j} = 1\right] < \mathsf{negl}(n)$$

Proof: The proof of this lemma is based on the security of the one-time signature scheme. By the assumption in the lemma, $\Pi_j^{l_j}$ received the same first message (VK, c) sent by $\Pi_i^{l_i}$. Furthermore, $\Pi_i^{l_i}$'s last message includes a signature on the projection keys $\mathsf{proj\text{-}keys}(\Pi_i^{l_i})$ as seen by $\Pi_i^{l_i}$. Now, $\Pi_j^{l_j}$ accepts only if $\mathsf{Verify}_{VK}(\mathsf{proj\text{-}keys}(\Pi_j^{l_j}), \sigma) = 1$ (where VK is the verification key sent by $\Pi_i^{l_i}$). Therefore, if $\mathsf{proj\text{-}keys}(\Pi_i^{l_i}) \neq \mathsf{proj\text{-}keys}(\Pi_j^{l_j})$, it follows that $\Pi_j^{l_j}$ would reject the signature sent by $\Pi_i^{l_i}$. Under the assumption that $\mathcal{A}$ cannot forge a different signature except with negligible probability, we obtain the lemma. This is formally shown using a reduction to the security of the signature scheme. The reduction is straightforward and is therefore omitted. ■

Terminology. Before proceeding, we introduce the following terminology: a commitment 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 a commitment c was generated by oracle $\Pi_i^{l_i}$, then we say that it was $\Pi_i^{l_i}$-oracle-generated. We say that a commitment 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 $c = C_\rho(VK \circ w \circ i \circ j)$ where VK is the verification key received by $\Pi_j^{l_j}$ along with c, 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 $c = C_\rho(w)$. Finally, we denote by $\mathsf{commitments}(\Pi_i^{l_i})$ the pair (c, c') of commitments that $\Pi_i^{l_i}$ sees in the execution. We are now ready to define H_2.

Hybrid experiment H_2: In this experiment, we consider a protocol instance $\Pi_j^{l_j}$ that receives a $\Pi_i^{l_i}$-oracle-generated commitment c in a Send_1 oracle call. If this occurs then the experiment is modified as follows:

1. If $\mathsf{commitments}(\Pi_i^{l_i}) = \mathsf{commitments}(\Pi_j^{l_j})$, then both $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ are given the same random session key $sk \in_R G$.[9] (Of course, if $\Pi_j^{l_j}$ does not accept, then no session key is actually defined for it.)

2. If $\mathsf{commitments}(\Pi_i^{l_i}) \neq \mathsf{commitments}(\Pi_j^{l_j})$, then $\Pi_j^{l_j}$ receives a random session key $sk \in_R G$, whereas $\Pi_i^{l_i}$'s session key is chosen as in H_1.

We now show that the advantage of $\mathcal{A}$ in H_2 is negligibly close to its advantage in H_1; this follows from the properties of smooth projective hash functions.

Claim 6.3 *For every non-uniform polynomial-time adversary $\mathcal{A}$,*

$$|\mathsf{Adv}(\mathcal{A}, H_2) - \mathsf{Adv}(\mathcal{A}, H_1)| < \mathsf{negl}(n)$$

Proof: Throughout the proof we will refer to oracle instances $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ where $\mathsf{pid}_i^{l_i} = j$ and $\mathsf{pid}_j^{l_j} = i$. We denote by w the joint password of P_i and P_j. Without loss of generality, we assume

[9] We note that there is at most one instance $\Pi_i^{l_i}$ for which $\mathsf{commitments}(\Pi_i^{l_i}) = \mathsf{commitments}(\Pi_j^{l_j})$. Therefore, this specification is well defined and there is no case that $\Pi_i^{l_i}$ must be given two different random session keys (one for $\Pi_j^{l_j}$ and one for $\Pi_j^{l'_j}$).

that $\Pi_i^{l_i}$ plays the first party and $\Pi_j^{l_j}$ the second party. That is, we consider the case that $\Pi_j^{l_j}$ receives a $\Pi_i^{l_i}$-oracle-generated c. The fact that $\mathcal{A}$'s advantage in H_1 is negligibly close to its advantage in H_2 is demonstrated in almost the same way as in Claim 6.1. That is, we construct a machine D to distinguish between $\mathsf{Expt\text{-}Hash}$ and $\mathsf{Expt\text{-}Unif}$ with probability negligibly close to $|\mathsf{Adv}(\mathcal{A}, H_2) - \mathsf{Adv}(\mathcal{A}, H_1)|$. Machine D receives a randomly chosen common reference string ρ, and chooses random passwords $w_1, w_2, \ldots$ for all the pairs of parties. Then, D emulates the H_1 experiment as prescribed, with the following differences:

In the event that $\mathcal{A}$ sends $\Pi_j^{l_j}$ a $\Pi_i^{l_i}$*-oracle-generated* c in a Send_1 oracle query, D does not compute a commitment c'. Rather, it obtains c' by querying its Commit oracle with w, and then obtains (s', h') by querying its Hash oracle with c'. The oracles $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ are modified by D as follows:

1. *Modification of* $\Pi_j^{l_j}$*:* Instance $\Pi_j^{l_j}$'s Send_1 oracle response is (s, c', s_{test}). The values s and s_{test} are generated as in H_1 (i.e., as specified in the protocol), whereas the commitment c' is as obtained by D from the Commit oracle query. In the case that $\Pi_j^{l_j}$ accepts, its session key is set to $H_k(c, vk \circ w \circ i \circ j) + h'$, where h' is as obtained by D from the Hash oracle.

2. *Modification of* $\Pi_i^{l_i}$*:* If $\mathsf{commitments}(\Pi_i^{l_i}) \neq \mathsf{commitments}(\Pi_j^{l_j})$, then no modification is made to $\Pi_i^{l_i}$'s specification. However, if $\mathsf{commitments}(\Pi_i^{l_i}) = \mathsf{commitments}(\Pi_j^{l_j})$, then the following modifications are made. First, $\Pi_i^{l_i}$'s response to its Send_2 oracle call is $(s', \sigma, h_{\text{test}})$. The values σ and h_{test} are generated as in H_1, whereas the projection key s' is as obtained by D from the Hash oracle call described above. Furthermore, the session key of $\Pi_i^{l_i}$ is set to $H_{\tilde{k}}(c, vk \circ w \circ i \circ j) + h'$, where $\tilde{s}$ is the projection key received by $\Pi_i^{l_i}$ in its Send_2 oracle call and h' is as obtained by D from the Hash oracle.

Everything else in the emulation remains unchanged (in particular, commitments c output by the Send_0 oracles and the part of the key $H_k(c, VK \circ w \circ i \circ j)$ are computed by D as in the protocol). At the conclusion of the emulation, D outputs whatever $\mathcal{A}$ does.

We begin by showing that if $\Pi_j^{l_j}$ accepts, then, except with negligible probability, $\mathsf{proj\text{-}keys}(\Pi_i^{l_i}) = \mathsf{proj\text{-}keys}(\Pi_j^{l_j})$. This follows from the following observations. First, if the commitment c is not valid, then $(c, VK \circ w \circ i \circ j) \notin L_\rho$ (recall that $L_\rho = \{(C_\rho(m), m)\}$). Therefore, by the definition of smooth projective hashing, we have that $\{c, VK \circ w \circ i \circ j, s_{\text{test}}, H_{k_{\text{test}}}(c, VK \circ w \circ i \circ j)\}$ is statistically close to $\{c, VK \circ w \circ j, s_{\text{test}}, g\}$ where $g \in_R G$. Thus, the probability that the value h_{test} received by $\Pi_j^{l_j}$ equals $H_{k_{\text{test}}}(c, VK \circ w \circ i \circ j)$ is negligible. We conclude that $\Pi_j^{l_j}$ accepts with at most negligible probability (irrespective of the projection keys). Second, consider the case that the commitment c *is* valid. The fact that c is valid means that (VK, c) was received by $\mathcal{A}$ from a Send_0 oracle call to $\Pi_i^{l_i}$, for some l_i (since c is a commitment to a string containing the verification key generated by $\Pi_i^{l_i}$, the validity of c implies that VK is as generated by $\Pi_i^{l_i}$). Thus, the event $\mathsf{Send}_0\text{-}\mathsf{unmodified}(\Pi_i^{l_i}, \Pi_j^{l_j})$ is true. Therefore, by Lemma 6.2, protocol instance $\Pi_j^{l_j}$ accepts and has a defined session key if and only if $\mathsf{proj\text{-}keys}(\Pi_i^{l_i}) = \mathsf{proj\text{-}keys}(\Pi_j^{l_j})$ (except with negligible probability).[10]

Now, if D interacts in $\mathsf{Expt\text{-}Hash}$, then the emulation carried out by D is negligibly close to H_1. This can be seen as follows. If $\Pi_j^{l_j}$ accepts, then except with negligible probability the

[10] This is important because if $\mathcal{A}$ sends $\Pi_j^{l_j}$ a projection key $\tilde{s}'$, then D would be unable to compute the value $H_{\tilde{k}'}(c', w)$. This is because D obtained c' from its Commit oracle and does not know the random coins r'. Therefore, knowledge of the projection key $\tilde{s}'$ does not suffice for computing the hash value. We note that this is exactly the point in the proof where the signature scheme is needed.

value s' that it received in its Send_3 oracle call is the one obtained by D from its Hash oracle query (this is the argument made in the previous paragraph). Therefore, its session key should be $H_k(c, VK \circ w \circ i \circ j) + H_{k'}(c', w)$, where k' is the hash key from which s' is derived. By the definition of the Hash oracle in $\mathsf{Expt\text{-}Hash}$, $h' = H_{k'}(c', w)$ and thus $\Pi_j^{l_j}$'s session key in the emulation by D is statistically close to its session key in H_1 (the only difference being in the case that $\Pi_j^{l_j}$ accepts and $\mathsf{proj\text{-}keys}(\Pi_i^{l_i}) \neq \mathsf{proj\text{-}keys}(\Pi_j^{l_j})$; however, this is negligible). We now turn to $\Pi_i^{l_i}$. If $\mathsf{commitments}(\Pi_i^{l_i}) \neq \mathsf{commitments}(\Pi_j^{l_j})$, then no modifications are made and so the result is exactly H_1. Otherwise, consider the case that $\mathsf{commitments}(\Pi_i^{l_i}) = \mathsf{commitments}(\Pi_j^{l_j})$. In H_1, the session key of $\Pi_i^{l_i}$ would be $H_{\tilde{k}}(c, VK \circ w \circ i \circ j) + H_{k'}(c', w)$ (where $\tilde{s}$ is the projection key received by $\Pi_i^{l_i}$ in its Send_2 oracle call). As above, when h' is obtained from the Hash oracle in $\mathsf{Expt\text{-}Hash}$, $h' = H_{k'}(c', w)$ and so $\Pi_i^{l_i}$'s session key is distributed exactly as in H_1. We conclude that the result of D's emulation when running $\mathsf{Expt\text{-}Hash}$ is negligibly close to H_1.

