IPAKE: Isomorphisms for Password-based Authenticated Key Exchange

Dario Catalano[1], David Pointcheval[1], and Thomas Pornin[2]

[1] CNRS–LIENS, Ecole Normale Supérieure, Paris, France
{Dario.Catalano,David.Pointcheval}@ens.fr.
[2] Cryptolog, Paris, France
Thomas.Pornin@cryptolog.com.

Abstract. In this paper we revisit one of the most popular password-based key exchange protocols, namely the OKE (for Open Key Exchange) scheme, proposed by Luck in 1997. Our results can be highlighted as follows. First we define a new primitive that we call *trapdoor hard-to-invert isomorphisms*, and give some candidates. Then we present a generic password-based key exchange construction, that admits a security proof assuming that these objects exist. Finally, we instantiate our general scheme with some concrete examples, such as the Diffie-Hellman function and the RSA function, but more interestingly the modular square root function, which leads to the first scheme with security related to the integer factorization problem. Furthermore, the latter variant is very efficient for one party (the server). Our results hold in the random-oracle model.

1 Introduction

Shortly after the introduction of the revolutionary concept of asymmetric cryptography, proposed in the seminal paper by Diffie and Hellman [9], people realized that properly managing keys is not a trivial task. In particular private keys tend to be pretty large objects, that have to be safely stored in order to preserve whatever kind of security. Specific devices have thus been developed in order to help human beings in storing their secrets, but it is clear that even the most technologically advanced device may become useless if lost or stolen. In principle the best way to store a secret is to keep it in mind. In practice, however, human beings are very bad at remembering large secrets (even if they are passwords or pass-phrases) and very often they need to write passwords down on a piece of paper in order to be able to keep track of them. As a consequence, either one uses a short (and memorable) password, or writes/stores it somewhere. In the latter case, security eventually relies on the mode of storage (which is often the weakest part in the system: a human-controlled storage). In the former case, a short password is subject to exhaustive search.

Indeed, by using a short password, one cannot prevent a brute force on-line exhaustive search attack: the adversary just tries some passwords of its own choice in order to try to impersonate a party. If it guesses the correct password, it can get in, otherwise it has to try with another password. In many applications, however, the number of such active attacks can be limited in various ways. For example one may impose some delay between different trials, or even closing the account after some fixed number of consecutive failures. Of course the specific limitations depend very much on the context – other kind of attacks, such as Denial of Service ones, for example, should be made hard to mount either. In any case, the important point we want to make here is that the impact of on-line exhaustive search can be limited. However on-line attacks are not the only possible threats to the security of a password-based system. Imagine for example an adversary who has access to several transcripts of communication between a server and a client. Clearly the transcript of a "real" communication somehow depends on the actual password. This means that a valid transcript (or several ones) could be used to "test" the validity of some password: the adversary chooses a random password and simply checks if the produced transcript is the same as the received one. In this way it is possible to mount an (off-line) exhaustive search attack that can be much more effective than the on-line one, simply because, in this scenario, the adversary can try all the possible passwords just until it finds the correct one. Such an off-line exhaustive search is usually called "dictionary attack".

1.1 Related Work

A password-based key exchange is an interactive protocol between two parties A and B, who initially share a short password pw, that allows A and B to exchange a session key sk. One expects from this key to be semantically secure w.r.t. any party, but A and B who should know it at the end of the protocol. The study of password-based protocols resistant to dictionary attacks started with the seminal work of Bellovin and Merritt [3], where they proposed the so-called *Encrypted Key Exchange* protocol (EKE). The basic idea of their solution is the following: A generates a public key and sends it to B encrypted – using a symmetric encryption scheme – with the common password. B uses the password to decrypt the received ciphertext. Then it proceeds by encrypting some value k using the obtained public key. The resulting ciphertext is then re-encrypted (once again using the password) and finally sent to A. Now A can easily recover k, using both his own private key and the common password. A shared session key is then derived from k using standard techniques.

A classical way to break password-based schemes is the partition attack [4]. The basic idea is that if the cleartexts encrypted with the password have any redundancy, or lie in a strict subset, a dictionary attack can be successfully mounted: considering one flow (obtained by eavesdropping) one first chooses a password, decrypts the ciphertext and checks whether the redundancy is present or not (or whether the plaintext lies in the correct range.) This technique allows to quickly select probable passwords, and eventually extract the correct one.

The partition attack can be mounted on many implementations of EKE, essentially because a public key usually contains important "redundancy" (as a matter of fact a public key – or at least its encoding – is not in general a random-looking string). Note that in the described approach (for EKE), the same symmetric encryption (using the same password) is used to encrypt both the public key, and the ciphertext generated with this key. This may create additional problems basically because these two objects (i.e. the public key and the ciphertext) are very often defined on completely unrelated sets. A nice exception to this general rule are ElGamal keys [12]. This is thus the sole effective application of EKE.

As noticed by the original authors [3], and emphasized by Lucks [17], it is "*counter-intuitive (...) to use a secret key to encrypt a public key*". For this reason Lucks [17] proposed OKE, (which stands for Open Key Exchange). The underlying idea of this solution is to send the public key in clear and to encrypt the second flow only. Adopting this new approach, additional public-key encryption schemes can be considered (and in particular RSA [23] for instance). However, one has to be careful when using RSA. The problem is that the RSA function is guaranteed to be a permutation only if the user behaves honestly and chooses his public key correctly. In real life, however, a malicious user may decide to generate keys that do not lead to a permutation at all. In such a case a partition attack becomes possible: an RSA-ciphertext would lie in a strict subset if $\mathbb{Z}_n^\star$. For this reason Lucks proposed a variant of his scheme, known as *Protected OKE*, to properly deal with the case of RSA. Later on, however, MacKenzie *et al.* [19, 18] proved that the scheme was flawed by presenting a way to attack it. At the same time they showed how to repair the original solution by proposing a new protocol they called SNAPI (for Secure Network Authentication with Password Identification), for which they provided a full proof of security in the random-oracle model. This proof, however, is specific to RSA, in the random-oracle model, and very intricate.

Interestingly enough, in the standard model, the problem of secure password-based protocols was not treated rigorously until very recently. The first rigorous treatment of the problem was proposed by Halevi and Krawczyk [15] who, however, proposed a solution that requires other setup assumptions on top of that of the human password. Later on, Goldreich and Lindell [14] proposed a very elegant solution that achieves security without any additional setup assumption. The Goldreich and Lindell proposal is based on sole existence of trapdoor permutations and, even though very appealing from a theoretical point of view, is definitely not practical. The first practical solution was proposed by Katz, Ostrovsky and Yung [16]. Their solution is based on the Decisional Diffie-Hellman assumption and assumes that all parties have access to a set of public parameters (which is of course a stronger set-up assumption than assuming that only human passwords are shared, but still a weaker one with respect to the Halevi-Krawczyk ones for example). Even more recently Gennaro and Lindell [13] presented an abstraction of the Katz, Ostrovsky and Yung [16] protocol that allowed them to construct a general framework for authenticated password-based key exchange in the common reference string model.

