This is the full version of the extended abstract, which appeared at CRYPTO '2004 [9].
Journal of Cryptology, Volume 20 - Number 1.
Pages 115–149. Springer-Verlag, 2007.

Trapdoor Hard-to-Invert Group Isomorphisms and Their Application to Password-based Authentication

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 the security chain the weakest link is definitely the human one: human beings cannot remember long secrets and often resort to rather insecure solutions to keep track of their passwords or pass-phrases. For this reason it is very desirable to have protocols that do not require long passwords to guarantee security, even in the case in which exhaustive search is feasible. This is actually the goal of password-based key exchange protocols, secure against off-line dictionary attacks: two people share a password (possibly a very small one, say a 4-digit number), and after the protocol execution, they end-up sharing a large secret session key (known to both of them, but nobody else). Then an adversary attacking the system should try several connections (on average 5,000 for the above short password) in order to be able to get the correct password. Such a large number of erroneous connections can be prevented by various means.
Our results can be highlighted as follows. First we define a new primitive that we call *trapdoor hard-to-invert group 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.

Keywords. Password-based key exchange, trapdoor, isomorphism.

1 Introduction

Shortly after the introduction of the revolutionary concept of asymmetric cryptography, proposed in the seminal paper by Diffie and Hellman [11], 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 any kind 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 memory. 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 also. 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 until it finds the correct one. Such an off-line exhaustive search is usually called a "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 this key to be semantically secure w.r.t. any party, except 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 [4], 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 [5]. 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 one to select probable passwords quickly, 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 [13]. This is thus the sole effective application of EKE.

For this case, indeed, if common parameters are fixed (i.e. the group $\mathbb{G} = \langle g \rangle$), one may very well have that the public key and the produced ciphertexts are elements in the same cyclic group. For this reason this is basically the unique, concrete, implementation of EKE presented so far. Formal security proofs for ElGamal based EKE have been proposed by Boyko *et al.* [6], Bellare *et al.* [1], and Bresson *et al.* [7, 8]. They stated resistance to dictionary attacks in the random-oracle model, and the ideal-cipher model too.

As noticed by the original authors [4], and emphasized by Lucks [20], it is "*counter-intuitive (...) to use a secret key to encrypt a public key*". For this reason Lucks [20] 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 [26] 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 of $\mathbb{Z}_n^\star$. For this reason Lucks proposed a variant of his scheme, known as *Protected OKE*, to deal with the case of RSA properly. Later, however, MacKenzie *et al.* [22, 21] 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.

Moreover their solution is not very efficient because of the fact that, in order to make sure that the RSA function is a permutation, they use, as public exponent e, a prime number that is larger than any possible choice of the modulus. For this reason the resulting protocol has the same prohibitive cost for each party, exactly as ElGamal-EKE or ElGamal-OKE. More recently, Zhu *et al.* [31] addressed the problem of efficiency by making possible the use of a small exponent. The resulting protocol becomes well-suited for imbalanced applications. However no complete proof of security was presented for this last scheme. Finally shortly after the publication of the first version of this paper [9], Zhang [30] proposed a method that actually improves on all of the RSA based constructions (including ours).

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 [16] who, however, proposed a solution that requires other setup assumptions on top of that of the human password. Later, Goldreich and Lindell [15] proposed a very elegant solution that achieves security without any additional setup assumption. The Goldreich and Lindell proposal is based on the 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 [17]. 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 [14] presented an abstraction of the Katz, Ostrovsky and Yung [17] 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 group isomorphisms with some specific computational properties that we call *trapdoor hard-to-invert group isomorphisms* (see the next section for a formal definition for these objects). Very roughly a trapdoor hard-to-invert group 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 [12]. 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 [26], Rabin with Blum moduli [25], ElGamal [13], and even the more recent Okamoto-Uchiyama's [23] and Paillier's schemes [24]). 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.

We postpone to Appendix D the two first applications, since they are not really new. Plugging ElGamal directly leads to one of the AuthA variants, proposed in IEEE P1363 [3], or to PAK [6]. The security has already been studied in several ideal models [6–8]. The case of RSA leads to a scheme similar to RSA-OKE, SNAPI [22, 21], or to the scheme proposed by Zhu *et al.* [31]. However, we believe that some of our techniques may be of independent interest: we present some methods to prove (efficiently) that a given function is a permutation with respect to the given public key.

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 Group Isomorphisms

Let I be a set of indices. Informally a family of *trapdoor hard-to-invert group isomorphisms* is a family $F = \{f_m : X_m \to Y_m\}_{m \in I}$ 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, efficiently uniformly samplable finite 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, meaning with this that on input m and a candidate value y one can efficiently test if $y \in Y_m$ or not[1].
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 definition looks very similar to the definition of trapdoor one-way bijective functions when the domain and the codomain of each function in the family are isomorphic groups. There is a crucial difference however: here 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.

Informally property 2 tells us that, on input an index m, one can efficiently sample elements $x \in X_m$ such that computing $f_m(x)$ is easy. We formalize this special "samplability" requirement by introducing two polynomial time Turing machines Sample^x and Sample^y. Informally Sample^x takes as input an index m and some random value (in the appropriate range) r and produces as output an element $x \in X_m$ with the following properties. (1) x is uniformly distributed in X_m and (2) computing

[1] Note that, since our constructions rely on the random oracle model, the efficient samplability property makes possible a random oracle that outputs a uniformly distributed element in Y_m, from a classical random oracle that outputs bit strings.

$f_m(x)$ can be done efficiently from r and m, applying Sample^y. Indeed, Sample^y takes as input an index m and some random value r, and produces as output the element $y = f_m(\mathsf{Sample}^x(m,r)) \in Y_m$. Therefore, y is uniformly distributed in Y_m, and $x = f_m^{-1}(y)$ can be computed efficiently from r and m, applying Sample^x.

Notice that if one can efficiently compute r from both m and $x = \mathsf{Sample}^x(m,r)$ then one can also efficiently compute $f_m(x)$ on *any* input x. On the other hand, if one assumes that computing r from both m and $x = \mathsf{Sample}^x(m,r)$ is infeasible, then the task of computing $f_m(x)$ on a random input x may be hard as well.

Formally we say that F defined as above is a family of *trapdoor hard-to-invert group 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 $I \cap \{0,1\}^k$ and $|t_m| < p(k)$. The index m contains the description of 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 uniformly and efficiently samplable values, 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].

2.2 – There exists a polynomial time Turing machine Sample^y, which on input $m \in I$ and $r \in R_m$ 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, for all $m \in I$ and for all $x \in X_m$, $\mathsf{Inv}(m, t_m, f_m(x)) = x$.