Next, if D interacts in $\mathsf{Expt\text{-}Unif}$, then we claim that the emulation carried out by D is negligibly close to H_2. First, if $\Pi_j^{l_j}$ accepts, then its session key equals $H_k(c, VK \circ w \circ i \circ j) + h'$ where $h' \in_R G$. It therefore has a uniformly distributed session key, as required. We now consider $\Pi_i^{l_i}$. If $\mathsf{commitments}(\Pi_i^{l_i}) \neq \mathsf{commitments}(\Pi_j^{l_j})$, then D makes no modifications to $\Pi_i^{l_i}$'s specification, as required. If $\mathsf{commitments}(\Pi_i^{l_i}) = \mathsf{commitments}(\Pi_j^{l_j})$, then $\Pi_i^{l_i}$'s session key is defined to be $H_{\tilde{k}}(c, vk \circ w \circ i \circ j) + h'$. Since h' is uniformly distributed, we have that $\Pi_i^{l_i}$ also outputs a uniformly distributed session key. It remains to show that when $\Pi_j^{l_j}$ accepts, $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ have the *same* random session key (recall that if $\Pi_j^{l_j}$ rejects then it has no session key and so we need not show equality). Now, as shown above, if $\Pi_j^{l_j}$ accepts, then except with negligible probability we have that $\mathsf{proj\text{-}keys}(\Pi_i^{l_i}) = \mathsf{proj\text{-}keys}(\Pi_j^{l_j})$. Thus, $\tilde{s} = s$ and $\Pi_i^{l_i}$'s session key equals $H_k(c, vk \circ w \circ i \circ j) + h'$, which is the same session key output by $\Pi_j^{l_j}$. This implies that D's emulation is negligibly close to H_2.

We conclude that D distinguishes between $\mathsf{Expt\text{-}Hash}$ and $\mathsf{Expt\text{-}Unif}$ with probability that is negligibly close to $|\mathsf{Adv}(\mathcal{A}, H_0) - \mathsf{Adv}(\mathcal{A}, H_1)|$. Thus, by Corollary 3.3, this value is negligible. This completes the proof of Claim 6.3.[11] ■

In the above hybrid we modified $\Pi_j^{l_j}$'s choice of a session key when it received an oracle-generated first message. We now modify $\Pi_j^{l_j}$'s choice of session key when it receives an adversarially-generated commitment.

Hybrid experiment H_3: In this experiment, we consider an instance $\Pi_j^{l_j}$ that receives an adversarially-generated commitment c in a Send_1 oracle call. In this case, if $\Pi_j^{l_j}$ accepts, then the experiment is halted and the adversary is said to have succeeded. (Recall that $\Pi_j^{l_j}$ only ac-

[11] We remark on a special case where $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ are such that $\mathsf{commitments}(\Pi_i^{l_i}) = \mathsf{commitments}(\Pi_i^{l_i})$ and yet $i = j$. Such a case would be a problem because by the description of H_2, both $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ should receive the same random session key. However, since these instances are *not* partnered (by the definition of partnering, a party cannot be partnered with itself), the adversary can distinguish $\Pi_i^{l_i}$'s session key from random by querying Reveal on $\Pi_j^{l_j}$'s session key. Nevertheless, this is not a problem because when $i = j$, no $\Pi_i^{l_i}$-oracle-generated commitment c is valid for $\Pi_j^{l_j}$. This is due to the fact that $\mathsf{pid}_j^{l_j} \neq j$ and thus the commitment contains the wrong identities. Therefore, with overwhelming probability $\Pi_j^{l_j}$ does not accept in such a case.

cepts if $h_{\text{test}} = H_{k_{\text{test}}}(c, VK \circ w \circ i \circ j))$. Now, this only improves the probability of success of the adversary. Therefore,

Claim 6.4 *For every non-uniform polynomial-time adversary $\mathcal{A}$:*

$$\mathsf{Adv}(\mathcal{A}, H_2) < \mathsf{Adv}(\mathcal{A}, H_3)$$

Notice that in H_3, whenever $\Pi_j^{l_j}$ concludes with a session key, it is random. That is, there are two possible cases. First, $\Pi_j^{l_j}$ may receive an oracle-generated commitment; this was dealt with already in H_2. Second, $\Pi_j^{l_j}$ may receive an adversarially-generated commitment; in this case it either rejects (in which case it does not conclude with any session key) or the experiment is halted with the adversary succeeding. Having modified the session keys received by $\Pi_j^{l_j}$, we proceed to modify the session keys received by $\Pi_i^{l_i}$.

Hybrid experiment H_4: In this experiment, if a protocol instance $\Pi_i^{l_i}$ receives an oracle-generated commitment 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, because it fulfills the conditions of H_3, then nothing different is done. The indistinguishability of H_3 and H_4 is due to the fact that for oracle-generated commitments, $\mathcal{A}$ cannot predict the hash function value. The argument here is very similar to Claim 6.3.

Claim 6.5 *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: Let $\Pi_i^{l_i}$ be a protocol instance that receives an oracle-generated commitment c' in a Send_2 oracle call. We differentiate between the case that c' is a valid commitment and the case that c' is not valid:

1. *c' is not valid:* As previously shown, when c' is not valid, the definition of smooth projective hashing implies that the resulting session key is statistically close to uniform. This case therefore contributes at most a negligible factor to the difference between $\mathcal{A}$'s advantage in H_3 and $\mathcal{A}$'s advantage in H_4.

2. *c' is valid:* Similarly to the proof of Claim 6.3, 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 $|\mathsf{Adv}(\mathcal{A}, H_4) - \mathsf{Adv}(\mathcal{A}, H_3)|$. Machine D receives a randomly chosen common reference string ρ, and chooses random passwords $w_1, w_2, \ldots$ for all the pairs of parties. Then, D emulates the H_3/H_4 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 commitment c' by itself, D queries its Commit 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 session key is set to be h'. We stress that D only modifies the choice of the session key for instances in which the session key is not already chosen randomly in H_3. If the key is already chosen at random (as defined in H_2), 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_3, protocol instances $\Pi_j^{l_j}$ who receive Send_1 oracle calls *always* choose their session keys at random, rather than using the projective hash functions. This is crucial because in the emulation, the commitment c' sent by such an instance is obtained by D from the Commit oracle. Therefore, D does not know the random coins used to generate c' and cannot compute the portion of the session key $H_{k'}(c', w)$ 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_3. On the other hand, if D interacts in $\mathsf{Expt\text{-}Unif}$, then the emulation is exactly that of H_4 because the session keys of instances $\Pi_i^{l_i}$ receiving oracle-generated c' commitments are chosen uniformly at random. Thus D distinguishes between $\mathsf{Expt\text{-}Hash}$ and $\mathsf{Expt\text{-}Unif}$ with probability $|\mathsf{Adv}(\mathcal{A}, H_4) - \mathsf{Adv}(\mathcal{A}, H_3)|$. By Corollary 3.3, we have that $\mathcal{A}$'s advantage in H_4 is negligibly close to its advantage in H_3, as required.

This completes the proof of Claim 6.5 ■

It remains to modify the session keys of protocol instances $\Pi_i^{l_i}$ receiving adversarially generated commitments c' in Send_2 oracle calls:

Hybrid experiment H_5: In this experiment, if a protocol instance $\Pi_i^{l_i}$ receives an *invalid* adversarially-generated commitment c' in a Send_2 oracle call, then its session key is chosen at random. This makes at most a negligible difference because when c' is not valid, $\Pi_i^{l_i}$'s session key is anyway statistically close to uniform. We remark that we have no way of efficiently checking whether or not such a c' is valid; however, we do not need to since the argument here is information theoretic. We therefore have:

Claim 6.6 *For every non-uniform polynomial-time adversary $\mathcal{A}$*

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

We are now ready to conclude the proof by bounding the advantage of the adversary in experiment H_5. First, consider the case that all the adversarially-generated commitments are *not valid.* Then, in experiment H_5, all session keys are actually chosen independently and uniformly at random. (The only exception is for partnered protocol instances $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ who receive the same key.) Intuitively, since the session keys are all random, $\mathcal{A}$'s advantage in this case is actually zero. However, if $\mathcal{A}$ successfully generates a *valid* commitment, it may be able to distinguish a random value from the real session key. We therefore bound the probability that $\mathcal{A}$ generates such a commitment. By the non-malleability of the commitment scheme, the oracle-generated commitments that $\mathcal{A}$ sees during the execution are of no help. Therefore, it can succeed in generating a valid commitment with essentially the same probability as guessing the password; i.e., the number of commitments that $\mathcal{A}$ generates divided by $|\mathcal{D}|$.