We note here that even though from a mathematical point of view a proof in the standard model is always preferable to a proof in the random-oracle model, all the constructions in the standard model presented so far are *way* less efficient with respect to those known in the random-oracle model. It is true that a proof in the random-oracle model should be interpreted with care, more as a heuristic proof than a real one. On the other hand in many applications efficiency is a big issue and it may be preferable to have a very efficient protocol with a heuristic proof of security than a much less efficient one with a complete proof of security.

1.2 Our Contributions

In this paper, we revisit the generic OKE construction by clearly stating the requirements about the primitive to be used: we need a family of isomorphisms with some specific computational properties that we call *trapdoor hard-to-invert isomorphisms* (see next section for a formal definition for these objects). Very roughly a trapdoor hard-to-invert isomorphism, can be seen as an isomorphic function that is in general hard to invert, unless some additional information (the trapdoor) is provided. Note that such an object is different with respect to traditional trapdoor functions. A trapdoor one-way function is always easy to compute, whereas a trapdoor hard-to-invert function may be not only hard to invert, but – at least in some cases – also hard to compute [10]. As it will become apparent in the next sections, this requirement is not strong because basically all the classical public-key encryption schemes fit it (RSA [23], Rabin with Blum moduli [22], ElGamal [12], and even the recent Okamoto-Uchiyama's [20] and Paillier's schemes [21]). More precisely our results can be described as follows.

First, after having described our security model, we present a very general construction – denoted **IPAKE** for *Isomorphism for Password-based Authenticated Key Exchange* – and we prove it is secure. Our security result relies on the computational properties of the chosen trapdoor hard-to-invert isomorphism family, in the random-oracle model. As a second result we pass instantiating the general construction with specific encryption schemes. We indeed show that trapdoor hard-to-invert isomorphisms can be based on the Diffie-Hellman problem, on the RSA problem, and even on integer factoring.

For lack of space, we refer to the full version [8] for the two first applications, since they are not really new. Plugging ElGamal directly leads to one of the AuthA variants, proposed to IEEE P1363 [2], or to PAK [5]. The security has already been studied in several ideal models [5–7]. The case of RSA leads to a scheme similar to RSA-OKE, SNAPI [19, 18], or to the scheme proposed by Zhu *et al.* [26].

More interestingly using such methods we can construct a very efficient solution from the Rabin function. To our knowledge this is the first efficient password-based authenticated key exchange scheme based on factoring.

2 Preliminaries

Denote with $\mathbb{N}$ the set of natural numbers and with $\mathbb{R}^+$ the set of positive real numbers. We say that a function $\varepsilon : \mathbb{N} \to \mathbb{R}^+$ is *negligible* if and only if for every polynomial $P(n)$ there exists an $n_0 \in \mathbb{N}$ such that for all $n > n_0$, $\varepsilon(n) \leq 1/P(n)$.

If A is a set, then $a \leftarrow A$ indicates the process of selecting a at random and uniformly over A (which in particular assumes that A can be sampled efficiently).

2.1 Trapdoor Hard-to-Invert Isomorphisms

Let I be a set of indices. Informally a family of *trapdoor hard-to-invert isomorphisms* is a set $F = \{f_m : X_m \rightarrow Y_m\}_{m \in I}$ satisfying the following conditions:

1. one can easily generate an index m, which provides a description of the function f_m – a morphism –, its domain X_m and range Y_m (which are assumed to be isomorphic groups), and a trapdoor t_m;
2. for a given m, one can efficiently sample pairs $(x, f_m(x))$, with x uniformly distributed in X_m;
3. for a given m, one can efficiently decide Y_m;
4. given the trapdoor t_m, one can efficiently invert $f_m(x)$, and thus recover x;
5. without the trapdoor, inverting f_m is hard.

This is almost the same definition as for trapdoor one-way permutations with homomorphic properties. There is a crucial difference however: one can sample pairs, but may not necessarily be able to compute $f_m(x)$ for a given x (point 2 above). As a consequence, the function is hard-to-invert, but it may be hard to compute as well.

More formally we say that F defined as above is a family of *trapdoor hard-to-invert isomorphisms* if the following conditions hold:

1 – There exist a polynomial p and a probabilistic polynomial time Turing Machine Gen which on input 1^k (where k is a security parameter) outputs pairs (m, t_m) where m is uniformly distributed in I and $|t_m| < p(k)$. The index m defines X_m and Y_m, which are isomorphic groups, an isomorphism f_m from X_m onto Y_m and a set R_m of values uniformly samplable, which will be used to sample $(x, f_m(x))$ pairs. The information t_m is referred as the *trapdoor*.

2.1 – There exists a polynomial time Turing Machine Sample^x which on input $m \in I$ and $r \in R_m$ outputs $x \in X_m$. Furthermore, for any m, the machine $\mathsf{Sample}^x(m, \cdot)$ implements a bijection from R_m onto X_m.

2.2 – There exists a polynomial time Turing Machine Sample^y, such that on input $m \in I$ and $r \in R_m$ it outputs $f_m(x)$ for $x = \mathsf{Sample}^x(m, r)$. Therefore, $\mathsf{Sample}^y(m, r) = f_m(\mathsf{Sample}^x(m, r))$.

3 – There exists a polynomial time Turing Machine Check^y which, on input $m \in I$ and any y, answers whether $y \in Y_m$ or not.

4 – There exists a (deterministic) polynomial time Turing Machine Inv such that $\mathsf{Inv}(m, t_m, f_m(x)) = x$, for all $x \in X_m$ and for all $m \in I$.

5 – For every probabilistic polynomial time Turing Machine $\mathcal{A}$ we have that, for large enough k,

$$\Pr[m \stackrel{R}{\leftarrow} I \,;\, x \stackrel{R}{\leftarrow} X_m \,;\, y = f_m(x) : \mathcal{A}(m, y) = x] \leq \varepsilon(k),$$

where $\varepsilon(\cdot)$ is a negligible function.

The last property is our formal *hard-to-invert* notion, which is quite similar to the usual *one-way* notion: they just differ if $\mathsf{Sample}^x(m, \cdot)$ is one-way.