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

$$\Pr[(m,t_m) \leftarrow \mathsf{Gen}(1^k)\,;\, x \overset{R}{\leftarrow} X_m\,;\, y = f_m(x) : \mathcal{A}(m,y) = x] \leq \varepsilon(k),$$

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

In the rest of this paper, to shorten slightly the name of the primitive defined above, we refer to it as *trapdoor hard-to-invert isomorphisms* (rather than *trapdoor hard-to-invert group isomorphisms*).

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

Let I be the set of indices defined as above. In the definition given in the previous section we assumed that for any $m \in I$, the function f_m is an isomorphism from the group X_m onto the group Y_m.

In practice, however, this assumption may be somehow unrealistic. Imagine, for example, the case in which a malicious client (i.e. one that does not share any password with the server) decides to propose an index $s \notin I$, such that f_s *is not* an isomorphism between X_s and Y_s. It goes without saying that this would have catastrophic consequences as it allows the client to run a partition attack (as already explained for the case of the RSA function).

To overcome this problem we restrict our attention to a specific class of trapdoor hard-to-invert isomorphisms, that we call *verifiable*. In a nutshell a trapdoor hard-to-invert isomorphism f_s is said to be verifiable if from an index s it is possible to prove (efficiently and in zero-knowledge) that f_s is actually an isomorphism between X_s and Y_s.

[2] Having a bijection from R_m to X_m allows one to choose an element $r \in R_m$ uniformly and then get a uniformly distributed element $x \in X_m$

In our scenario, we exploit this useful feature by requiring the client to produce a function f (or an index m) together with a proof that it is actually an isomorphism (or that m actually lies in I). More precise details follow.

Let S be a samplable set of indexes, for which there may exist some indices s such that f_s is not an isomorphism between X_s and Y_s. We say that $F' = \{f_s\}_{s \in S}$ contains a verifiable subfamily of *trapdoor hard-to-invert isomorphisms* if the following conditions are met:

- There exists a subset $I \subseteq S$, such that $F = \{f_m : X_m \rightarrow Y_m\}_{m \in I}$ is a family of *trapdoor hard-to-invert isomorphisms*;
- There exists an efficient zero-knowledge proof of membership for the language I.

2.3 Zero-Knowledge Proofs of Membership

As noticed above, the only property we want to be able to verify is the isomorphic one, i.e. the fact that a given index m actually lies in I.

Standard zero-knowledge proof systems allow one to achieve this without revealing any side information. Moreover, by the soundness property, one has that a cheating client can convince the server of a false statement only with negligible probability.

In our security proof we assume that a given valid index m is given to the simulator. The goal of the simulator is then to be able to use the adversary to break some conjectured hard problem related to m (*soundness*).

For technical reasons, that will become apparent from the security proof, we also need the simulator to be able to simulate a proof of validity of m without actually knowing the corresponding witness. In other words the simulator needs to prove that the received m lies in I without actually knowing why this is the case (the witness), but knowing that this is actually the case (*zero-knowledge*).

Thus in order for our proof to go through correctly we need to assume that the employed zero-knowledge proof remains sound even if the adversary is allowed to see a simulated proof (*not* of its choosing), of a true statement. This notion of soundness is reminiscent to that of *simulation soundness* introduced by Sahai [28] for the case of non-interactive proofs. Sahai's notion assumes that the adversary gets access to a limited number of simulated proofs before attempting to produce his own (fake) one. This notion was later extended by De Santis *et al.* [10] to the unbounded case (the reader is referred to [28] and [10] for further details not discussed here).

Note however that in our case the problem is quite different: in Sahai's definition the adversary is allowed to see a bounded number of simulated proofs *of its own choice*, and thus of possibly wrong statements, before attempting to generate a false one. In our setting, on the other hand, we allow the adversary to see simulated proofs for valid statements that come from outside and on which, then, the adversary has no control. A simple argument (honest players will sample m by running the trapdoor hard-to-invert isomorphisms generation algorithm Gen on input 1^k, which also outputs the trapdoor information t_m to be used as a witness for m) shows that, for our protocols, the standard notion of soundness suffices to model the security properties we require.

Furthermore, since we just have to simulate one proof without the witness (other executions will be performed as in an actual execution, knowing the trapdoor information) *concurrent zero-knowledge* is not needed here.

One may observe that, in principle, the above reasoning does not rule out the fact that *concurrent soundness* might still be required, in a general—possibly interactive—setting. We point out, however, that concurrent soundness is not an issue in our setting. This is because, for efficiency reasons, we focus on a very specific class of proofs, which are almost entirely non-interactive. More precisely, 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 $m \in I$, w.r.t. the random seed seed; the proof can be checked without the witness by just checking if $\mathsf{Check}^m(m, \mathsf{seed}, p) = 1$.

Definition 1. Let k be a security parameter and let $p(\cdot)$ be a polynomial, we denote with $\texttt{Seeds}$ an efficiently samplable set of challenges such that $\forall\,\mathsf{seed} \in \texttt{Seeds}$ one has that $|\mathsf{seed}| < p(k)$. Moreover, denoting with R a witness relation for the language I and with Prove^m and Check^m two probabilistic algorithms whose running time is polynomial in k, we require

Completeness – For all $\mathsf{seed} \in \texttt{Seeds}$, for all $m \in I$ and all w such that $R(m, w) = 1$, we have that

$$\mathsf{Check}^m(m, \mathsf{seed}, \mathsf{Prove}^m(m, w, \mathsf{seed})) = 1.$$

Soundness – For all probabilistic polynomial time adversaries $\mathcal{A} = (A_1, A_2)$, one has that $\mathsf{Succ}^{\mathsf{forge}}(\mathcal{A}) = \Pr[\texttt{Exp}(\mathcal{A})]$ is negligible in k, where this probability is taken over the random coin tosses of $\mathcal{A}$, and where $\texttt{Exp}(\mathcal{A})$ is defined as follows:

```
$\texttt{Exp}(\mathcal{A})$ :
    $(m, state) \leftarrow A_1(1^k)$
    $\mathsf{seed} \overset{R}{\leftarrow} \texttt{Seeds}$
    $p \leftarrow A_2(m, \mathsf{seed}, state)$
    return  1 iff
        $(m \notin I) \wedge (\mathsf{Check}^m(m, \mathsf{seed}, p) = 1)$
```

Zero-Knowledge – There exists a probabilistic polynomial time machine (i.e. a *simulator*) $\mathcal{S}$ such that for all $m \in I$, for all $\mathsf{seed} \in \texttt{Seeds}$, and for all adversaries $\mathcal{A}$, we have that the following distributions

$$\{\mathsf{View}_{\mathcal{A}}[(m, \mathsf{seed}, p) \,|\, p \leftarrow \mathsf{Prove}^m(m, w, \mathsf{seed})]\}$$

$$\text{and} \quad \{\mathsf{View}_{\mathcal{A}}[(m, \mathsf{seed}, p) \,|\, p \leftarrow \mathcal{S}(m, \mathsf{seed})]\}$$

are (perfectly/statistically) indistinguishable. We denote by $\mathsf{Adv}^{\mathsf{sim}}(\mathcal{S}, \mathcal{A})$ the statistical distance between the two distributions, and by $\mathsf{Adv}^{\mathsf{sim}}(T)$ the best distance one can get with a simulator running within time T whatever the adversary $\mathcal{A}$ is (min/max).