Claim 6.7 *Let $\mathcal{A}$ be a non-uniform polynomial-time adversary that makes at most Q_{send} queries to the Send_1 and Send_2 oracles. Then,*

$$\mathsf{Adv}(\mathcal{A}, H_5) < \frac{Q_{\text{send}}}{|\mathcal{D}|} + \mathsf{negl}(n)$$

Proof: Let $\mathsf{adv\text{-}invalid}$ be the event that all adversarially generated commitments are not valid. Conversely, let $\mathsf{adv\text{-}valid}$ be the event that at least one adversarially generated commitment was valid. Then, we have that

$$\begin{aligned}
&\Pr[\mathcal{A} \text{ succeeds in } H_5] \\
&\quad = \Pr[\mathcal{A} \text{ succeeds in } H_5 \mid \mathsf{adv\text{-}invalid}] \cdot \Pr[\mathsf{adv\text{-}invalid}] \\
&\qquad + \Pr[\mathcal{A} \text{ succeeds in } H_5 \mid \mathsf{adv\text{-}valid}] \cdot \Pr[\mathsf{adv\text{-}valid}] \\
&\quad \leq \Pr[\mathcal{A} \text{ succeeds in } H_5 \mid \mathsf{adv\text{-}invalid}] \cdot \Pr[\mathsf{adv\text{-}invalid}] + \Pr[\mathsf{adv\text{-}valid}]
\end{aligned}$$

Rewriting this equation, we obtain

$$\begin{aligned}
\Pr[\mathcal{A} \text{ succeeds in } H_5] \leq\ & \Pr[\mathcal{A} \text{ succeeds in } H_5 \mid \mathsf{adv\text{-}invalid}] \\
& + \Pr[\mathsf{adv\text{-}valid}] \cdot \Big(1 - \Pr[\mathcal{A} \text{ succeeds in } H_5 \mid \mathsf{adv\text{-}invalid}]\Big)
\end{aligned} \tag{6}$$

We now bound the above quantities.

Bounding $\mathcal{A}$'s success when all adversarially-generated commitments are invalid. We first bound the probability that $\mathcal{A}$ succeeds conditioned on the fact that all the adversarially-generated commitments are invalid. Recall that in H_5, adversary $\mathcal{A}$ succeeds if it distinguishes a session key from random, or if in a Send_1 oracle query it gives an adversarially-generated commitment c to a protocol instance $\Pi_j^{l_j}$ that accepts. The case we are considering here is where all such commitments c are not valid; therefore, $(c, VK \circ w \circ i \circ j) \notin L_\rho$ and h_{test} is (almost) uniformly distributed in G. Therefore, the probability that $\Pi_j^{l_j}$ accepts is negligible. It remains to bound the probability that $\mathcal{A}$ can distinguish a session key from random.

Now, in experiment H_5, when all adversarially-generated commitments are invalid, all the session keys are chosen at random. Therefore, the Test oracle generates exactly the same distribution when it is outputting the chosen session key or a random key. Thus, the only way that $\mathcal{A}$ can distinguish the session key from a random key is if it has explicitly seen this key by using the Reveal oracle. However, in such a case, the key of this instance and its partnered instance cannot be counted as adversarial success. This leaves the case that the same key is chosen for two instances that are not partnered. However, the only case where protocol instances $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ receive the same session key is when $\mathsf{commitments}(\Pi_i^{l_i}) = \mathsf{commitments}(\Pi_j^{l_j})$ and $\mathsf{acc}_j^{l_j} = 1$. Now, in such a case, it must be that $\mathsf{Send}_0\text{-}\mathsf{unmodified}(\Pi_i^{l_i}, \Pi_j^{l_j})$ occurred, $\mathsf{pid}_i^{l_i} = j$ and $\mathsf{pid}_j^{l_j} = i$. (Otherwise, c would not be valid, due to the verification key or identities in the commitment, and with overwhelming probability $\Pi_j^{l_j}$ rejects.) Now, since $\mathsf{Send}_0\text{-}\mathsf{unmodified}(\Pi_i^{l_i}, \Pi_j^{l_j})$ holds, it follows that except with negligible probability, both instances also saw the same keys s and s' (by Lemma 6.2). We therefore have that both instances have the same series (c, s, c', s') in their views. That is, $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ are such that $\mathsf{pid}_i^{l_i} = j$, $\mathsf{pid}_j^{l_j} = i$ and $\mathsf{sid}_i^{l_i} = \mathsf{sid}_j^{l_j} \neq null$. In other words, $\Pi_i^{l_i}$ and $\Pi_j^{l_j}$ are partnered. We conclude that non-partnered pairs always have independent and uniformly chosen keys and thus $\mathcal{A}$'s probability of success in this case is negligibly close to 1/2. That is,

$$\Pr[\mathcal{A} \text{ succeeds in } H_5 \mid \mathsf{adv\text{-}invalid}] = \frac{1}{2} \pm \mathsf{negl}(n) \tag{7}$$

Bounding the probability that $\mathcal{A}$ generates a valid commitment. We now upper bound the probability that $\mathcal{A}$ ever generates a valid commitment by $Q_{\text{send}}/|\mathcal{D}| + \mathsf{negl}(n)$. This is proved by a reduction to the non-malleability of the commitment scheme (for multiple commitments). See Definition 6 (non-malleability for multiple commitments) for the notation used below.

We begin by defining a distribution D and a relation R. Let $\ell(n)$ equal $3t \cdot N^2$ where t is a bound on the running-time of the (password protocol) adversary $\mathcal{A}$, and N equals the number of participating parties. Then, the distribution D chooses N^2 passwords $w_{i,j}$ randomly from the dictionary $\mathcal{D}$, so that $w_{i,j}$ is the joint password of P_i and P_j (to be exact, $N(N-1)/2$ passwords are needed). In addition, D randomly chooses $t \cdot N^2$ pairs of signing and verification keys (VK, SK) for the one-time signature scheme. Then, for every i and j, D generates t strings of the form $(VK \circ w_{i,j} \circ i \circ j)$, t strings of the form $(VK \circ j \circ w_{i,j})$ and t strings of the form $(w_{i,j})$. That is, D outputs all the possible commitment values that could be seen by $\mathcal{A}$ through its $\mathsf{Execute}$ and Send oracles. (Notice that since $\mathcal{A}$ runs for only t steps, this is the most number of oracle calls it can make. Since it can make these calls adaptively, we have D prepare all the possible responses ahead of time.) We remark that the order of the strings is fixed and known.

Next, we define a relation R such that $(\bar{\alpha}, \bar{\beta}) \in R$ if $\bar{\alpha}$ is a $3tN^2$-length vector of the format output by D, and $\bar{\beta}$ is a $3tN^2$-length vector where at most Q_{send} entries are not $\perp$ *and* there exists an index l $(1 \leq l \leq 3tN^2)$ for which α_l and β_l contain the same password $w_{i,j}$. In other words, $\bar{\beta}$ is a vector that contains a correct guess of at least one of the passwords. (The fact that the same password should be in the l^{th} entry in both $\bar{\alpha}$ and $\bar{\beta}$ ensures that $\mathcal{A}$ correctly guessed the password of a specific pair of parties. Furthermore, the requirement that only Q_{send} places are not $\perp$ ensures that only Q_{send} password guesses are counted.)

We now build an attacker M for the non-malleable commitment scheme who succeeds in outputting a related vector of commitments with the same probability that $\mathcal{A}$ sends a valid adversarially-generated commitment in a Send_1 or Send_2 oracle call in its attack on the password protocol. The machine M is given all the VK_i's that are chosen by D along with all the corresponding SK_i's (this is the auxiliary information function h that is applied to α). In addition, M receives the series of $3tN^2$ commitments corresponding to the $3tN^2$ strings output by D. Then, M invokes $\mathcal{A}$ and emulates an execution of the H_5 hybrid experiment. This emulation is carried out as follows. Whenever, a protocol instance $\Pi_i^{l_i}$ or $\Pi_j^{l_j}$ needs to send a commitment c or c', machine M takes the commitment of the appropriate form from the commitments that it received. (Since the order of strings output by D is known, M can make sure that the same pairs of parties always use the same password and that the format of the commitments is always correct.) Furthermore, the projection keys are chosen as instructed[12], the appropriate messages are signed as required (recall that M knows the signature keys and so it can sign), and the session keys are *all* chosen randomly (taking care that when $\mathsf{commitments}(\Pi_i^{l_i}) = \mathsf{commitments}(\Pi_j^{l_j})$ they have the same keys, as specified in H_2). Finally, during the emulation, M records all of the adversarially-generated commitments sent by $\mathcal{A}$ (in Send_1 and Send_2 oracle calls) and to whom $\mathcal{A}$ sent them. At the end of the emulation, M defines a vector of $3tN^2$ commitments so that if $\mathcal{A}$ sent an adversarially-generated commitment c in a Send_1 oracle call to $\Pi_j^{l_j}$ where $\mathsf{pid}_j^{l_j} = i$, then c is placed in the same position as a string of the form $(VK \circ w_{i,j} \circ i \circ j)$ in $\bar{\alpha}$. Likewise, if $\mathcal{A}$ sent an adversarially-generated commitment c' in a Send_2 oracle call to $\Pi_i^{l_i}$ where $\mathsf{pid}_i^{l_i} = j$, then c' is placed in the same position as a string of the form $(w_{i,j})$ in $\bar{\alpha}$. Recall that the order of strings in α is known and so M can do this. The rest of

[12] This is the point in the proof where it is needed that the projection key is chosen as a function of k and c only, and not a function of the message m committed to in c as well (see Section 3.4). This is needed because M does not know the passwords and therefore does not know m. Were it necessary to know m to compute the projection key, M could not carry out the emulation.

the commitments in M's output vector are to $\perp$ (M generates these itself).