2.2 Verifiable Sub-Family of Trapdoor Hard-to-Invert Isomorphisms

In the above definition, it is clear that for any $m \in I$, the function f_m is an isomorphism from the group X_m onto Y_m. However, in practice, the family of functions $\{f_m\}_m$ maybe indexed by a potentially larger set S (i.e. $I \subseteq S$), for which there may exist some indices that do not lead to an isomorphism. Therefore, we require more properties to be satisfied.

- there exists a large subset $I \subseteq S$, such that $F = \{f_m : X_m \to Y_m\}_{m \in I}$ is a family of *trapdoor hard-to-invert isomorphisms*;
- there exists a set J, of indices which provide an isomorphism – such that $I \subseteq J \subseteq S$ –, which admits an efficient zero-knowledge proof of membership.

The last property turns out to be crucial for the application we have in mind. In our setting the client has to choose the specific function to use in the protocol. This means that a dishonest client (i.e. one that does not share a password with the server) could propose an index whose corresponding function is not an isomorphism. This would give him the ability to run a partition attack (as already explained for RSA). For this reason we require the client to produce a function f together with a proof that it is actually an isomorphism.

2.3 Zero-Knowledge Proofs of Membership

As noticed above, the only property we want to be able to verify is the isomorphic one, and thus the fact that the index m actually lies in J: we just want the adversary not to be able to prove a wrong statement, we do not care about malleability [11]. One second point is that the zero-knowledge property will be required in the security proof: a valid index m is given, one tries to use the adversary to solve a hard problem related to m. Thus, we need to be able to provide a proof of validity of m, without any witness. Note however that the simulation is performed for valid statements only, and thus *simulation soundness* [24] is not required. Moreover, since we just have to simulate one proof without the witness (other executions will be performed as in an actual execution) *concurrent zero-knowledge* is not needed either.

For efficiency reasons, we will focus on a specific class of zero-knowledge proofs: for a given statement m, the verifier sends a random seed seed and then the prover non-interactively provides a proof $p = \mathsf{Prove}^m(m, w, \mathsf{seed})$ using a witness w that $m \in J$, w.r.t. the random seed seed; the proof can be checked without the witness $\mathsf{Check}^m(m, \mathsf{seed}, p)$. In our protocol, honest players will sample $m \in I$, and thus together the trapdoor t_m. This trapdoor will generally be a good witness. More formally we require:

- Completeness – Prove^m and Check^m are two efficient (polynomial time) algorithms, and for any $m \in J$ and any challenge seed, a witness helps to build a proof $p = \mathsf{Prove}^m(m, w, \mathsf{seed})$ which is always accepted: $\mathsf{Check}^m(m, \mathsf{seed}, p)$ accepts;

- Soundness – for any $m \notin J$, the probability for any adversary (on its random tape and the random seed seed) to forge a valid proof (accepted by the Check^m algorithm) is negligible within time t: $\mathsf{Succ}^{\mathsf{forge}}(t)$ will denote the maximal success probability for any adversary within time t;
- ROM-simulatability – granted the programmability of the random oracle, for any $m \in I$ and any seed, there exists an efficient way to perfectly simulate an accepted proof.

2.4 Concrete Examples

The Diffie-Hellman Family. The most natural example of family of trapdoor hard-to-invert isomorphisms is the Diffie-Hellman one. The machine Gen, on input the security parameter k, does as follows. First it chooses a random prime q of size k, and a prime p such that q divides $p-1$. Next, it chooses a subgroup $\mathbb{G}$ of order q in $\mathbb{Z}_p^\star$ and a corresponding generator g. Finally it chooses a random element a in $\mathbb{Z}_q$, it sets $h = g^a \bmod p$ and outputs the pair (m, t_m) where $t_m = a$ and m is an encoding of (g, p, q, h). This defines our set I.

Now f_m is instantiated as follows. Set $X_m = Y_m = \mathbb{G}\backslash\{1\}$, $R_m = \mathbb{Z}_q$ and $\mathsf{Sample}^x : \mathbb{Z}_q \to \mathbb{G}$ is defined[3] as $\mathsf{Sample}^x(x) = g^x \bmod p$. Moreover f_m is defined as (for any $X \in \mathbb{G}\backslash\{1\}$): $f_m(X) = X^a \bmod p$.

Clearly, to efficiently evaluate f_m on a random point X, one should know either the trapdoor information a or any x such that $\mathsf{Sample}^x(x) = X$ (assuming, of course, that the computational Diffie-Hellman problem is infeasible in $\mathbb{G}$): $\mathsf{Sample}^y(x) = h^x$. Similarly knowledge of the trapdoor is sufficient to invert f_m on a random point Y: $\mathsf{Inv}(a, Y) = Y^{1/a}$. However inverting the function without knowing the trapdoor seems to be infeasible. Nevertheless, $Y_m = \mathbb{G}$ is efficiently decidable: $\mathsf{Check}^y(y)$ simply checks whether $y^q = 1 \bmod p$ or not.

For our functions to be isomorphisms, one just needs a to be co-prime with q, where q is actually the order of g. For better efficiency, the group informations (g, p, q) can be fixed, and considered as common trusted parameters. Therefore, Gen just chooses a and sets $h = g^a \bmod p$: one just needs to check that $h \neq 1 \bmod p$ and $h^q = 1 \bmod p$, no witness is required, nor additional proof: Prove^m does not need any witness for outputting any proof, since Check^m simply checks the above equality/inequality.

The RSA Family. Another natural example is the RSA permutation. In this case the machine Gen on input the security parameter k does as follows. First it chooses two random primes p, q of size $k/2$ and sets $n = pq$. Next, it chooses a public exponent e such that $\gcd(e, \varphi(n)) = 1$. Finally it outputs the pair (m, t_m) where $t_m = (p, q)$ and m is an encoding of (n, e). This defines our set I.

The function f_m is instantiated as follows. Set $X_m = Y_m = R_m = \mathbb{Z}_n^\star$, and $\mathsf{Sample}^x : \mathbb{Z}_n^\star \to \mathbb{Z}_n^\star$ is the identity function, i.e. $\mathsf{Sample}^x(x) = x$. The function

[3] Note that we allow a slight misuse of notation here. Actually the function Sample^x should be defined as $\mathsf{Sample}^x : I \times \mathbb{Z}_q \to \mathbb{G}$. However we prefer to adopt a simpler (and somehow incorrect) notation for visual comfort.

f_m is defined as (for any $x \in \mathbb{Z}_n^\star$): $f_m(x) = x^e \bmod n$. Hence, $\mathsf{Sample}^y(x) = x^e \bmod n$. The Inv algorithm is straightforward, granted the trapdoor. And the Check^y algorithm simply has to check whether the element is prime to n.