Now we briefly discuss why the definition given above is enough for our purposes. Recall that, basically, all that we need to make sure is that a cheating prover should not be able to convince an honest verifier of a false theorem even after having seen a valid (but simulated) proof for *a different* index. Thus, any prover P that manages to produce a false proof after having seen a valid one, can be turned into a prover P', breaking the soundness property, as follows. P' runs algorithm Gen (on input 1^k) in order to obtain the couple (m, t_m). Next, it produces a valid proof p for m, using t_m and feeds P with p. Finally, P' outputs whatever P produces. Notice that p is (at least) statistically indistinguishable from a simulated proof, thus the probability of success of P' is basically the same as the probability of success of P.

Again, for efficiency reasons, and since our results hold in the random oracle model, we focus on zero-knowledge proofs which can be simulated without rewinding the adversary. This can be efficiently realized by taking advantage of the *programmability* of the random oracle: the simulator is given full control of the oracle and, in particular, it is allowed to answer new oracle queries arbitrarily (as long as the provided values are uniformly distributed in the co-domain of the oracle). As a final note, we point out that, in our construction, we always provide proofs which are zero knowledge in a statistical sense [3].

[3] The reason why we cannot achieve perfect zero knowledge will become apparent from the proof of security. Very informally, this comes from the fact that, in the proof of security one has to take into account the fact that, due to the birthday paradox, the simulator may fail to program the oracle correctly with some negligible probability

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[4] 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 evaluate f_m on a random point X efficiently, 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 $\mathbb{G}$) and $\mathsf{Sample}^y(x) = h^x$. 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 information (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).

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 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, actually to be an isomorphism, (n, e) does not really need to be exactly as defined above, which would be very costly to prove (while still possible). It just needs to satisfy $\gcd(e, \varphi(n)) = 1$, which defines our set I. An efficient proof of validity is provided in Appendix D.2, 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

[4] 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.

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*[5] 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. 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 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. This defines our set I. A validity proof is provided, with the mathematical justification, in Section 6, which thus formally defines both Prove^m and Check^m.

3 The Formal Model

3.1 Security Model

Players. We assume to have a fixed set of protocol participants (or *principals*). Each participant can be either a client or a server. For the sole sake of simplicity we denote with A and B two different principals participating in a key exchange protocol P. Principals are allowed to participate to several different, possibly concurrent, executions of P. We model this by allowing each participant an unlimited number of *instances* in which to execute the protocol. If A and B are two parties participating to the key exchange protocol P, we denote with A^i and B^j the instances of A and B – respectively – which are actually running the protocol. Sometimes we will also use the symbol U to denote a generic user instance.

The two parties share a low-entropy secret pw which is drawn from a small dictionary $\mathsf{Password}$, according to an efficiently samplable 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. We denote by $\mathcal{U}_N$ the uniform distribution among N passwords (i.e. $\mathcal{U}_N = 1/N$) and we indicate with $\mathcal{U}_N(n)$ the variable n/N.

Queries. We use the security model introduced by Bellare *et al.* [1], to which 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.

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

In the active scenario, the 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, Execute-queries are essential to deal with dictionary attacks properly. Actually the number q_s of Send-queries directly asked by the adversary *does not* take into account the number of 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 (resp. q_b) is the number of send queries to A (resp. B). 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 (i.e. known to our simulator), assuming the efficient samplability of the underlying dictionary.

3.2 Security Notions

Two main security notions have been defined for key exchange protocols. The first is the semantic security of the key, which means that the exchanged key is unknown to anybody other 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 $\mathsf{Test}(U)$ for which some restrictions must be imposed. Before discussing these, we formalize the notion of *partnering.* Informally we say that two instances are partners if they both participate to the same execution of P. More formally we define a *session id* SID for each instance and we say that two instance are partnered if they share the same (non-null) SID. In what follows we define SID as the concatenation of all the messages (except possibly the last one) sent and received – i.e. the *flow* – by an instance.

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. More formally an instance is said to be *Fresh* if the following conditions are met:

1. The instance has successfully completed execution and therefore it has a non-null SID. More precisely, the notion of successfully completing an execution is formalized as follows. At the beginning of a new execution of the protocol, each instance initializes to "false" a boolean variable (i.e. a *flag*) accept. If, at the end of the execution, everything went through correctly accept is set to "true".
2. Neither it nor its partner 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 $\mathsf{terminate}$ 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 Number-theoretic Assumptions

In this section we state some concrete 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. $\otimes_f$)[6]. As an 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:

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

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:

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

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. Note that the $\mathsf{SuccInSet}^{\mathsf{NI}}_{\mathcal{F}}$ parameter is useful to deal properly with hard-to-invert functions that cannot be efficiently computed. Indeed it allows us to model the case on which the function is implemented as the Diffie-Hellman function (see below). This may seem of little interest in practice: after all the adversary can just choose to output a random element among those in the set. This would allow him to succeed with a factor loss of $1/n$. However, when the hard-to-invert function is implemented as the Diffie-Hellman function, or more generally a random self-reducible function, adding this parameter allows us to obtain a *tight* reduction to the computational Diffie-Hellman (using techniques originally proposed by Shoup [29]). The reader is referred to [29] for further details.

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. More precisely, we define the variable $\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{RSA}}(t)$ as the maximum probability (for all adversaries running in time t) of breaking the one-wayness of RSA, for a correctly generated modulus n. One can define the variable $\mathsf{SuccInSet}^{\mathsf{ow}}_{\mathsf{RSA}}(n, t)$ similarly. 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. Note that the relation above comes from the fact that one exponentiation suffices for checking each value in the set.

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

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

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

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 [29] 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:

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

where $\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{Rabin}}$ and $\mathsf{SuccInSet}^{\mathsf{ow}}_{\mathsf{Rabin}}$ are defined similarly as for the case of RSA. Note that, the relation above 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). We note here that for our construction to work we actually *do not need* to assume that the modulus n is a Blum one. Details are given below, here we just state that, in order for our techniques to go through correctly, we only need to make sure that n is a composite modulus containing at least two different prime factors such that -1 has Jacobi symbol $+1$ in $\mathbb{Z}^\star_n$.