Now, if in a Send_1 or Send_2 oracle call, $\mathcal{A}$ sends any valid commitment that was not oracle generated, then M outputs a commitment to $\bar{\beta}$ where $(\bar{\alpha}, \bar{\beta}) \in R$. Furthermore, the probability that $\mathcal{A}$ sends such a valid commitment in the emulation with M is statistically close to the probability that it sends such a commitment in H_5. This can be seen as follows. If $\mathcal{A}$ never sent a valid adversarially-generated commitment during the emulation, then M's emulation is exactly that of H_5. In contrast, once $\mathcal{A}$ sends such a valid commitment, the emulation by M may not be correct (specifically, the key chosen by $\Pi_i^{l_i}$ should not be random). However, if this occurs, then $\mathcal{A}$ already sent a valid commitment and so the event being considered already occurred. We conclude that the probability that $\mathcal{A}$ generates a valid commitment is bound (up to a negligible difference) by the probability that M can output a commitment to $\bar{\beta}$ for which $(\bar{\alpha}, \bar{\beta}) \in R$. By the definition of non-malleability, we have that M can succeed in outputting a $\bar{\beta}$ for which $(\bar{\alpha}, \bar{\beta}) \in R$ with probability only negligibly greater than a machine M' who does *not* receive a commitment to $\bar{\alpha}$. This machine M' is given all the verification and signing keys (like M), but has no information about the passwords. Therefore, the probability that any given non-$\perp$ value in $\bar{\beta}$ contains the correct password is $1/|\mathcal{D}|$. Since there are at most Q_{send} non-$\perp$ values, by the union-bound the probability that M' can output a commitment to $\bar{\beta}$ such that $(\bar{\alpha}, \bar{\beta}) \in R$ is at most $Q_{\text{send}}/|\mathcal{D}|$. In summary, the probability that in a Send_1 or Send_2 oracle call, $\mathcal{A}$ sends a valid non oracle-generated commitment is at most $Q_{\text{send}}/|\mathcal{D}| + \mathsf{negl}(n)$. That is,

$$\Pr[\mathsf{adv\text{-}valid}] < \frac{Q_{\text{send}}}{|\mathcal{D}|} + \mathsf{negl}(n) \tag{8}$$

Combining Equations (6), (7) and (8), we obtain:

$$\Pr[\mathcal{A} \text{ succeeds in } H_5] < \frac{1}{2} + \frac{1}{2} \cdot \frac{Q_{\text{send}}}{|\mathcal{D}|} + \mathsf{negl}(n)$$

and therefore $\mathsf{Adv}(\mathcal{A}, H_5) < \frac{Q_{\text{send}}}{|\mathcal{D}|} + \mathsf{negl}(n)$. This completes the proof of Claim 6.7. ■

By combining Claims 6.1 to 6.7, we obtain that $\mathcal{A}$'s advantage in a real execution is at most $Q_{\text{send}}/|\mathcal{D}| + \mathsf{negl}(n)$, as required. This completes the proof of Theorem 7. ■

7 Encryption Schemes With Smooth Projective Hashing

In this section we describe some examples of encryption schemes that admit efficient constructions of smooth projective hashing. (Recall that although our protocol framework uses non-interactive non-malleable commitments, all specific instantiations are via CCA2-secure encryption schemes. Thus, in all our constructions, we refer directly to the notion of encryption.) As we discussed in Section 3.4, the underlying language for the hard subset membership problem that we refer to is the language of pairs (c, m) where $c = E_{pk}(m)$. (In the context of our protocol, the common reference string ρ is defined to be the public-key pk of the encryption scheme, and a commitment to m is defined as an encryption of m.)

In the remainder of this paper, we construct smooth projective hash functions for the three CCA2-secure encryption schemes proposed by Cramer and Shoup [9, 10]. Our presentation assumes basic familiarity with these encryption schemes. We actually begin by presenting smooth projective hash functions for the El-Gamal encryption scheme. This scheme is only secure against chosen plaintext attack. Nevertheless, the specific construction is quite straightforward and therefore serves as a good warm-up. For an outline of the organization of the rest of the paper, see Section 1.2.

7.1 El-Gamal Encryption

The El-Gamal encryption scheme is defined as follows:

- *Key and message spaces:* Let n be the security parameter, let G be a cyclic group of prime order q where $|q| = n$, and let g be a generator g of G. Then, the space of messages and public keys is G, and the space of ciphertexts is G^2. These parameters are the same for all public-keys generated with security parameter n.
- *Key generation:* The key generation algorithm chooses a random $z \in_R Z_q^*$. The secret-key is then defined to be z and the public-key is defined to be $h = g^z$.
- *Encryption:* To encrypt $m \in G$, a random $r \in_R Z_q$ is chosen and the ciphertext is defined to be $(u, e) = (g^r, h^r \cdot m)$.
- *Decryption:* Upon input (u, e), decryption is carried out by computing $m = e \cdot u^{-z}$.

It is well known that under the Decisional Diffie-Hellman Assumption over G, the El-Gamal scheme is semantically secure against chosen-plaintext attacks. Now, in order to define a smooth projective hash function for this encryption scheme, we need to specify the key space K, the projection function α, and the hash function H_k:

- *Key space:* The key space is defined by $K = Z_q^2$ (i.e., a key is a pair (a_1, a_2), with $a_i \in_R Z_q$).
- *Projection:* The key projection function α is defined by $s_x = \alpha(a_1, a_2, c) = g^{a_1} h^{a_2}$. (We note that in this specific example, the projection depends only on the hash key $k = (a_1, a_2)$ and not on the specific ciphertext. This coincides with the original formulation of projective hash functions by Cramer and Shoup [10].)
- *Smooth hash function:* The hash function is defined as

$$H_k((u,e),m) = u^{a_1}\left(\frac{e}{m}\right)^{a_2}$$

We first show correctness. That is, we show that given s_x, a plaintext/ciphertext pair $((u, e), m)$ and the random coins r used in computing (u, e), it is possible to compute $H_k((u, e), m)$. This follows from the fact that

$$(s_x)^r = (g^{a_1})^r (h^{a_2})^r = (g^r)^{a_1} (h^r)^{a_2} = u^{a_1}\left(\frac{e}{m}\right)^{a_2} = H_k((u,e),m)$$

It now remains to prove the smoothness property. Consider $x = ((u, e), m) \notin L$ (i.e., x such that (u, e) is *not* an encryption of m, under the public key h). Then this implies that $(u, e) = (g^r, h^{r'} \cdot m)$ with $r \neq r'$. Now consider the distribution of $H_k(x) = u^{a_1}(\frac{e}{m})^{a_2} = g^{ra_1 + r'za_2}$ given $s_x = g^{a_1+za_2}$. Since $r \neq r'$, we have that the two equations

$$a_1 + za_2 = \log_g s_x$$

$$ra_1 + r'za_2 = \log_g H_k(x)$$

are linearly independent. That is, for every choice of s_x and $H_k(x)$, there exists a pair (a_1, a_2) that fulfills this equation. Therefore, s_x provides no information on $H_k(x)$ and $H_k(x)$ is *uniformly distributed* over G, given s_x. We conclude that the projective hash function is smooth.

7.2 The DDH Cramer-Shoup Scheme

We now extend the above smooth projective hashing scheme to the Cramer-Shoup CCA2-secure encryption scheme [9]. This family of smooth projective hash functions is actually implicit in the KOY protocol [21]. In the [9] scheme, for every n, the values q and G are fixed as in the El-Gamal scheme described above. Now, the key generation algorithm chooses two additional random generators $g_1, g_2 \in_R G$ and a universal one-way hash function H [23]. The key generation algorithm also chooses $z, \tilde{z}_1, \tilde{z}_2, \hat{z}_1, \hat{z}_2 \in_R Z_q$, with $z \neq 0$. All of these values are taken as the secret key. The public-key is defined to be $h = g_1^z$, $\tilde{h} = g_1^{\tilde{z}_1} g_2^{\tilde{z}_2}$, $\hat{h} = g_1^{\hat{z}_1} g_2^{\hat{z}_2}$. Thus the public key space is $PK = G^3$. We assume w.l.o.g. that $h, \tilde{h}, \hat{h}$ are all different from the identity.

To encrypt a message $m \in G$, the sender chooses $r \in_R Z_q^*$, and computes $u_1 = g_1^r$, $u_2 = g_2^r$, $e = h^r \cdot m$, $\theta = H(u_1, u_2, e)$ and $v = (\tilde{h} \cdot \hat{h}^\theta)^r$. The ciphertext is $c = (u_1, u_2, e, v)$.

The decryption algorithm is not important in our context, but we describe it anyway for completeness. On input $c = (u_1, u_2, e, v)$, the receiver computes $\theta = H(u_1, u_2, e)$, and tests if v equals $u_1^{\tilde{z}_1 + \theta \hat{z}_1} u_2^{\tilde{z}_2 + \theta \hat{z}_2}$. If equality does not hold, it outputs $\perp$; otherwise, it outputs $m = e u_1^{-z}$.