As already noticed, since Sample^x is easy to invert, the RSA family is not only a trapdoor hard-to-invert isomorphism family, but also a trapdoor one-way permutation family. However, to actually be an isomorphism, (n, e) does not really need to lie in I, which would be very costly to prove (while still possible). It just needs to satisfy $\gcd(e, \varphi(n)) = 1$, which defines our set J. An efficient proof of validity is provided in the full version [8], where both Prove^m and Check^m are formally defined.

The Squaring Family. As a final example, we suggest the squaring function which is defined as the RSA function with the variant that $e = 2$. A problem here arises from the fact that squaring is not a permutation over $\mathbb{Z}_n^\star$, simply because 2 is not co-prime with $\varphi(n)$. However, if one considers *Blum* moduli (i.e. composites of the form $n = pq$, where $p \equiv q \equiv 3 \bmod 4$) then it is easy to check that the squaring function becomes an automorphism onto the group of quadratic residues modulo n (in the following we refer to this group as to Q_n.) However this is not enough for our purposes. An additional difficulty comes from the fact that we need an efficient way to check if a given element belongs to Y_m (which would be Q_n here): the need of an efficient algorithm Check^y. The most natural extension of Q_n is the subset J_n of $\mathbb{Z}_n^\star$, which contains all the elements with Jacobi symbol equal to $+1$. Note that for a Blum modulus $n = pq$, this set is isomorphic to $\{-1, +1\} \times Q_n$ (this is because -1 has a Jacobi symbol equal to $+1$, but is not a square). By these positions we get the *signed squaring*[4] isomorphism:

$$\begin{aligned} f_n : \{-1, +1\} \times Q_n &\to J_n \\ (b\ ,\ x) &\mapsto b \times x^2 \bmod n. \end{aligned}$$

For this family, the machine Gen, on input the security parameter k, does as follows. First it chooses two random Blum primes p, q of size $k/2$ and sets $n = pq$. Then it outputs the pair (m, t_m) where $t_m = (p, q)$ and m is an encoding of n. This thus defines our set I. The function f_m is instantiated as follows. Set $X_m = R_m = \{-1, +1\} \times Q_n$, $Y_m = J_n$ and $\mathsf{Sample}^x : \{-1, +1\} \times Q_n \to \{-1, +1\} \times Q_n$ is the identity function, i.e. $\mathsf{Sample}^x(b, x) = (b, x)$. The function f_m is defined as (for any $(b, x) \in \{-1, +1\} \times Q_n$): $f_m(b, x) = b \times x^2 \bmod n$. Hence, $\mathsf{Sample}^y(b, x) = f_m(b, x)$. The Inv algorithm is straightforward, granted the trapdoor. And the Check^y algorithm simply computes the Jacobi symbol.

As above, since Sample^x is easy to invert, the squaring family is not only a trapdoor hard-to-invert isomorphism family, but also a trapdoor one-way permutation family. However, to actually be an isomorphism, n does not really need to be a Blum modulus, which would be very costly to prove. What we need is just that -1 has Jacobi symbol $+1$ and any square in $\mathbb{Z}_n^\star$ admits exactly 4 roots. A validity proof is provided, with the mathematical justification, in the section 6, which thus formally defines both Prove^m and Check^m.

[4] By *signed*, we mean that the output of the function has a sign (plus or minus).

3 The Formal Model

3.1 Security Model

Players. We denote by A and B two parties that can participate in the key exchange protocol P. Each of them may have several *instances* called oracles involved in distinct, possibly concurrent, executions of P. We denote A (resp. B) instances by A^i (resp. B^j), or by U when we consider any user instance. The two parties share a low-entropy secret pw which is drawn from a small dictionary $\mathsf{Password}$, according to a distribution $\mathcal{D}$. In the following, we use the notation $\mathcal{D}(n)$ for the probability to be in the most probable set of n passwords:

$$\mathcal{D}(n) = \max_{P \subseteq \mathsf{Password}} \left\{ \Pr_{pw \overset{R}{\leftarrow} \mathcal{D}} [pw \in P \mid \mathtt{Card}(P) \leq n] \right\}.$$

If we denote by $\mathcal{U}_N$ the uniform distribution among N passwords, $\mathcal{U}_N(n) = n/N$.

Queries. We use the security model introduced by Bellare *et al.* [1], to which paper we refer for more details. In this model, the adversary $\mathcal{A}$ has the entire control of the network, which is formalized by allowing $\mathcal{A}$ to ask the following queries:

- $\mathsf{Execute}(A^i, B^j)$: This query models passive attacks, where the adversary gets access to honest executions of P between the instances A^i and B^j by eavesdropping.
- $\mathsf{Reveal}(U)$: This query models the misuse of the session key by any instance U (use of a weak encryption scheme, leakage after use, etc). The query is only available to $\mathcal{A}$ if the attacked instance actually "holds" a session key and it releases the latter to $\mathcal{A}$.
- $\mathsf{Send}(U, m)$: This query models $\mathcal{A}$ sending a message to instance U. The adversary $\mathcal{A}$ gets back the response U generates in processing the message m according to the protocol P. A query $\mathsf{Send}(A^i, \mathtt{Start})$ initializes the key exchange algorithm, and thus the adversary receives the flow A should send out to B.

In the active scenario, the $\mathsf{Execute}$-query may seem rather useless: after all the Send-query already gives the adversary the ability to carry out honest executions of P among parties. However, even in the active scenario, $\mathsf{Execute}$-queries are essential to properly deal with dictionary attacks. Actually the number q_s of Send-queries directly asked by the adversary *does not* take into account the number of $\mathsf{Execute}$-queries. Therefore, q_s represents the number of flows the adversary may have built by itself, and thus the number of passwords it may have tried. Even better, $q_a + q_b$ is an upper-bound on the number of passwords it may have tried, where q_a (and q_b resp.) is the number of A (B resp.) instances involved in the attack. For the sake of simplicity, we restricted queries to A and B only. One can indeed easily extend the model, and the proof, to the more general case, keeping in mind that we are interested in the security of executions involving at least A or B, with the password pw shared by them. Additional queries would indeed use distinct passwords, which could be assumed public in the security analysis (known to our simulator).

3.2 Security Notions

Two main security notions have been defined for key exchange protocols. The first one is the semantic security of the key, which means that the exchanged key is unknown to anybody else than the players. The second one is unilateral or mutual authentication, which means that either one, or both, of the participants actually know the key.