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 in Fig. 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. In what follows, we will write that $\mathcal{F}$ is a family of trapdoor hard-to-invert isomorphisms with parameters (s, q) to indicate that for each $f \in \mathcal{F}$ one has that $|Y_f| > q$ and the set of indexes I has size s.
- 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.

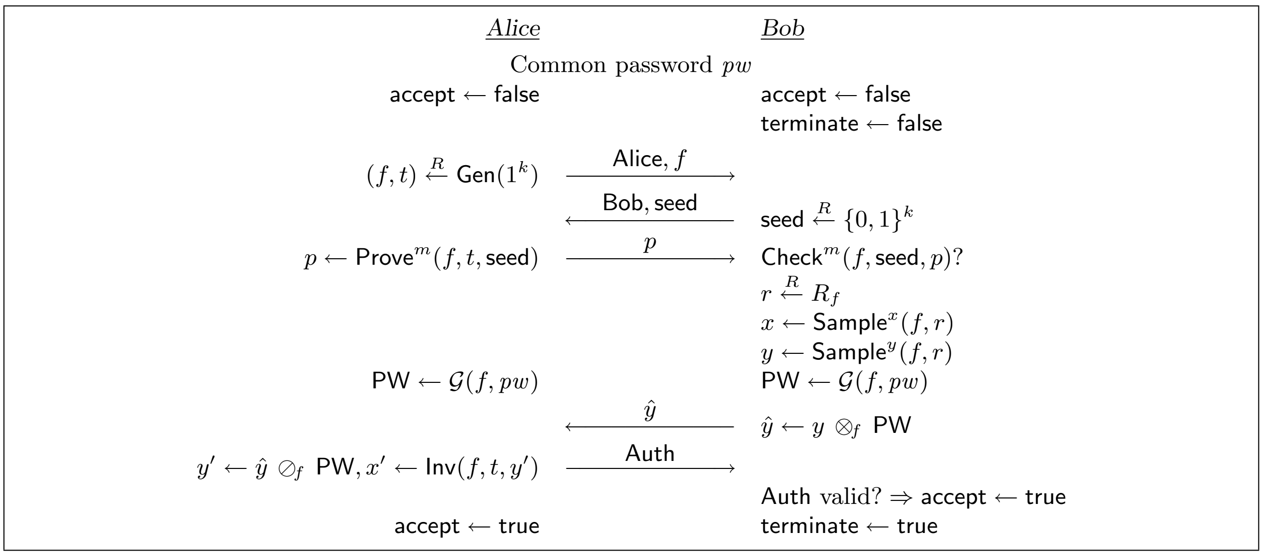
Alice
Bob
Common password pw
$\mathsf{accept} \leftarrow \mathsf{false}$
$\mathsf{accept} \leftarrow \mathsf{false}$
$\mathsf{terminate} \leftarrow \mathsf{false}$
$(f, t) \overset{R}{\leftarrow} \mathsf{Gen}(1^k)$
$\xrightarrow{\mathsf{Alice}, f}$
$\xleftarrow{\mathsf{Bob}, \mathsf{seed}}$
$\mathsf{seed} \overset{R}{\leftarrow} \{0,1\}^k$
$p \leftarrow \mathsf{Prove}^m(f, t, \mathsf{seed})$
$\xrightarrow{p}$
$\mathsf{Check}^m(f, \mathsf{seed}, p)$?
$r \overset{R}{\leftarrow} R_f$
$x \leftarrow \mathsf{Sample}^x(f, r)$
$y \leftarrow \mathsf{Sample}^y(f, r)$
$\mathsf{PW} \leftarrow \mathcal{G}(f, pw)$
$\mathsf{PW} \leftarrow \mathcal{G}(f, pw)$
$\xleftarrow{\hat{y}}$
$\hat{y} \leftarrow y \otimes_f \mathsf{PW}$
$y' \leftarrow \hat{y} \oslash_f \mathsf{PW}, x' \leftarrow \mathsf{Inv}(f, t, y')$
$\xrightarrow{\mathsf{Auth}}$
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 (resp. Bob) as $\mathcal{H}_1(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|x)$ (resp. $\mathcal{H}_1(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|x')$), and sk is computed by Alice (resp. Bob) as $\mathcal{H}_0(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|x)$ (resp. $\mathcal{H}_0(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|x')$.)

5.2 Security Result

Theorem 2 (AKE/UA Security). *We consider the protocol* **IPAKE**, *over a family $\mathcal{F}$ of trapdoor hard-to-invert isomorphisms, with parameter (s, q), where* Password *is an N-word dictionary from which passwords are sampled according to the uniform distribution $\mathcal{U}_N$. Let $\mathcal{A}$ be any adversary whose running time is bounded by t, that is allowed (less than) q_s active interactions with the parties (*Send*-queries) and q_p passive maximum number of hash queries (to $\mathcal{G}$ and any $\mathcal{H}_i$ respectively) that $\mathcal{A}$ is allowed to ask. Then*

$$\mathsf{Adv}^{\mathsf{ake}}_{\mathsf{ipake}}(\mathcal{A}) \le 4\varepsilon$$

and

$$\mathsf{Adv}^{\mathsf{A-auth}}_{\mathsf{ipake}}(\mathcal{A}) \le \varepsilon$$

where ε is a quantity upper-bounded by

$$\frac{q_a + q_b}{N} + 4Q_p \mathsf{SuccInSet}^{\mathsf{NI}}_{\mathcal{F}}(q_h^2, t + 2q_h^2 \tau_{law}) + q_b \mathsf{Succ}^{\mathsf{forge}}(t) + \frac{q_b}{2^{\ell_1}} + \mathsf{Adv}^{\mathsf{sim}}(T) + \frac{Q^2}{2q} + \frac{Q_P^2}{2s}$$

In the relation above,

1. *q_a and q_b denote the number of A and B instances involved in active attacks (each of these quantity is upper-bounded by q_s);*
2. *Q_P denotes the number of involved instances ($Q_P \le 2q_p + q_s$);*
3. *$Q = q_g + q_h + 2q_p + q_s$;*
4. *τ_{law} is the time needed for evaluating one group operation;*
5. *T is the time needed for simulating one proof;*
6. *ℓ is the output length of $\mathcal{H}_0$ (i.e. the function used to produce the session key);*
7. *ℓ_1 is the output length of $\mathcal{H}_1$ (i.e. the function used to generate the authenticator);*
8. $\mathsf{Succ}^{\mathsf{forge}}(t)$ *is the probability that the adversary manages to break the soundness of the provided proof;*
9. $\mathsf{Adv}^{\mathsf{sim}}(T)$ *is the distance between the distribution of the actual proof and the proof simulated within time T.*

Let us first briefly explain the main terms in the above 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 $\mathcal{U}_N(q_a + q_b)$, plus some negligible terms. For technical reasons in the 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 bijection, 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. However, 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. However, this requires the ability to compute $f^{-1}(\mathsf{PW})$. Hence the term $Q_p \times \mathsf{SuccInSet}^{\mathsf{NI}}_{\mathcal{F}}(\cdot,\cdot)$.
– send a correct authenticator Auth, but being lucky. Hence the term $q_b/2^{\ell_1}$.