The Smooth Projective Hash Function. We define a smooth projective hashing for this encryption scheme by specifying the key space K, the projection function α, and the hash function H_k (in this case, the superset of all possible ciphertexts X can be set to $\{0,1\}^*$ and there is no limitation):

- The key space is $K = Z_q^4$, i.e. a key is a tuple (a_1, a_2, a_3, a_4), with $a_i \in_R Z_q$.
- The key projection function α is defined by

$$s_x = \alpha(a_1, a_2, a_3, a_4, u_1, u_2, e, v) = g_1^{a_1} g_2^{a_2} h^{a_3} \left(\tilde{h}\hat{h}^\theta\right)^{a_4}$$

(Notice that here α depends on θ and therefore the specific ciphertext $c = (u_1, u_2, e, v)$.)

- The hash function is defined as

$$H_k(x) = u_1^{a_1} u_2^{a_2} \left(\frac{e}{m}\right)^{a_3} v^{a_4}$$

We first show correctness. That is, we show that given s_x, a plaintext/ciphertext pair $((u_1, u_2, e, v), m)$ and the random coins r used in computing (u_1, u_2, e, v), it is possible to compute $H_k((u_1, u_2, e, v), m)$. This follows from the fact that

$$(s_x)^r = (g_1^r)^{a_1} (g_2^r)^{a_2} (h^r)^{a_3} \left(\tilde{h}\hat{h}^\theta\right)^{r \cdot a_4} = u_1^{a_1} u_2^{a_2} \left(\frac{e}{m}\right)^{a_3} v^{a_4} = H_k((u_1, u_2, e, v), m)$$

We now prove the smoothness property. Consider $x = (c, m) = ((u_1, u_2, e, v), m) \notin L$ (i.e., x such that $c = (u_1, u_2, e, v)$ is *not* a correct encryption of m under public key $(h, \tilde{h}, \hat{h})$). Let $\theta = H(u_1, u_2, e)$.

Denote with λ, λ' and $\hat{\lambda}$ the discrete logs in base g_1 of the values g_2, $(\tilde{h}\hat{h}^\theta)$ and v respectively. Also, let r_1, r_2 and r_3 be such that the ciphertext c equals $u_1 = g_1^{r_1}$, $u_2 = g_2^{r_2}$ and $e = h^{r_3} m$.

Now, the values s_x and $H_k(x)$ define two linear equations in the unknown variables a_i:

$$\log_{g_1} s_x = a_1 + \lambda a_2 + z a_3 + \lambda' a_4 \tag{9}$$

and

$$\log_{g_1} H_k(x) = r_1 a_1 + r_2 \lambda a_2 + r_3 z a_3 + \hat{\lambda} a_4 \tag{10}$$

We now prove that the above two equations are linearly independent. This then implies that given s_x, the distribution of $H_k(x)$ is uniform over G (since any value of $H_k(x)$ can be obtained with the same number of solutions of Equations (9) and (10).) We consider three cases:

1. $\lambda' = \hat{\lambda} = 0$ (i.e., $v = \tilde{h}\hat{h}^\theta = 1$):

   Since c is not a correct encryption of m, it must be that $|\{r_1, r_2, r_3\}| > 1$, i.e. the r_i's are not all equal. Recall that both $\lambda \neq 0$ (since g_2 is a generator) and $z \neq 0$ (enforced by the key generation algorithm). Thus we obtain the desired linear independence.

2. $\lambda' = 0$ but $\hat{\lambda} \neq 0$ (i.e., $\tilde{h}\hat{h}^\theta = 1$, but $v \neq 1$):

   This immediately yields the desired linear independence.

3. Both $\lambda' \neq 0$ and $\hat{\lambda} \neq 0$ (i.e., both $\tilde{h}\hat{h}^\theta \neq 1$ and $v \neq 1$):

   Then $\tilde{h}\hat{h}^\theta$ is a generator of G and we can write $v = (\tilde{h}\hat{h}^\theta)^{r_4}$, i.e. $\hat{\lambda} = r_4\lambda'$. Since c is not a correct encryption of m it must be that $|\{r_1, r_2, r_3, r_4\}| > 1$, i.e. the r_i's are not all equal. Again since z, λ, λ' are all non-zero, this yields the desired linear independence.

The KOY protocol [21]. We remark that a protocol that is almost identical to that of [21] can be described in terms of the Cramer-Shoup CCA2-encryption scheme based on the DDH assumption, combined with the above construction of a smooth projective hash function (which is thus implicit in the work of [21]).

8 Constructions Using a Relaxed Notion of Smooth Projective Hashing

We were unable to construct smooth projective hashing for encryption schemes based on assumptions other than the DDH assumption. However by relaxing the definition, we *are* able to obtain constructions based on the Quadratic Residuosity and N-Residuosity Assumptions. The resulting weaker notion is still sufficient to prove the security of our password-based key exchange protocol. The original (stronger) notion was presented earlier for the sake of clarity.

8.1 The Relaxed Notion

The relaxation here involves modifying the stronger smoothness condition as defined in Section 3.2. Informally speaking, we no longer require that the smoothness condition hold for *all* values $x \in X \backslash L$ (as required in Section 3.2). Rather, we define a subset $\Delta \subset X \backslash L$ in which the smoothness property may not hold. However, we require that it is computationally hard to find any element in Δ. This suffices because the result is that a computationally bound adversary cannot produce any "bad" elements for which the smoothness property will not hold. Formally, we say that the family $\mathcal{H}$ is a weak smooth projective hash family if the stronger smoothness condition is replaced by the following two conditions:

1. There exists a subset $\Delta \subset X \backslash L$ which is hard to sample, i.e. for all probabilistic polynomial-time Turing Machines A

$$\Pr[A(1^n, X, L) \in \Delta] < \mathsf{negl}(n)$$

Furthermore, it is easy to verify membership in Δ. That is, there exists a probabilistic polynomial-time Turing Machine T such that for every $x \in \Delta$, $\Pr[T(X, L, x) = 1] > 1 - \mathsf{negl}(|x|)$ and for every $x \notin \Delta$, $\Pr[T(X, L, x) = 1] < \mathsf{negl}(|x|)$;

2. For *every* $x' \in X \setminus (\Delta \cup L)$:

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

The definition of the random variable V appears in Section 3.1.

In other words, the strong smoothness property only holds for values $x \notin \Delta$. In particular, it may be possible to distinguish $H_k(x')$ from a random g for $x' \in \Delta$. This would be worrisome were it not for the first condition above that tells us that the probability that the adversary will find such an $x \in \Delta$ is negligible.

We now show how to adapt the main proof of security of our password-based key exchange protocol so that it works even with this weaker notion of smooth projective hashing. Later we present two examples that satisfy this weaker definition.

8.2 Adapting The Proof of Security

There are two places in which the weaker definition of smooth projective hashing requires changes into the proof. One is in the proof of Lemma 3.1 and one is in the main proof.

Adapting the proof of Lemma 3.1. The whole proof goes through unchanged except when we have to bound

$$\left|\Pr[\mathsf{Expt\text{-}Unif}_{X\setminus L}(D) = 1] - \Pr[\mathsf{Expt\text{-}Hash}_{X\setminus L}(D) = 1]\right| \leq \mathsf{negl}(n)$$

Here we need to consider the possibility that the oracle $\Omega_{X\setminus L}$ outputs an $x \in \Delta$. However, for all efficiently samplable distribution (and in particular for $D(X \setminus L)$), the weight of the set Δ must be negligible. (Otherwise, it would be easy to find $x \in \Delta$.) The rest of the proof remains unchanged.

In the specific case of our protocol (where x is defined as a pair (c, m) with c a commitment and m a message), we require that the definition of the subset Δ depends *only* on c. I.e. if $(c, m) \in \Delta$ for some m, then $(c, m') \in \Delta$ for all m'. This is required by the adaptation of the protocol proof of security as it can be seen below.

Adapting the proof of Theorem 7. We create a new hybrid experiment H_0' which is the first hybrid we use. In this experiment if the adversary ever presents a ciphertext c such that $c \in \Delta$ then we stop and count this as a success for the adversary. We note that because it is easy to verify if $c \in \Delta$, we can detect this "bad event" and halt (as required in the experiment). This is where we need the definition of Δ to depend only on c and not on m, since the message includes the password w and the simulation would not know if $(c, m) \in \Delta$ or not. Everything else remains unchanged with respect to H_0 (i.e., the real protocol). We include this clause in all the other hybrids $H_1, ..., H_5$ as well.

Clearly the above "bad" event can happen with only negligible probability. Therefore, the advantage of the adversary in H_0' can be at most negligibly more that its advantage in H_0. Now that the possibility of the adversary producing $c \in \Delta$ has been ruled out, the rest of the proof remains unchanged.

8.3 Universal Hashing

We assume that the reader is familiar with the concept of universal hashing and their application for *entropy smoothing*, which we are going to use in the following. The reader is referred to Section 3.5.1.1 in [16] for more details.

8.4 The Cramer-Shoup Scheme Based on Quadratic Residuosity

We now recall the Cramer-Shoup CCA-2 secure encryption scheme based on Quadratic Residuosity. Let $N = pq$ be the product of two safe primes, i.e. $p = 2p' + 1$ and $q = 2q' + 1$, with p', q' prime as well. Assume w.l.o.g. that $q' < p'$ and that $|q'| = |p'| = \mathsf{poly}(n)$ the security parameter[13].

With J_N we denote the subgroup of elements of Z_N^* with Jacobi symbol 1. With $QR_N \subset J_N$ we denote the subgroup of quadratic residues. The Quadratic Residuosity Assumption says that it is computationally infeasible to distinguish between the uniform distributions over J_N and QR_N.