AKE Security. The semantic security of the session key is modeled by an additional query $\mathsf{Test}(U)$. The Test-query can be asked at most once by the adversary $\mathcal{A}$ and is only available to $\mathcal{A}$ if the attacked instance U is *Fresh*. The freshness notion captures the intuitive fact that a session key is not "obviously" known to the adversary. An instance is said to be *Fresh* if the instance has accepted (i.e. the flag accept is set to true) and neither it nor its partner (i.e. the other instance with same session tag —or SID— which is defined as the view the player has of the protocol —the flows— before it accepts) have been asked for a Reveal-query. The Test-query is answered as follows: one flips a (private) coin b and forwards sk (the value $\mathsf{Reveal}(U)$ would output) if $b = 1$, or a random value if $b = 0$.

We denote the **AKE advantage** as the probability that $\mathcal{A}$ correctly guesses the value of b. More precisely we define $\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}) = 2\Pr[b = b'] - 1$, where the probability space is over the password, all the random coins of the adversary and all the oracles, and b is the output guess of $\mathcal{A}$ for the bit b involved in the Test-query. The protocol P is said to be (t, ε)-**AKE-secure** if $\mathcal{A}$'s advantage is smaller than ε for any adversary $\mathcal{A}$ running with time t.

Entity Authentication. Another goal of the adversary is to impersonate a party. We may consider unilateral authentication of either A (A-Auth) or B (B-Auth), thus we denote by $\mathsf{Succ}_P^{\mathsf{A-auth}}(\mathcal{A})$ (resp. $\mathsf{Succ}_P^{\mathsf{B-auth}}(\mathcal{A})$) the probability that $\mathcal{A}$ successfully impersonates an A instance (resp. a B instance) in an execution of P, which means that B (resp. A) terminates (i.e. the terminate flag is set to true) even though it does not actually share the key with any accepting partner A (resp. B).

A protocol P is said to be (t, ε)-**Auth-secure** if $\mathcal{A}$'s success for breaking either A-Auth or B-Auth is smaller than ε for any adversary $\mathcal{A}$ running with time t. This protocol then provides *mutual authentication*.

4 Algorithmic Assumptions

In this section we state some algorithmic assumptions we need in order to construct an **IPAKE** protocol. As already sketched in section 1.2, our basic building block is a family of trapdoor hard-to-invert bijections $\mathcal{F}$. More precisely each bijection $f \in \mathcal{F}$ needs to be a group isomorphism from a group $(X_f, \oplus_f)$ into a group $(Y_f, \otimes_f)$, where $\ominus_f$ (resp. $\oslash_f$) is the inverse operation of $\oplus_f$ (resp. $\otimes_f$)[5]. As additional assumption we require the existence of a generalized full-domain hash function $\mathcal{G}$, which on a new input (f, q), outputs a uniformly distributed element in Y_f. This is the reason why we need the decidability of Y_f: in practice, $\mathcal{G}$ will be implemented by iterating a hash function until the output is in Y_f.

The non-invertibility of the functions in the family $\mathcal{F}$ is measured by the "ability", for any adversary $\mathcal{A}$, in inverting a random function (in $\mathcal{F}$) on a random point, uniformly drawn from Y_f:

$$\mathsf{Succ}^{\mathsf{NI}}_{\mathcal{F}}(\mathcal{A}) = \Pr[f \stackrel{R}{\leftarrow} \mathcal{F}, x \stackrel{R}{\leftarrow} X_f : \mathcal{A}(f, f(x)) = x].$$

More precisely, we denote by $\mathsf{Succ}^{\mathsf{NI}}_{\mathcal{F}}(t)$ the maximal success probability for all the adversaries running within time t. A simpler task for the adversary may be to output a list of n elements which contains the solutions:

$$\mathsf{SuccInSet}^{\mathsf{NI}}_{\mathcal{F}}(\mathcal{A}) = \Pr[f \stackrel{R}{\leftarrow} \mathcal{F}, x \stackrel{R}{\leftarrow} X_f, S \leftarrow \mathcal{A}(f, f(x)) : x \in S].$$

As above, we denote by $\mathsf{SuccInSet}^{\mathsf{NI}}_{\mathcal{F}}(n, t)$ the maximal success probability for all the adversaries running within time t, which output sets of size n.

4.1 The RSA Family: $\mathcal{F} = \mathsf{RSA}$

As described in section 2.4 the function f is defined by n and e, $Y_f = X_f = \mathbb{Z}_n^\star$. And, for any $x \in \mathbb{Z}_n^\star$, $f(x) = x^e \bmod n$. For a correctly generated n and a valid e (i.e an e such that $\gcd(\varphi(n), e) = 1$) the non-invertibility of the function is equivalent to the, widely conjectured, one-wayness of RSA. This leads to the following

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{RSA}}(t + nT_{\mathsf{exp}}) = \mathsf{Succ}^{\mathsf{NI}}_{\mathsf{RSA}}(t + nT_{\mathsf{exp}}) \geq \mathsf{SuccInSet}^{\mathsf{NI}}_{\mathsf{RSA}}(n, t) = \mathsf{SuccInSet}^{\mathsf{ow}}_{\mathsf{RSA}}(n, t)$$

where T_{exp} is an upper-bound on the time required to perform an exponentiation.

4.2 The Diffie-Hellman Family: $\mathcal{F} = \mathsf{DH}$

Let $\mathbb{G} = \langle g \rangle$ be any cyclic group of (preferably) prime order q. As sketched in section 2.4, the function f is defined by a point $P = g^x$ in $\mathbb{G}\backslash\{1\}$ (and thus $x \neq 0 \bmod q$), and $X_f = Y_f = \mathbb{G}$. For any $Q = g^y \in \mathbb{G}$, $f(Q) = g^{xy}$.

A (t, ε)-$\mathsf{CDH}_{g,\mathbb{G}}$ attacker, in the finite cyclic group $\mathbb{G}$ of prime order q, generated by g, is a probabilistic machine Δ running in time t such that

$$\mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(\Delta) = \Pr_{x,y}[\Delta(g^x, g^y) = g^{xy}] \geq \varepsilon$$

[5] For visual comfort in the following we adopt the symbols f, X_f, Y_f rather than (respectively) f_m, X_m, Y_m.