Additional negligible terms come from very unlikely collisions, or non-perfect simulations. All the remaining kinds of attacks need some information about the password.

5.3 Sketch of the Proof

This proof goes by a sequence of games, starting from $\mathbf{G}_0$ which represents an actual attack. In all the games, we focus on the two following events:

– S (for semantic security). This event occurs if the adversary correctly guesses the bit b involved in the Test-query;
– A (for A-authentication). This event occurs if an instance B^j terminates with no accepting partner instance A^i (with the same transcript $(f, \hat{y}, \mathsf{Auth})$).

In the full proof we will consider the restricted event $\mathsf{SwA} = \mathsf{S} \wedge \neg\mathsf{A}$.

Here we briefly illustrate how every new game differs with respect to the previous one

1. First, we make the classical perfect simulation of the random oracle with random answers, and lists for storing the query-answer pairs as one can see in Fig. 2 (Section 5.4).
2. One then cancels those situations (and declares the adversary wins) in which some collisions appear:
   – collisions on the partial transcripts $(\mathsf{Alice}, \mathsf{Bob}, f, \hat{y})$.
   – collisions on the output of $\mathcal{G}$.

   We furthermore ask for $\mathcal{G}(f, pw)$ when any $\mathcal{H}_i(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|x')$ is queried. Thus the two games just differ because of the birthday paradox, for the collisions. We also now replace the number of queries to $\mathcal{G}$ by q'_g which is upper-bounded by $q_g + q_h$.
3. We furthermore cancel executions which involve an accepted proof of a wrong statement (forgery), about the isomorphic property of a function f. The *soundness* requirement of the proof appears here.
4. Then, the real parties compute the session key sk and the authenticator Auth using private oracles $\mathcal{H}'_0$ and $\mathcal{H}'_1$ respectively, on the input $(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y})$. The authenticator is computed with a private random oracle, then it cannot be guessed by the adversary, better than at random for each attempt, unless the same partial transcript $(\mathsf{Alice}, \mathsf{Bob}, f, \hat{y})$ appeared in another session with a real instance B^j. However, such a case has already been excluded. The A-Auth can thus be broken with a minute probability only. A similar remark can be made about the session key, in the case of no A-Auth break.

   The adversary detects the difference only if it queries the correct values to the public oracles $\mathcal{H}_0$ and $\mathcal{H}_1$. This event, denoted AskH, is now the crucial event whose probability has to be upper-bounded.

5. We introduce a random instance (φ, ρ), where φ is randomly drawn from $\mathcal{F}$, using Gen and $\rho = \varphi(\sigma)$ is uniformly drawn from Y_φ. Actually, the pair (φ, ρ) is given from outside, and the goal of our simulation is to use the adversary to invert φ on ρ. We first introduce φ in the simulation of one specific instance of the party A (its chosen bijection), while we introduce the other part ρ in the simulation of the oracle $\mathcal{G}$ (the common password PW under φ). The *simulatability* requirement of the proof that φ is indeed an isomorphism appears here, since we do not know the trapdoor.
6. Since neither x, nor the password is used, this game is perfectly equivalent to a simpler one, where the party B directly generates a random pair $(\hat{x}, \hat{y} = f(\hat{x}))$. This does not change anything from the view point of the adversary, because of the *isomorphic* property of f. Furthermore the password does not need to be known!
   In order to evaluate the event AskH, we cancel a few more games, wherein for some pair $(f, \hat{y})$, involved in a communication between **an instance** A^i and either the adversary or an instance B^j, there are two distinct passwords pw, and thus elements PW, since $\mathsf{PW} = \mathcal{G}(f, pw)$, such that the tuple $(f, \hat{y}, pw, f^{-1}(\hat{y} \oslash_f \mathsf{PW}))$ is in **H-List**. We can prove that such a collision is unlikely unless one can invert the family $\mathcal{F}$. This is the technical part of the proof.
   Finally, since the password is never used during the simulation, it can be chosen at the very end only. Then, an information-theoretic analysis can be performed, which simply uses cardinalities of some sets. Simple counting arguments lead to the theorem.

□

5.4 Complete Security Proof

In this proof, we incrementally define a sequence of games starting from the one describing a real execution of the protocol (i.e. $\mathbf{G}_0$) and ending up with game $\mathbf{G}_6$.

Game $\mathbf{G}_0$: This is the real protocol, in the random-oracle model. We are interested in the two following events:

- S_0 (for semantic security), which occurs if the adversary correctly guesses the bit b involved in the Test-query;
- A_0 (for A-authentication), which occurs if an instance B^j accepts with no partner instance A^i (possessing the same transcript (f, y, Auth).

By definition of $\mathsf{Adv}^{\mathsf{ake}}_{\mathsf{ipake}}(\mathcal{A})$ and $\mathsf{Adv}^{\mathsf{A-auth}}_{\mathsf{ipake}}(\mathcal{A})$ one has that

$$\mathsf{Adv}^{\mathsf{ake}}_{\mathsf{ipake}}(\mathcal{A}) = 2\Pr[\mathsf{S}_0] - 1 \qquad \mathsf{Adv}^{\mathsf{A-auth}}_{\mathsf{ipake}}(\mathcal{A}) = \Pr[\mathsf{A}_0]. \tag{1}$$

We remark that, in the following games $\mathbf{G}_n$ below, we will focus on the event A_n, and the restricted event $\mathsf{SwA}_n = \mathsf{S}_n \wedge \neg \mathsf{A}_n$.

Game $\mathbf{G}_1$: Here we modify the previous game by simulating the hash oracles. This will involve $\mathcal{G}$, $\mathcal{H}_0$ and $\mathcal{H}_1$, but also two additional hash functions, $\mathcal{H}'_i : \{0,1\}^\star \rightarrow \{0,1\}^{\ell_i}$ (for $i = 0, 1$) that will be introduced in Game $\mathbf{G}_4$. This is done by introducing and maintaining three hash lists **G-List**, **H-List** and **H'-List** (a formal description of these lists is given in Fig. 2).