It is not hard to see that QR_N is a cyclic group of order $N' = p'q'$. We can obtain a generator by choosing at random $\mu \in Z_N^*$ and setting $g = \mu^2 \bmod N$. This g will be a generator with overwhelming probability and its distribution is statistically close to the uniform distribution over all generators of QR_N. We can sample QR_N almost at random, by choosing $w \in [0..N/4]$ and setting $x = g^w \bmod N$. This also is a distribution over QR_N which is statistically close to uniform.

A preliminary Lemma. Before describing the scheme we prove a technical Lemma that will be useful later in the construction of the smooth projective hash function for it. The Lemma basically says that, if factoring is hard, it is computationally infeasible to find elements of "low order" in Z_N^*.

Lemma 8.1 *Let $N = pq$, with $p = 2p' + 1$, $q = 2q' + 1$ and p, q, p', q' all distinct primes. Let g be a generator of QR_N and let $h = g^z \bmod N$, with $h \neq \pm 1$. If $GCD(z, N') > 1$ then $GCD(h \pm 1, N) \neq 1, N$.*

Proof: Assume that $GCD(z, N') = p'$ (the argument is the same if $GCD(z, N') = q'$). Then $h^{q'} = g^{zq'} = 1 \bmod N$ since $N'|zq'$. This implies that $h^{q'} = 1 \bmod p$. If $h = \pm 1 \bmod p$ then $GCD(h \pm 1, N) = p$ as desired (the GCD is not N since $h \neq \pm 1 \bmod N$). If $h \neq \pm 1 \bmod p$ then q' must divide the order of the group Z_p^*, i.e. we have that $q'|(p-1) = 2p'$, a contradiction. ■

We now describe the scheme, assuming that the message space is $\{0,1\}^t$. We assume we have a universal one-way hash function [23] H that maps inputs to $\{0,1\}^n$.

Key Generation: Choose $\mu \in_R Z_N^*$ and set $g = \mu^2 \bmod N$. Randomly choose

$$z_1, \ldots, z_t, \tilde{z}_1, \ldots, \tilde{z}_n, \hat{z}_1, \ldots, \hat{z}_{2n-1} \in [0..N/2]$$

which constitute the secret key. Compute

$$h_i = g^{z_i} \;\; \tilde{h}_i = g^{\tilde{z}_i} \;\; \hat{h}_i = g^{\hat{z}_i}$$

which (together with g and N) is the public key. We assume w.l.o.g. that $h_i \neq 1$ and that $GCD(z_i, N') \neq p', q'$ since both cases happen with negligible probability.

Encryption: To encrypt a message $m = m_1 ... m_t$ where $m_i \in \{0,1\}$, one does the following. Choose r randomly in $[0..N/4]$ and compute $u = g^r \bmod N$. Then compute

$$e_i = (-1)^{m_i} h_i^r \in J_N \text{ for } i = 1...t$$

and denote $e = [e_1 \ldots e_t]$. Compute $\theta = H(u, e) = \theta_1 ... \theta_n$ with $\theta_i \in \{0,1\}$. Finally compute

$$v_i = w_i^r \quad \text{for } i = 1...n$$

[13] The length of the primes should be such that the probability of factoring N in polynomial-time should be negligible in n

where $w_i = \tilde{h}_i \prod_{j=1}^{n} \hat{h}_{i+j-1}^{\theta_j}$. Denote with $v = [v_1 \dots v_n]$. The ciphertext is (u, e, v).

Decryption: Once again we are not interested in how to decrypt but we report it anyway for completeness. On input (u, e, v), the receiver checks that it is of the right format (including testing that $u \in J_N$). Then computes $\theta = H(u, e)$ and verifies that for all $i = 1, \dots, n$

$$v_i = u^{\tilde{z}_i + \sum_{j=1}^{n} \hat{z}_{i+j-1}\theta_j}$$

If the test fails, outputs "?" otherwise compute $\tilde{m}_i = e_i u^{-z_i}$. If for all $i = 1, \dots, t$, $\tilde{m}_i = (-1)^{m_i}$ for some $m_i \in \{0, 1\}$, then output $m = m_1 ... m_t$, otherwise output "?".

A smooth projective hash function. We define a smooth projective hashing for this encryption scheme, by specifying the sets X, L, K, the projection function α, and the hash function H_k. In order to define the set C (that must be an efficiently recognizable superset of all possible ciphertexts; see Section 3.4), we first define the notion of a proper ciphertext: A ciphertext $c = (u, e, v)$ is called proper if u,e_i and v_j (for all i, j) have Jacobi symbol equal to 1. Also consider the n values w_i defined as above, $w_i = \tilde{h}_i \prod_{j=1}^{n} \hat{h}_{i+j-1}^{\theta_j}$. We require that when $w_i = 1$, then the corresponding $v_i = 1$ as well (otherwise the ciphertext is clearly not valid). We now define $X = \{(c, m)\}$, where c is any proper ciphertext and m is any message (note that an invalid ciphertext may still be proper). Observe that proper ciphertexts can be easily recognized, as required. As defined in Section 3.4, the language L is the subset of X where c is a correct encryption of m with public-key pk. The hash function is defined as $H_k : X \longrightarrow \{0, 1\}^n$.

- The key space is $K = [0..2N']^{2n(n+t+1)}$, i.e. a key is an n-tuple $(k_1, \dots, k_{2n})$. For clarity of presentation we drop the indices on the keys, but the reader should keep in mind that each key is selected independently at random. Each component of the key is of the form $k = (a, a_1, \dots, a_t, \tilde{a}_1, \dots, \tilde{a}_n)$, with each item in $Z_{2N'}$. However this space is not efficiently samplable, so we replace it with a good approximation $K' = [0..N/2]^{2n(n+t+1)}$. It's not hard to see that the uniform distribution over K' is statistically close to the uniform distribution over K; The description of the family also includes a hash function UH randomly chosen from a 2-universal family. $UH : J_N^{2n} \longrightarrow \{0, 1\}^n$.
- Given an input $x = (c, m)$ where $pk = (N, g, h_1, \dots, h_t, \tilde{h}_1, \dots, \tilde{h}_n, \hat{h}_1, \dots, \hat{h}_{2n-1})$, $c = (u, e, v)$ as above, and $m \in \{0, 1\}^t$, the generalized projection is the vector $[s_{x,1}, \dots, s_{x,2n}]$. Each $s_{x,i}$ is computed using only the component k_i of the key, i.e., $s_{x,i} = \alpha(k_i, x)$.

  Again dropping the indices on the keys:

  $$s_x = g^a \prod_{i=1}^{t} h_i^{a_i} \prod_{i=1}^{n} w_i^{\tilde{a}_i}$$

  where $w_i = \tilde{h}_i \prod_{j=1}^{n} \hat{h}_{i+j-1}^{\theta_j}$ and $\theta = H(u, e)$;
- The hash function is defined as $H_k(x) = UH[f_{k_1}(x), \dots, f_{k_{2n}}(x)]$ where (again dropping the indices on the keys):

  $$f_k(x) = u^a \prod_{i=1}^{t} \left(\frac{e_i}{(-1)^{m_i}}\right)^{a_i} \prod_{i=1}^{n} v_i^{\tilde{a}_i}$$

  If $x \in L$, i.e. c is a correct encryption of m, under public key pk above, then one can compute $f_{k_i}(x) = s_{x,i}^r$ where r is the randomness used to construct c. Therefore, it is possible to compute $H_k(x)$ given only s_x and a witness w for x.

We need to prove the smoothness property. Consider $x = (c, m) \in X \backslash L$; i.e., $c = (u, e, v)$ is a *proper* ciphertext but is *not* a correct encryption of m, under public key pk as above. Then let $\theta = H(u, e)$ and define the w_i's as above. Denote with λ_i the discrete log in base g of w_i (i.e. $\lambda_i = \tilde{z}_i + \sum_{j=1}^{n} \hat{z}_{i+j-1}\theta_j$).

We are going to prove that H_k is a *weak* smooth projective hash. Thus we need to define a set Δ of ciphertexts for which the smoothness property may not hold, but such that finding an element in Δ is infeasible. We say that a ciphertext is in Δ if there exist a λ_j such that $GCD(\lambda_j, N') \neq 1, N'$. Lemma 8.1 shows that producing a ciphertext with this property is equivalent to factoring N. It also shows how to easily recognize such elements (just test if $GCD(w_i \pm 1, N)$ is a non-trivial factor of N). From now on, we assume to be outside of Δ.

By assumption then, if $w_i \neq 1$ then it is a generator of QR_N so we can write the ciphertext as: $u = (-1)^b g^r$, $e_i = h_i^{r_i}(-1)^{m_i+b_i}$ and $v_i = (-1)^{\tilde{b}_i} w_i^{\tilde{r}_i}$, for those indices such that $w_i \neq 1$.

To prove smoothness we are going to work independently on each key k_i so we drop the index relative to the key for clarity. Consider now the equation in the $a, a_i, \tilde{a}_j$'s defined by the projection s_x:

$$\log_g s_x = a + \sum_{i=1}^{t} z_i a_i + \sum_{i=1}^{n} \lambda_i \tilde{a}_i \mod N' \tag{11}$$

Let us distinguish two cases:

- $1 \in \{b, b_1, \ldots, b_t, \tilde{b}_1, \ldots, \tilde{b}_n\}$. Then we have that

  $$f_k(x) = (-1)^{ab + \sum_{i=1}^{t} a_i b_i + \sum_{i=1}^{n} \tilde{a}_i \tilde{b}_i} g^{\sigma} \tag{12}$$

  for some σ. Let $[a, a_1, \ldots, a_t, \tilde{a}_1, \ldots, \tilde{a}_n]$ be a solution of Equation (11), mod N'. Consider the set of keys obtained as

  $$A_{a,a_i,\tilde{a}_i} = [a + dN', a_1 + d_1 N', \ldots, a_t + d_t N', \tilde{a}_1 + \tilde{d}_1 N', \ldots, \tilde{a}_n + \tilde{d}_n N']$$

  with $d, d_i, \tilde{d}_i \in \{0, 1\}$. Given s_x, the key k is uniformly distributed in the union of the $A_{a,a_i,\tilde{a}_i}$ over all $[a, a_1, \ldots, a_t, \tilde{a}_1, \ldots, \tilde{a}_n]$ which are solutions of Eq.(11). Let's focus on one specific set $A_{a,a_i,\tilde{a}_i}$. It's not hard to see that for half of the keys in the set one gets $f_k(x) = g^{\sigma}$, while one gets $f_k(x) = -g^{\sigma}$ with the other half. Thus the value $f_x(x)$ can be guessed with probability at most $1/2$.