Alice		*Bob*
	Common password pw	
$\mathsf{accept} \leftarrow \mathsf{false}$		$\mathsf{accept} \leftarrow \mathsf{false}$
		$\mathsf{terminate} \leftarrow \mathsf{false}$
$(f,t) \xleftarrow{R} \mathsf{Gen}(1^k)$	$\xrightarrow{\mathsf{Alice}, f}$	
	$\xleftarrow{\mathsf{Bob}, \mathsf{seed}}$	$\mathsf{seed} \xleftarrow{R} \{0,1\}^k$
$p \leftarrow \mathsf{Prove}^m(f,t,\mathsf{seed})$	$\xrightarrow{p}$	$\mathsf{Check}^m(f,\mathsf{seed},p)$?
		$r \xleftarrow{R} R_f$
		$x \leftarrow \mathsf{Sample}^x(f,r)$
		$y \leftarrow \mathsf{Sample}^y(f,r)$
$\mathsf{PW} \leftarrow \mathcal{G}(f,pw)$		$\mathsf{PW} \leftarrow \mathcal{G}(f,pw)$
	$\xleftarrow{\hat{y}}$	$\hat{y} \leftarrow y \otimes_f \mathsf{PW}$
$y' \leftarrow \hat{y} \oslash_f \mathsf{PW}, x' \leftarrow \mathsf{Inv}(f,t,y')$	$\xrightarrow{\mathsf{Auth}}$	
		Auth valid? $\Rightarrow \mathsf{accept} \leftarrow \mathsf{true}$
$\mathsf{accept} \leftarrow \mathsf{true}$		$\mathsf{terminate} \leftarrow \mathsf{true}$

Fig. 1. An execution of the **IPAKE** protocol: Auth is computed by Alice (Bob resp.) as $\mathcal{H}_1(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|x)$ ($\mathcal{H}_1(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|x')$ resp.), and sk is computed by Alice (Bob resp.) as $\mathcal{H}_0(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|x)$ ($\mathcal{H}_0(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|x')$ resp.)

where the probability is taken over the random values x and y in $\mathbb{Z}_q$. As usual, we denote by $\mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(t)$ the maximal success probability over every adversary running within time t. Then, when g and $\mathbb{G}$ are fixed, $\mathsf{Succ}^{\mathsf{NI}}_{\mathsf{DH}}(t) = \mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(t)$. Using Shoup's result [25] about "self-correcting Diffie-Hellman", one can see that if $\mathsf{SuccInSet}^{\mathsf{NI}}_{\mathsf{DH}}(n,t) \geq \varepsilon$, then $\mathsf{Succ}^{\mathsf{NI}}_{\mathsf{DH}}(t') \geq 1/2$ for some $t' \leq 6/\varepsilon \times (T + nT_{\mathsf{exp}})$.

4.3 The Squaring Family: $\mathcal{F} = \mathsf{Rabin}$

As discussed in section 2.4 if one assumes that the modulus n is the product of two Blum primes, the signed squaring function f becomes an isomorphism from $\{-1,+1\} \times Q_n$ onto J_n. Furthermore, for a correctly generated n the non-invertibility of f is trivially equivalent to the one-wayness of factoring Blum composites. This leads us to the following inequality

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{Rabin}}(t + nT_{\mathsf{exp}}) = \mathsf{Succ}^{\mathsf{NI}}_{\mathsf{Rabin}}(t + nT_{\mathsf{exp}}) \geq \mathsf{SuccInSet}^{\mathsf{ow}}_{\mathsf{Rabin}}(n,t),$$

which provides a very tight bound because, in this case, T_{exp} represents the time required to perform a single modular multiplication (i.e. to square).

5 Security Proof for the IPAKE Protocol

5.1 Description and Notations

In this section we show that the **IPAKE** protocol distributes session keys that are semantically secure and provides unilateral authentication for the client A.

The specification of the protocol can be found on Figure 1. Some remarks, about notation, are in order

- We assume $\mathcal{F}$ to be a correct family, with a verifiable sub-family of trapdoor hard-to-invert isomorphisms f from X_f into Y_f. In the following, we identify m to f_m, and thus f. We denote by s the size of I. Furthermore, we denote by q a lower bound on the size of any Y_f.
- For this choice of parameters for the family $\mathcal{F}$, we can define the function $\mathcal{G}$ which is assumed to behave like a generalized full-domain random oracle. In particular we model $\mathcal{G}$ as follows: on input a couple (f, q) it outputs a random element, uniformly distributed in Y_f.

Since we only consider unilateral authentication (of A to B), we just introduce a $\mathsf{terminate}$ flag for B.

5.2 Security Proof

Theorem 1 (AKE/UA Security). *Let us consider the protocol* **IPAKE**, *over a family $\mathcal{F}$ of trapdoor hard-to-invert isomorphisms, with parameter (s, q), where* $\mathsf{Password}$ *is a dictionary equipped with the distribution $\mathcal{D}$. For any adversary $\mathcal{A}$ within a time bound t, with less than q_s active interactions with the parties (*Send*-queries) and q_p passive eavesdroppings (*$\mathsf{Execute}$*-queries), and asking q_g and q_h hash queries to $\mathcal{G}$ and any $\mathcal{H}_i$ respectively:* $\mathsf{Adv}^{\mathsf{ake}}_{\mathsf{ipake}}(\mathcal{A}) \le 4\varepsilon$ *and* $\mathsf{Adv}^{\mathsf{A-auth}}_{\mathsf{ipake}}(\mathcal{A}) \le \varepsilon$, *with ε upper-bounded by*

$$3\mathcal{D}(q_a + q_b) + 6q_a \mathsf{SuccInSet}^{\mathsf{NI}}_{\mathcal{F}}(q_h^2, t + 2q_h^2 \tau_{law}) + q_b \mathsf{Succ}^{\mathsf{forge}}(t) + \frac{q_b}{2^{\ell_1}} + \frac{Q^2}{2q} + \frac{Q_P^2}{2s},$$

where q_a and q_b denote the number of A and B instances involved during the attack (each upper-bounded by $q_p + q_s$), $Q \le q_g + q_h + 2q_p + q_s$ and Q_P denotes the number of involved instances ($Q_P \le 2q_p + q_s$), and τ_{law} is the time needed for evaluating one law operation. Let us remind that ℓ_1 is the output length of $\mathcal{H}_1$ (the authenticator.)

For lack of space, we refer to the full version [8] for the full proof, here we justify the main terms in the security result.