Additionally, during this game, we simulate the answers of all the instances for all queries (i.e. Send, $\mathsf{Execute}$, Reveal and Test-queries) asked by the adversary. This is done *exactly* as real instances would do. In particular, a formal description of this part of the simulation is given in Figs. 3–5.

Now, notice that, since (a) the simulated hash oracles produce outputs which have the *exact* same distribution as the outputs produced by the random oracles and (b) all the queries are answered *exactly* as the real instances would do, one can easily deduce that this game remains perfectly indistinguishable from the real one (i.e. previous game).

Game $\mathbf{G}_2$: As a first step we, essentially, modify the way on which queries to $\mathcal{H}_i$ are managed. In particular, whenever a query to $\mathcal{H}_i$ occurs, we query $\mathcal{G}$ as well. Notice that, $\mathcal{H}_i$ *is not* replaced by $\mathcal{G}$;

For a hash-query $\mathcal{H}_i(q)$ (resp. $\mathcal{H}'_i(q)$), such that a record (i, q, r) appears in **H-List** (resp. **H'-List**), the answer is r. Otherwise one chooses a random element $r \in \{0,1\}^{\ell}$, answers and, adds the record (i, q, r) to **H-List** (resp. **H'-List**).

▶**Rule** $\mathcal{H}^{(1)}$

| Nothing to do. `% To be defined later`

For a hash-query $\mathcal{G}(f, q)$ such that a record $(f, q, r, \star, \star)$ appears in **G-List**, the answer is r. Otherwise the answer r is defined according to the following rule:

▶**Rule** $\mathcal{G}^{(1)}$

| Choose a random element $r \overset{R}{\leftarrow} Y_f$. The record $(f, q, r, \perp, \perp)$ is added to **G-List**.

Note: the fourth and fifth components of the elements of this list will be explained later.

Fig. 2. Simulation of the **IPAKE** protocol: $\mathcal{G}$ and $\mathcal{H}_i$ oracles

We answer to the Send-queries to an A-instance as follows:

– A $\mathsf{Send}(A^i, \mathtt{Start})$-query is processed according to the following rule:
  ▶**Rule A1**$^{(1)}$
  | Choose a random trapdoor isomorphism $(f, t) \overset{R}{\leftarrow} \mathsf{Gen}(1^k)$.
  Then the query is answered with (Alice, f), and the instance goes to an expecting state.
– If the instance A^i is in an expecting state, a query $\mathsf{Send}(A^i, (\mathsf{Bob}, \mathsf{seed}))$ is processed by generating the proof of validity for f:
  ▶**Rule A2**$^{(1)}$
  | Compute $p \leftarrow \mathsf{Prove}^m(f, t, \mathsf{seed})$.
  And the query is answered with p.
– If the instance A^i is in an expecting state, a query $\mathsf{Send}(A^i, \hat{y})$, for $\hat{y} \in Y_f$, is processed by computing the authenticator and the session key. We apply the following rules:
  ▶**Rule A3**$^{(1)}$
  | Compute $\mathsf{PW} \leftarrow \mathcal{G}(f, pw)$, $y' \leftarrow \hat{y} \oslash_f \mathsf{PW}$ and $x' \leftarrow f^{-1}(y')$.
  ▶**Rule A4**$^{(1)}$
  | Compute the authenticator and the session key

$$\mathsf{Auth} \leftarrow \mathcal{H}_1(\mathsf{Alice} \| \mathsf{Bob} \| f \| \hat{y} \| pw \| x'),$$
$$sk_A \leftarrow \mathcal{H}_0(\mathsf{Alice} \| \mathsf{Bob} \| f \| \hat{y} \| pw \| x').$$

  Finally the instance accepts, and the query is answered with Auth.

Fig. 3. Simulation of the **IPAKE** protocol: Send-queries to A

We answer to the Send-queries to a B-instance as follows:

- A $\mathsf{Send}(B^j, (\mathsf{Alice}, f))$-query is processed by generating a random seed, which is then the answer, and the instance goes to an expecting state.
- If the instance B^j is in an expecting state, a $\mathsf{Send}(B^j, p)$-query is first processed by checking the validity of the proof, evaluating $\mathsf{Check}^m(f, \mathsf{seed}, p)$, and in case of validity, one applies the following rules:

  ▶**Rule B1**$^{(1)}$

  Generate $(x, y = f(x))$ using the Sample^x and Sample^y algorithms, compute $\mathsf{PW} \leftarrow \mathcal{G}(f, pw)$ and $\hat{y} \leftarrow y \otimes_f \mathsf{PW}$.

  Then the query is answered with $(\mathsf{Bob}, \hat{y})$, and the instance goes to an expecting state.
- If the instance B^j is in an expecting state, a query $\mathsf{Send}(B^j, \mathsf{Auth})$ is processed by computing the alleged authenticator, and the session key.

  ▶**Rule B2**$^{(1)}$

  Compute the expected authenticator and the session key

$$\mathsf{Auth}' \leftarrow \mathcal{H}_1(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|x),$$
$$sk_B \leftarrow \mathcal{H}_0(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|x).$$

  If $\mathsf{Auth}' = \mathsf{Auth}$, then instance accepts. In any case, it terminates.

Fig. 4. Simulation of the **IPAKE** protocol: Send-queries to B

An $\mathsf{Execute}(A^i, B^j)$-query is processed using successively the simulations of the Send-queries:

$$(\mathsf{Alice}, f) \leftarrow \mathsf{Send}(A^i, \mathtt{Start}), (\mathsf{Bob}, \mathsf{seed}) \leftarrow \mathsf{Send}(B^j, (\mathsf{Alice}, f)),$$
$$p \leftarrow \mathsf{Send}(A^i, (\mathsf{Bob}, \mathsf{seed})), \hat{y} \leftarrow \mathsf{Send}(B^j, p),$$
$$\mathsf{Auth} \leftarrow \mathsf{Send}(A^i, \hat{y}), \mathsf{Send}(B^j, \mathsf{Auth}),$$

and outputting the transcript $((\mathsf{Alice}, f), (\mathsf{Bob}, \mathsf{seed}), p, \hat{y}, \mathsf{Auth})$.

A $\mathsf{Reveal}(U)$-query returns the session key (sk_A or sk_B) computed by the instance U (if the latter has accepted).

A $\mathsf{Test}(U)$-query first gets sk from $\mathsf{Reveal}(U)$, and flips a coin b. If $b = 1$, we return the value of the session key sk, otherwise we return a random value drawn from $\{0, 1\}^{\ell_0}$.