- $b = b_1 = \ldots = b_t = \tilde{b}_1 = \ldots = \tilde{b}_n = 0$. Then $f_k(x)$ is a quadratic residue and thus its discrete log with respect to g defines another equation in the a's:

  $$\log_g f_k(x) = ra + \sum_{i=1}^{t} r_i z_i a_i + \sum_{i=1}^{n} \tilde{r}_i \lambda_i \tilde{a}_i \tag{13}$$

  Notice that the last term of the above equation should be $\sum_{i=1}^{n} (\log_g v_i)\tilde{a}_i$. But since we are only considering proper ciphertexts, we have that if $\lambda_i = 0$ then $\log_g v_i = 0$ so the two terms are equivalent.

  We want to prove that Equations (11) and (13) are "linearly independent", i.e. the $2 \times (n + t + 1)$ matrix of the coefficients of these equations in the $a, a_i, \tilde{a}_j$, has at least one 2×2 minor whose determinant is non-zero mod N'.

  Since c is not a correct encryption of m then we must have that there exist an index i (resp. j) such that $r \neq r_i$ (resp. $r \neq \tilde{r}_j$). Then the corresponding minor has determinant $z_i(r - r_i)$

(resp. $\lambda_j(r - \tilde{r}_j)$). Since z_i (resp. λ_j) is co-prime with N' (recall that we are outside of Δ), we have that this minor has non-zero determinant.

If this determinant is invertible mod N', then it is easy to see that $H_k(x)$ is uniformly distributed over QR_N given s_x.

Assume now that *all* 2×2 minors of the above form have non-invertible determinant, then it must be that for all indices i, j $r_i - r$ and $\tilde{r}_j - r$ are multiples of either p' or q'. Assume w.l.o.g. that there exists an index i such that $r_i = r + \gamma p'$ and an index j such that $r_j = r + \delta q'$ (the argument is the same if the one or both of the indices correspond to one of the $\tilde{r}_i$'s). Then the corresponding minor has determinant $z_i z_j(r_i - r_j)$ which is invertible mod N'. Thus again $H_k(x)$ is uniformly distributed over QR_N given s_x.

Finally we are left with the case in which for all indices i, j we have that $r_i = r + \gamma_i p'$ and $\tilde{r}_j = r + \tilde{\gamma}_j p'$ (again w.l.o.g. since the argument is the same for q'). This means that Equations (11) and (13) are linearly dependent mod p' but they must be linearly independent mod q' (recall that for at least one index i or j, $r \neq r_i$ or $r \neq \tilde{r}_j$ mod N'). Thus $\log_g H_k(x)$ is uniformly distributed mod q'.

In any case we can bound the probability of guessing $H_k(x)$ given s_x, with $1/q'$.

Since $q' > 2$, the whole vector $[f_{k_1}(x), \ldots, f_{k_{2n}}(x)]$ can be guessed with probability at most 2^{-2n}. Applying the properties of 2-universal hash functions we have that the distribution of $H_k(x)$ is $2^{-n/3}$ close to uniform.

Remark: In the proof of smoothness we could have included the case in which $r_i - r \neq 0$ mod N' but is a multiple of p' or q', in the "forbidden" cases defined by the set Δ. Indeed in this case too we could show that we could use such a ciphertext to factor. But to get the value g^{r-r_i} we would need to use the secret key of the encryption scheme (since we are not given g^{r_i} but only $h_i^{r_i}$). The two approaches are basically equivalent.

8.5 The Cramer-Shoup scheme based on N-residuosity

The last example is for the Cramer-Shoup CCA2-secure encryption scheme based on the hardness of deciding N-residuosity in $Z^*_{N^2}$ [10]. This assumption was originally introduced by Paillier in [24]. We describe a small variation of the original Cramer-Shoup scheme, which can easily be proven secure using the same techniques used in [10].

Let $N = pq$ be the product of two safe primes, i.e. $p = 2p' + 1$ and $q = 2q' + 1$, with p', q' prime as well. Let $N' = p'q'$. Assume w.lo.g. that $q' < p'$ and that $|q'| = |p'| = \mathsf{poly}(n) > 2n$ where n is the security parameter.

Consider the group $Z^*_{N^2}$; its order is $4NN'$. Let us consider the subgroup J_N of $Z^*_{N^2}$ which contains all the elements whose Jacobi symbol with respect to N is 1. It is not hard to see that J_N is cyclic, has order $2NN'$ and can be written as the direct product of three cyclic subgroup $J_N = G \cdot G_1 \cdot G_2$ where G is the subgroup of J_N which contains all the $(2N)$-residues. Clearly G has order N'. On the other hand G_1 is a group of order N and G_2 is the group generated by (-1). Denote $G' = G \cdot G_2$. See [10] for details.

A generator g for G' can be found by selecting $\mu \in_R Z^*_{N^2}$ and setting $g = -\mu^{2N}$. It is not hard to see that this results in a generator with overwhelming probability, and that the distribution is statistically close to uniform over all generators of G'. Clearly g^2 will then be a generator for G.

The N-residuosity assumption says that it's hard to distinguish between a random element of $Z^*_{N^2}$ and a random N-residue mod N^2. The following encryption scheme is CCA2-secure under this assumption.

A preliminary Lemma. Before describing the scheme we prove a technical Lemma that will be useful later in the construction of the smooth projective hash function for it. Again the Lemma states that, if factoring is hard, it is computationally infeasible to find elements of "low" order in the group G.

Lemma 8.2 *Let $N = pq$, with $p = 2p' + 1$, $q = 2q' + 1$ and p, q, p', q' all distinct primes. Let g' be a generator of G as above and let $h = (g')^z \bmod N$, with $h \neq \pm 1$. If $GCD(z, N') > 1$ then $GCD(h \pm 1, N) > 1$.*

The proof is identical to the one of Lemma 8.1

Key Generation. Randomly choose the secret key $z, \tilde{z}, \hat{z} \in [0..N^2/2]$ and publish the public key $h = g^z$, $\tilde{h} = g^{\tilde{z}}$ and $\hat{h} = g^{\hat{z}}$. We assume w.l.o.g that $h \neq 1$ and $GCD(\tilde{z}, N') = 1$ since the opposite happens only with negligible probability. The public key also includes a universal one-way hash function [23] H which maps inputs to Z_N.

Encryption. To encrypt a message $m \in Z_N$, choose $r \in_R [0..N/4]$ and compute $u = g^r$, $e = (1 + mN)h^r$ and $v = (\tilde{h}\hat{h}^\theta)^r$, where $\theta = H(u, e)$.

Decryption. We describe the decryption mechanism for completeness. Given a ciphertext (u, e, v) the receiver computes $\theta = H(u, e)$ and checks if $v = u^{\tilde{z}+\theta\hat{z}}$. If the test fails it outputs "?", otherwise let $\tilde{m} = eu^{-z}$. If $\tilde{m} = (1 + mN)$ for some m, output m, otherwise output "?".

A smooth projective hash function. We define a smooth projective hashing for this encryption scheme, by specifying the sets X, L, K, the projection function α, and the hash function H_k. In order to define the set C (that must be an efficiently recognizable superset of all possible ciphertexts; see Section 3.4), we first define the notion of a proper ciphertext: A ciphertext $c = (u, e, v)$ is called proper if u, e, v all have Jacobi symbol equal to 1, with respect to N. Also given $\theta = H(u, e)$, we require that if $\tilde{h}\hat{h}^\theta = 1$ then $v = 1$ as well. We now define $X = \{(c, m)\}$, where c is any proper ciphertext and m is any message. Observe that proper ciphertexts can be easily recognized, as required. As defined in Section 3.4, the language L is the subset of X where c is an encryption of m with public-key pk.

The key space is $[0..2NN']^3$ i.e. the key for a hash function is $k = (a_1, a_2, a_3)$ such that $a_i \in_R [0..2NN']$. However this space is not efficiently samplable so we replace it with $[0..N^2/2]$. The uniform distribution over $[0..N^2/2]$ is statistically close to the uniform one over $[0..2NN']$. The description of the family also includes a hash function UH randomly chosen from a 2-universal family. $UH : J_N \longrightarrow \{0, 1\}^n$.

Given an input $x = (c, m) = ((u, e, v), m)$ the projection is defined as

$$s_x = \alpha(k, x) = g^{2a_1} h^{2a_2} (\tilde{h}\hat{h}^\theta)^{2a_3}$$

where $(g, h, \tilde{h}, \hat{h})$ constitutes the public key and $\theta = H(u, e)$.

Given an input $x = (c, m) = ((u, e, v), m)$ the hash function is defined as $H_k(x) = UH[f_k(x)]$ where

$$f_k(x) = u^{2a_1} \left(\frac{e}{1 + mN} \right)^{2a_2} v^{2a_3}$$

Notice that if c is a proper encryption of m under key pk, then $f_k(x) = s_x^r$, where r is the randomness used to construct c. Thus, it is possible to compute $H_k(x)$ given only the projection and the witness, as required.