Ideally, when one considers a password-based authenticated key exchange, one would like to prove that the two above success/advantage are upper-bounded by $\mathcal{D}(q_a + q_b)$, plus some negligible terms. For technical reasons in the proof (to get a *clear* proof) we have a small additional constant factor. This main term is indeed the basic attack one cannot avoid: the adversary guesses a password and makes an on-line trial. Other ways for it to break the protocol are:

- use a function f that is not a permutation, and in particular not a surjection. With the view of $\hat{y}$, the adversary tries all the passwords, and only a strict fraction leads to y in the image of f: this is a partition attack. But for that, it has to forge a proof of validity for f. Hence the term $q_b \times \mathsf{Succ}^{\mathsf{forge}}(t)$;

– use the authenticator Auth to check the correct password. But this requires the ability to compute $f^{-1}(\mathsf{PW})$. Hence the term $q_a \times \mathsf{SuccInSet}^{\mathsf{NI}}_{\mathcal{F}}(\cdot,\cdot)$.
– send a correct authenticator Auth, but being lucky. Hence the term $q_b/2^{\ell_1}$.

Additional negligible terms come from very unlikely collisions. All the remaining kinds of attacks need some information about the password.

6 A Concrete Example: The SQRT-IPAKE Protocol

An important contribution of this work (at least from a practical point of view) is the first efficient and provably secure password-based key exchange protocol based on factoring. The formal protocol appears in Figure 2. Here we describe the details of this specific implementation.

<u>*Alice*</u>		<u>*Bob*</u>
	Shared password: pw	
$\mathsf{accept} \leftarrow \mathsf{false}$		$\mathsf{accept} \leftarrow \mathsf{false}$
		$\mathsf{terminate} \leftarrow \mathsf{false}$
$p_1, p_2 \in BlumPrimes(k/2)$		
$n \leftarrow p_1 p_2$	$\xrightarrow{\mathsf{Alice}, n}$	
	$\xleftarrow{\mathsf{Bob}, \mathsf{seed}}$	$\mathsf{seed} \overset{R}{\leftarrow} \{0,1\}^k$
$p \leftarrow \mathsf{Prove}^m(n, (p_1, p_2), \mathsf{seed})$	$\xrightarrow{p}$	$\mathsf{Check}^m(n, \mathsf{seed}, p)$?
		$z \overset{R}{\leftarrow} \mathbb{Z}_n^\star, x \leftarrow z^2 \bmod n$
		$b \overset{R}{\leftarrow} \{0,1\}, y \leftarrow (-1)^b x^2 \bmod n$
$\mathsf{PW} \leftarrow \mathcal{G}(n, pw)$		$\mathsf{PW} \leftarrow \mathcal{G}(n, pw)$
	$\xleftarrow{\hat{y}}$	$\hat{y} \leftarrow y \times \mathsf{PW} \bmod n$
$y' \leftarrow \hat{y} \times \mathsf{PW}^{-1} \bmod n$		
$x' = \mathsf{SQRT}(y') \bmod n$	$\xrightarrow{\mathsf{Auth}}$	
		Auth valid? $\Rightarrow \mathsf{accept} \leftarrow \mathsf{true}$
$\mathsf{accept} \leftarrow \mathsf{true}$		$\mathsf{terminate} \leftarrow \mathsf{true}$
	$sk = \mathcal{H}_0(\mathsf{Alice} \| \mathsf{Bob} \| n \| \hat{y} \| pw \| x)$	
	$\mathsf{Auth} = \mathcal{H}_1(\mathsf{Alice} \| \mathsf{Bob} \| n \| \hat{y} \| pw \| x)$	

Fig. 2. SQRT – **IPAKE** protocol

6.1 Description of the SQRT-IPAKE Protocol

In order for the protocol to be correct we need to make sure that the adopted function is actually an isomorphism. As seen in section 2.4 this is the case if one assumes that the modulus n is the product of two Blum primes, and $f_n : \{-1,+1\} \times Q_n \to J_n$ is the signed squaring function.

We thus set $X_f = \{-1, +1\} \times Q_n$ and $Y_f = J_n$, and, of course, the internal law is the multiplication in the group $\mathbb{Z}_n^\star$. In order for the password PW to be generated correctly, we need a $\mathcal{G}(n, \cdot)$ hash function onto J_n. Constructing such a function is pretty easy: we start from a hash function onto $\{0, 1\}^k$, and we iterate it until we get an output in J_n. The details of this technique are deferred to the full version of this paper [8]. Here we stress that if $n \geq 646$ then very few iterations are sufficient. As already noticed, we require Alice to prove the following about the modulus n, so that the function is actually an isomorphism:

- The modulus n is in the correct range ($n \geq 646$);
- The Jacobi symbol of -1 is $+1$ in $\mathbb{Z}_n^\star$ (this is to make sure that f_n is actually a morphism);
- The signed squaring function is actually an isomorphism from $\{-1, +1\} \times Q_n$ onto J_n (this is to make sure that any square in $\mathbb{Z}_n^\star$ has exactly 4 roots).

Proving the first two statements is trivial. For the third one we need some new machinery.

6.2 Proof of Correct Modulus.

With the following theorem (whose proof can be found in the full version of this paper [8]) we show that if n is a composite modulus (with at least two different prime factors) then the proposed function is an isomorphism.

Theorem 2. *Let n be a composite modulus containing at least two different prime factors and such that -1 has Jacobi symbol $+1$ in $\mathbb{Z}_n^\star$. Moreover let f_n be the morphism defined above. The following facts are true*

1. *If f_n is surjective then it is an isomorphism.*
2. *If f_n is not surjective, then at most half of the elements in J_n have a pre-image.*

The theorem above leads to the protocol `Prove-Surjective` (see Figure 3). The basic idea of this protocol is that we prove that our function is a bijection by proving it is surjective. Soundness follows from the second statement. However, in order to fall into the hypotheses of the theorem, we need to make sure n is actually a composite modulus of the required form (i.e. with at least two distinct prime factors). We achieve this with the `Prove-Composite` protocol (see Figure 3). The correctness (completeness, soundness and zero-knowledge properties) of these protocols is deferred to the full version of this paper [8].

Remark 3. We point out that our protocol is very efficient, for the verifier, in terms of modular multiplications. It is also possible for Alice to use the same modulus for different sessions.

Acknowledgments

We thank the anonymous referees for their fruitful comments.