Fig. 5. Simulation of the **IPAKE** protocol: Other queries

rather $\mathcal{G}$ is called whenever a call to $\mathcal{H}_i$ occurs. In particular, each output of $\mathcal{H}_i$ is still an element chosen uniformly and at random in the appropriate range. We modify the rule $\mathcal{H}_i^{(1)}$, introduced in game $\mathbf{G}_1$ (see Fig. 2), as follows:

▶**Rule** $\mathcal{H}^{(2)}$

The query q is parsed as $q = \mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|x$, then one queries $\mathcal{G}(f, pw)$.

The number of queries to $\mathcal{G}$ thus becomes $q_g' \leq q_g + q_h$.

Next, we consider in our analysis the collisions that may occur in the simulation of the hash functions. More precisely, in this experiment, we simulate the oracles as before except that we halt all executions in which a collision occurs (and set events S and A to true). These collisions may be of the following two kinds:

- Collisions on the partial transcripts $(\mathsf{Alice}, \mathsf{Bob}, f, \hat{y})$. This case encompasses collisions on the outputs of the $\mathcal{H}$'s. We remark here that, since every transcripts is produced by at least one honest party, we are guaranteed that either f is correctly generated (i.e. chosen uniformly and at random from $\mathcal{F}$) or y is uniformly distributed. In both cases, the probability of a collision is bounded by $\frac{(q_s+q_p)^2}{2} \times (\frac{1}{s} + \frac{1}{q})$, according to the birthday paradox.
- Collisions on the output of $\mathcal{G}$. By a similar reasoning, the probability of such collisions is bounded by $\frac{{q_g'}^2}{2q}$.

We denote with Coll the event that at least one of the collisions described above happens. Clearly, the probability of such an event is bounded by the probability of collisions in the partial transcript plus the probability of collisions on the output of $\mathcal{G}$. This leads to the following:

$$\Pr[\mathsf{Coll}_2] \leq \frac{(q_s + q_p)^2}{2} \times \left(\frac{1}{s} + \frac{1}{q}\right) + \frac{{q_g}'^2}{2q} \leq \frac{Q^2}{2q} + \frac{Q_P^2}{2s}. \tag{2}$$

where the second inequality follows from the fact that $q_g' \leq q_g + q_h$, $Q \leq q_g + q_h + 2q_p + q_s$ and $Q_P \leq 2q_p + q_s$.

Game $\mathbf{G}_3$: In this game we consider all those executions of the protocol on which the adversary manages to produce false (zero knowledge) proofs that are actually accepted as correct ones. We denote by Forge the event that a forged proof is produced by the adversary. We proceed exactly as in previous games with the only difference that we halt all those executions where forged proofs are provided. Thus, all we need to prove is that event Forge happens only with small enough probability.

Note that the number of (different) proofs is at most q_b, thus we have that

$$\Pr[\mathsf{Forge}_3] \leq q_b \mathsf{Succ}^{\mathsf{forge}}(t). \tag{3}$$

Moreover since we are considering proofs which are zero knowledge in the sense of definition 1, we can conclude that $\mathsf{Succ}^{\mathsf{forge}}(t)$ is negligible. More precisely, the reduction can be sketched as follows. Given an adversary $\mathcal{B}$ that manages to produce a forgery, we show how to construct an adversary (A_1, A_2) breaking the soundness of the zero knowledge proof as follows. A_1 starts by guessing which proof instance j will be the forgery and performs the simulation until $\mathcal{B}$ sends the j-th f to Bob. Then A_1 outputs f and *state*, where the latter is the state of the simulation. A_2 gets $(f$,*state*$)$ and a random seed seed, simulates Bob's answer using the received seed and continues the simulation in the obvious way. Then, if $\mathcal{B}$ sends a proof p to Bob, A_2 outputs p.

Remark 3. Game $\mathbf{G}_3$ differs from the previous one in that all executions in which the adversary produces an accepting proof of a false statement are halted. Notice, however, that it is not always possible to detect this fact efficiently. To address this concern one might just assume that the adversary wins if a forgery occurs and then bound, as we did, the probability that such an event happens. From a technical point of view, later proofs should then bound the probability of events E as $\Pr[E|\neg\mathsf{Forge}]$, rather than simply $\Pr[E]$. However, to make the proof easier to read, we prefer not to add the conditional probability to all later bounds. Still, the reader should consider all future bounds to be conditioned on $\neg\mathsf{Forge}$.

Game $\mathbf{G}_4$: In this game we replace the oracles $\mathcal{H}_1$ and $\mathcal{H}_0$ with two (secret) oracles $\mathcal{H}'_1$ and $\mathcal{H}'_0$. Recall that the oracles $\mathcal{H}_1$ and $\mathcal{H}_0$ are used to compute the authenticators and the session keys, respectively. By this modification the authenticators and the session keys are computed using $\mathcal{H}'_1$ and $\mathcal{H}'_0$ respectively, and become, thus, unpredictable (in a strong information theoretic sense) to any adversary. Since oracles $\mathcal{H}_1$ and $\mathcal{H}_0$ were used by rules **A4/B2** (see Figs. 3 and 4), we modify them as follows:

▶Rule A4/B2$^{(4)}$

| Compute the authenticator $\mathsf{Auth} \leftarrow \mathcal{H}'_1(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y})$.
| Compute the session key $sk_{A/B} \leftarrow \mathcal{H}'_0(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y})$.

We denote $\mathsf{PW} = \mathcal{G}(f, pw)$. Now, observe that the common secret $x' = x = f^{-1}(\hat{y} \oslash_f \mathsf{PW})$ depends only on f, $\hat{y}$ and on the common password pw shared by the participants. This implies that games $\mathbf{G}_4$ and $\mathbf{G}_3$ are indistinguishable as long as the adversary does not explicitly query $\mathcal{H}_1$ or $\mathcal{H}_0$ on input $(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|f^{-1}(\hat{y} \oslash_f \mathsf{PW}))$. More formally, games $\mathbf{G}_4$ and $\mathbf{G}_3$ remains indistinguishable unless the following event $\mathsf{AskH}_4 = \mathsf{AskH0w1}_4 \vee \mathsf{AskH1}_4$ occurs, where :

– $\mathsf{AskH1}_4$: $(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|f^{-1}(\hat{y} \oslash_f \mathsf{PW}))$ has been queried by $\mathcal{A}$ to $\mathcal{H}_1$ for some transcript $((\mathsf{Alice}, f), (\mathsf{Bob}, \mathsf{seed}), p, \hat{y}, \mathsf{Auth})$;
– $\mathsf{AskH0w1}_4$: $(\mathsf{Alice}\|\mathsf{Bob}\|f\|\hat{y}\|pw\|f^{-1}(\hat{y} \oslash_f \mathsf{PW}))$ has been queried by $\mathcal{A}$ to $\mathcal{H}_0$ for some transcript $((\mathsf{Alice}, f), (\mathsf{Bob}, \mathsf{seed}), p, \hat{y}, \mathsf{Auth})$, where some party has accepted, but event $\mathsf{AskH1}_4$ did not happen.