We need to prove the smoothness property. Consider $x = (c, m) \in X \backslash L$; i.e., $c = (u, e, v)$ is a *proper* ciphertext, but is *not* a correct encryption of m using the public key $(g, h, \tilde{h}, \hat{h})$. Let $\theta = H(u, e)$ and consider $\lambda = \tilde{z} + \theta\hat{z} \bmod N'$. Notice that we can write $(\tilde{h}\hat{h}^\theta)^2 = g^{2\lambda}$.

We are going to prove that H_k is a *weak* smooth projective hash. Thus we need to define a set Δ of commitments for which the smoothness property may not hold, but such that finding an element in Δ is infeasible. We say that a commitment is in Δ if $GCD(\lambda, N') \neq 1, N'$. Lemma 8.2 shows that producing a commitment with this property is equivalent to factoring N. It also shows how to easily recognize such elements (just test if $GCD((\tilde{h}\hat{h}^\theta)^2 \pm 1, N)$ is a non-trivial factor of N). From now on, we assume to be outside of Δ.

By assumption, if $(\tilde{h}\hat{h}^\theta)^2$ is different than 1, it is also a generator for G. Notice that g^2 and h^2 are also generators of G, thus we can write the commitment as $u = (-1)^{b_1}\gamma_1(g^2)^{r_1}$, $e = (-1)^{b_2}\gamma_2(h^2)^{r_2}(1+mN)$ and $v = (-1)^{b_3}\gamma_3[(\tilde{h}\hat{h}^\theta)^2]^{r_3}$ where $\gamma_i \in G_1$. Notice that in the computation of $H_k(x)$ the (-1) components are irrelevant since we raise each term to the power $2a$. So we ignore them from now on.

Consider now the equation in the a_i's defined by the projection s_x. Notice that $s_x \in G$ since we are squaring each term. Therefore,

$$\log_{g^2} s_x = a_1 + za_2 + \lambda a_3 \mod N' \tag{14}$$

Let us distinguish two cases:

- There exists $\gamma_i \neq 1$. Then we have that

  $$f_k(x) = (\gamma_i^2)^{a_i} \cdot \sigma$$

  for some $\sigma \in J_N$. Let $[a_1, a_2, a_3]$ be a solution of Equation (14), mod N'. Consider the set of keys obtained as

  $$A_{a_1,a_2,a_3} = [a_1 + d_1N', a_2 + d_2N', a_3 + d_3N']$$

  for $d_i \in [0..2N-1]$. Given s_x the key k is uniformly distributed in the union over all $[a_1, a_2, a_3]$ which are solutions of Eq.(14) of the sets A_{a_1,a_2,a_3}. Let's focus on one specific set A_{a_1,a_2,a_3}. Recall that $\gamma_i \in G_1$ and $\gamma_i \neq 1$ so its order is either p, q, N. Since 2 is co-prime with these values, the order of γ_i^2 is the same as the one of γ_i. And since $GCD(N, N') = 1$ we have that keys of the form $a_i + d_iN$ map $(\gamma_i^2)^{a_i+d_iN}$ uniformly over the group generated by γ_i.

  In conclusion, given s_x the value $(\gamma_i^2)^{a_i}$ and consequently $f_k(x)$, can be guessed with probability at most $1/q$.

- $\gamma_i = 1$ for all $i = 1, 2, 3$. Then $f_k(x)$ is an element of G and thus its discrete log with respect to g^2 defines another equation in the a_i's (recall that $GCD(2, N') = 1$ thus 2^{-1} is well defined mod N'):

  $$2^{-1}\log_{g^2} f_k(x) = r_1a_1 + r_2za_3 + r_3\lambda a_3 \mod N' \tag{15}$$

  We want to prove that Equations (14) and (15) are "linearly independent", i.e. the 2×3 matrix of the coefficients of these equations in the a_i's has at least one 2×2 minor whose determinant is non-zero mod N'.

  Since c is not a correct commitment to m then we must have that either $r_2 \neq r_1$ or $r_3 \neq r_1$. Then the corresponding minors have determinant $z(r_2 - r_1)$ and $\lambda(r_3 - r_2)$. Since z and λ are co-prime with N' (recall that we are outside of Δ), we have that at least one of these minors has non-zero determinant.

  If this determinant is invertible mod N', then it is easy to see that $f_k(x)$ is uniformly distributed over G given s_x.

Assume now that both determinants are non-invertible. Then it must be that both $r_2 - r_1$ and $r_3 - r_1$ are multiples of either p' or q'. Assume w.l.o.g. that $r_2 = r_1 + \gamma_2 p'$ and $r_3 = r_1 + \gamma_3 q'$ (the argument is the same if you switch p' and q'). Then the corresponding minor has determinant $z\lambda(r_3 - r_2)$ which is invertible mod N'. Thus again $f_k(x)$ is uniformly distributed over G given s_x.

Finally we are left with the case in which $r_2 = r_1 + \gamma_2 p'$ and $r_3 = r_1 + \gamma_3 p'$ (again w.l.o.g. since the argument is the same for q'). This means that Equations (14) and (15) are linearly dependent mod p' but they must be linearly independent mod q' (recall that either $r_2 \neq r_1$ or $r_3 \neq r_1$ mod N'). Thus $\log_g f_k(x)$ is uniformly distributed mod q'.

In any case we can bound the probability of guessing $H_k(x)$ given s_x, with $1/q'$.

In conclusion we have that given s_x the value $f_k(x)$ can be guessed with probability at most $1/q' < 2^{-2n}$ by choice of the security parameter. Applying the properties of 2-universal hash functions we have that the distribution of $H_k(x)$ is $2^{-n/3}$ close to uniform over $\{0,1\}^n$.

8.6 A Note on Efficiency

The following table summarizes the efficiency of the three schemes we have presented. For each scheme (DDH, Quadratic Residuosity, and N-Residuosity) we present the cost of each operation (Encryption, computation of the projection function and computation of the hash function).

There are two numbers on each box. The first is the number of "full" exponentiations required (i.e. if the modulus is 1024 bits, these exponentiations take a 1024-bit exponent). The second is the number of "small" exponentiations (in typical applications the exponent in this case will be 160-bit).

For the Quadratic Residuosity case, the value t is the length of the message, while n is the security parameter.

	DDH	Q-Residuosity	N-Residuosity
Encryption	$0, 5$	$t + n + 1, 0$	$3, 1$
Projection	$0, 5$	$2n(t + n + 1), 0$	$3, 1$
Hash	$0, 4$	$2n(n + t + 1), 0$	$3, 0$

It should be clear that the Quadratic Residuosity based scheme is the least efficient of the schemes. The DDH and N-Residuosity schemes are comparable in cost, except that all the exponentiations in the DDH case are performed with small exponents, which in practice makes a considerable difference.

Acknowledgements

We are grateful to both Jonathan Katz and Victor Shoup for answering our questions about their respective papers [21] and [10]. We also thank Jonathan for some helpful comments on the presentation.

References

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

[2] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *1st Conf. on Computer and Communications Security*, ACM, pages 62–73, 1993.

[3] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *CRYPTO'93*, Springer-Verlag (LNCS 773), pages 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*, pages 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*, pages 244–250, 1993.

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

[7] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. In 30*th STOC*, pages 209–218, 1998.

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

[9] R. Cramer and V.Shoup. A practical public-key cryptosystem secure against adaptive chosen ciphertexts attacks. In *CRYPTO'98*, Springer-Verlag (LNCS 1462), pages 13–25, 1998.

[10] 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), pages 45–64, 2002.

[11] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Non-Interactive and Non-Malleable Commitment. In 30*th STOC,* pages 141–150, 1998.

[12] G. Di Crescenzo, J. Katz, R. Ostrovsky and A. Smith. Efficient and Non-interactive Non-malleable Commitment. In *Eurocrypt 2001*, Springer-Verlag (LNCS 2045), pages 40–59, 2001.

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

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

[15] C. Dwork. *The non-malleability lectures.* Course notes for CS 359, Stanford University, Spring 1999. Available at: `theory.stanford.edu/∼gdurf/cs359-s99`.

[16] O. Goldreich. *Foundations of Cryptography – Basic Tools.* Cambridge University Press. 2001.

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

[18] S. Halevi and H. Krawczyk. Public-Key Cryptography and Password Protocols. In *ACM Conference on Computer and Communications Security*, 1998.

[19] D.P. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Computer Communication Review*, 26(5):5–26, 1996.

[20] J. Katz. *Efficient Cryptographic Protocols Preventing "Man-in-the-Middle" Attacks.* Ph.D. Thesis, Columbia University, 2002.

[21] 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.

[22] S. Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Proceedings of the Workshop on Security Protocols*, Ecole Normale Superieure, 1997.

[23] M. Naor and M. Yung. Universal One-Way Hash Functions and their Cryptographic Applications. In 21*st STOC*, pages 33–43, 1989.

[24] P. Paillier. Public-Key Cryptosystems based on Composite Degree Residue Classes. In *EUROCRYPT'99*, Springer-Verlag (LNCS 1592), pages 223–228, 1999.

[25] S. Patel. Number theoretic attacks on secure password schemes. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 236–247, 1997.

[26] M. Steiner, G. Tsudik and M. Waidner. Refinement and extension of encrypted key exchange. *ACM SIGOPS Oper. Syst. Rev.*, 29(3):22–30, 1995.

[27] T. Wu. The secure remote password protocol. In *1998 Internet Society Symposium on Network and Distributed System Security*, pp.97–111, 1998.