Protocol `Prove-Composite`	**Protocol** `Prove-Surjective`
$\mathcal{H}_2(n,\cdot,\cdot)$ and $\mathcal{H}_4(n,\cdot,\cdot)$ are full-domain hash functions onto J_n $\mathcal{H}_3$ ($\mathcal{H}_5$ resp.) is a random oracle onto $\{0,1\}^k$ ($\{0,1\}^\ell$ resp.) Bob chooses a random seed seed and sends it to Alice	
For $i \leftarrow 1$ to ℓ, Alice	
1. Sets $y_i = \mathcal{H}_2(n, \mathsf{seed}, i) \in J_n$ 2. Computes $(\beta_i, \alpha_{i,0}, \alpha_{i,1}, \alpha_{i,2}, \alpha_{i,3})$ such that – $\alpha_{i,0} = -\alpha_{i,1} \bmod n$ – $\alpha_{i,2} = -\alpha_{i,3} \bmod n$ – $\alpha_{i,j}^2 = y_i\beta_i \bmod n$ $(j = 0, \ldots, 3)$, where $\beta_i \in \{-1,+1\}$ 3. Sets $h_{i,j} = \mathcal{H}_3(n, \alpha_{i,j})$ $(j = 0, \ldots, 3)$ One defines $c_1 \ldots, c_\ell = \mathcal{H}_5(n, \mathsf{seed}, \{h_{i,j}\})$	1. Sets $z_i = \mathcal{H}_4(n, \mathsf{seed}, i) \in J_n$ 2. Computes $(b_i, x_i) = f^{-1}(z_i)$ such that $(b_i, x_i) \in \{-1,+1\} \times Q_n$ 3. Computes a value γ_i such that $\gamma_i^2 = x_i \bmod n$ (this is to make sure that x_i is actually in Q_n);
Alice answers with, for $i = 1, \ldots, \ell$,	
$(\beta_i, \alpha_{i,2c_i}, \alpha_{i,2c_i+1})$	(γ_i, b_i)
Bob checks that, for each $i = 1, \ldots, \ell$,	
1. the $h_{i,j}$, for $j = 0, \ldots, 3$, are all distinct 2. $\alpha_{i,2c_i} = -\alpha_{i,2c_i+1} \bmod n$ 3. $h_{i,2c_i} = \mathcal{H}_3(n, \alpha_{i,2c_i})$ and $h_{i,2c_i+1} = \mathcal{H}_3(n, \alpha_{i,2c_i+1})$ 4. $\mathcal{H}_2(n, \mathsf{seed}, i) = \beta_i\alpha_{i,2c_i}^2 \bmod n$	$b_i\gamma_i^4 = \mathcal{H}_4(n, \mathsf{seed}, i) \bmod n$

Fig. 3. Proof of Correct Modulus

References

1. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Eurocrypt '00*, LNCS 1807, pages 139–155. Springer-Verlag, Berlin, 2000.
2. M. Bellare and P. Rogaway. The AuthA Protocol for Password-Based Authenticated Key Exchange. Contributions to IEEE P1363. March 2000.
3. S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure against Dictionary Attacks. In *Proc. of the Symposium on Security and Privacy*, pages 72–84. IEEE, 1992.
4. C. Boyd, P. Montague, and K. Nguyen. Elliptic Curve Based Password Authenticated Key Exchange Protocols. In *ACISP '01*, LNCS 2119, pages 487–501. Springer-Verlag, Berlin, 2001.
5. V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password Authenticated Key Exchange Using Diffie-Hellman. In *Eurocrypt '00*, LNCS 1807, pages 156–171. Springer-Verlag, Berlin, 2000.
   Full version available at: `http://cm.bell-labs.com/who/philmac/research/`.
6. E. Bresson, O. Chevassut, and D. Pointcheval. Security Proofs for Efficient Password-Based Key Exchange. In *Proc. of the 10th CCS*, pages 241–250. ACM Press, New York, 2003.

7. E. Bresson, O. Chevassut, and D. Pointcheval. New Security Results on Encrypted Key Exchange. In *PKC '04*, LNCS, pages 145–159. Springer-Verlag, Berlin, 2004.
8. D. Catalano, D. Pointcheval, and T. Pornin. IPAKE: Isomorphisms for Password-based Authenticated Key Exchange. In *Crypto '04*, LNCS. Springer-Verlag, Berlin, 2004. Full version available from `http://www.di.ens.fr/users/pointche/`.
9. W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT–22(6):644–654, November 1976.
10. Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong Key-Insulated Signature Schemes. In *PKC '03*, LNCS, pages 130–144. Springer-Verlag, Berlin, 2003.
11. D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.
12. T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, IT–31(4):469–472, July 1985.
13. R. Gennaro and Y. Lindell. A Framework for Password-Based Authenticated Key Exchange. In *Eurocrypt '03*, LNCS 2656, pages 524–543. Springer-Verlag, Berlin, 2003.
14. O. Goldreich and Y. Lindell. Session-Key Generation Using Human Passwords Only. In *Crypto '01*, LNCS 2139, pages 408–432. Springer-Verlag, Berlin, 2001.
15. S. Halevi and H. Krawczyk. Public-Key Cryptography and Password Protocols. In *Proc. of the 5th CCS*. ACM Press, New York, 1998.
16. J. Katz, R. Ostrovsky, and M. Yung. Efficient Password-Authenticated Key Exchange Using Human-Memorizable Passwords. In *Eurocrypt '01*, LNCS 2045, pages 475–494. Springer-Verlag, Berlin, 2001.
17. S. Lucks. Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys. In *Proc. of the Security Protocols Workshop*, LNCS 1361. Springer-Verlag, Berlin, 1997.
18. P. MacKenzie, S. Patel, and R. Swaminathan. Password-Authenticated Key Exchange Based on RSA. In *Asiacrypt '00*, LNCS 1976, pages 599–613. Springer-Verlag, Berlin, 2000.
19. P. MacKenzie and R. Swaminathan. Secure Network Authentication with Password Identification. Submission to IEEE P1363a. August 1999.
20. T. Okamoto and S. Uchiyama. A New Public Key Cryptosystem as Secure as Factoring. In *Eurocrypt '98*, LNCS 1403, pages 308–318. Springer-Verlag, Berlin, 1998.
21. P. Paillier. Public-Key Cryptosystems Based on Discrete Logarithms Residues. In *Eurocrypt '99*, LNCS 1592, pages 223–238. Springer-Verlag, Berlin, 1999.
22. M. O. Rabin. Digitalized Signatures. In R. Lipton and R. De Millo, editors, *Foundations of Secure Computation*, pages 155–166. Academic Press, New York, 1978.
23. R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.
24. A. Sahai. Non-Malleable Non-Interactive Zero-Knowledge and Chosen-Ciphertext Security. In *Proc. of the 40th FOCS*. IEEE, New York, 1999.
25. V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In *Eurocrypt '97*, LNCS 1233, pages 256–266. Springer-Verlag, Berlin, 1997.
26. F. Zhu, A. H. Chan, D. S. Wong, and R. Ye. Password Authenticated Key Exchange based on RSA for Imbalanced Wireless Network. In *Proc. of ISC '02*, LNCS 2433, pages 150–161. Springer-Verlag, Berlin, 2002.