We need to show that $\Pr[\mathsf{AskH}_4]$ is small enough, however we postpone this until later. For now, notice that after the modifications above have been implemented, we no longer need to know the value x nor to compute the value x' either. This is because, in rules **A4/B2** we do not even use them to compute the authenticator and the session key. Thus we can simplify rules **A3** and **B1** as follows:

▶Rule A3$^{(4)}$

| Do nothing.

▶Rule B1$^{(4)}$

| Generate a random pair $(\hat{x}, \hat{y} = f(\hat{x}))$ using the Sample^x and Sample^y algorithms.

As the alert reader may have already noticed, even the actual password is not used anymore. Indeed, by the isomorphic property of the function f, the value $\hat{y}$ defined by rule **B1$^{(4)}$** is perfectly indistinguishable from one generated according to the original rule **B1$^{(1)}$**. As a matter of fact, there exists a unique pair (x, y), such that

$$x = f^{-1}(\hat{y} \oslash_f \mathsf{PW}) = \hat{x} \ominus_f f^{-1}(\mathsf{PW}) \qquad y = f(x) \qquad \text{such that } \hat{y} = y \otimes_f \mathsf{PW}.$$

As already mentioned, the authenticator is computed by means of a random oracle that is kept secret from the adversary. This means that, for each execution of the protocol, the adversary cannot guess the authenticator better than at random, unless, of course, the same partial transcript $(\mathsf{Alice}, \mathsf{Bob}, f, \hat{y})$ appeared in some another session with a real instance B^j. This last case, however, has already been considered (i.e. excluded) in Game $\mathbf{G}_2$. A similar reasoning can be done for the session key. Thus we can conclude that

$$\Pr[\mathsf{A}_4] \le \frac{q_b}{2^{\ell_1}} \qquad \Pr[\mathsf{SwA}_4] = \frac{1}{2}. \tag{4}$$

This means that we can write

$$\Pr[\mathsf{A}_0] \le \frac{q_b}{2^{\ell_1}} + \Delta \quad \text{and } \Pr[\mathsf{SwA}_0] \le \frac{1}{2} + \Delta,$$

where

$$\Delta \le \left(\frac{Q^2}{2q} + \frac{Q_P^2}{2s}\right) + q_b \mathsf{Succ}^{\mathsf{forge}}(t) + \Pr[\mathsf{AskH}_4].$$

In order to start evaluating $\Pr[\mathsf{AskH}_4]$, we remark that, once having excluded collisions of partial transcripts, the event $\mathsf{AskH1}_4$ can be split in 3 mutually-exclusive sub-events:

– $\mathsf{AskH1\text{-}Passive}_4$: the transcript $((\mathsf{Alice}, f), (\mathsf{Bob}, \mathsf{seed}), p, \hat{y}, \mathsf{Auth})$ comes from an execution between instances of A and B (Execute-queries or forward of Send-queries, replay of part of them). In this case both f and $\hat{y}$ have been simulated;
– $\mathsf{AskH1\text{-}WithA}_4$: the execution involved an instance of A, but $\hat{y}$ has not been sent by any instance of B. This means that f has been simulated, but y has been manufactured by the adversary;
– $\mathsf{AskH1\text{-}WithB}_4$: the execution involved an instance of B, but f has not been sent by any instance of A. This means that $\hat{y}$ has been simulated, but f has been manufactured by the adversary.

Game $\mathbf{G}_5$: In this game we introduce a random challenge (φ, ρ), where φ is randomly drawn from $\mathcal{F}$ and ρ is randomly drawn from Y_φ (or equivalently, because of the isomorphic property of f, σ is randomly drawn from X_φ and $\rho = \varphi(\sigma)$). In particular we try to invert φ on ρ, i.e. to compute the value σ.

We introduce the challenge (φ, ρ) in the simulation of the party A, using the well known "plug and pray" approach: we choose a value c uniformly and at random in the set $\{1 \ldots q_a\}$, (recall that q_a is the number of instances of the party A, involved in active attacks), and then we use φ in the c-th instance. More formally:

►**Rule A1**$^{(5)}$

If this is the c-th instance of the party A, f is set to φ. Otherwise, f is randomly drawn from $\mathcal{F}$, as follows: Choose a random trapdoor isomorphism $(f, t) \overset{R}{\leftarrow} \mathsf{Gen}(1^k)$.

For consistency, we modify the behavior of the c-th instance (of A) in a standard way:

►**Rule A2**$^{(5)}$

If $f = \varphi$, use the simulator to build the proof of validity p of f. Otherwise, compute $p \leftarrow \mathsf{Prove}^m(f, t, \mathsf{seed})$.

We introduce the remaining component ρ of the challenge in the simulation of the oracle $\mathcal{G}$, using again the homomorphic property of φ. Specifically, the simulation introduces values in the third and fourth components of the elements of **G-List** (i.e. in the components that were left empty in game $\mathbf{G}_1$ – see Fig. 2). These values correspond to random input/output couples of φ. This leads us to modify rule $\mathcal{G}^{(1)}$ as follows:

►**Rule** $\mathcal{G}^{(5)}$

If $f = \varphi$, generate a random pair $(u, v = \varphi(u))$ using the algorithms Sample^x and Sample^y, as well as a random bit b, and compute $r \leftarrow v$, if $b = 0$, and $r \leftarrow v \otimes_\varphi \rho$, if $b = 1$. Then, the record (φ, q, r, u, b) is added to **G-List**.

Else Choose a random element $r \overset{R}{\leftarrow} Y_f$. Then, the record $(f, q, r, \perp, \perp)$ is added to **G-List**.

Note that, if $f \neq \varphi$, this rule remains exactly the same as the original one.

On the other hand, if $f = \varphi$ its execution is indistinguishable from an execution of the original one, simply because of the isomorphic property of φ, except for the simulation of the proof which may just be statistically indistinguishable: $\mathsf{Adv}^{\mathsf{sim}}(T)$ is the distance between the original distribution and the one provided by the simulator in time T.

Game $\mathbf{G}_6$: In this game we, finally, evaluate the probability of the event AskH (or, more precisely, of all its sub-cases).

First notice that, during the simulation as per game $\mathbf{G}_5$, the password is actually never used. Thus, one can safely choose it at the very end of the protocol. This is a crucial observation as it tells us that the entire simulation is basically independent from the chosen password. Thus, the security of the protocol is also independent from the chosen password and solely depends from public parameters.

Before proceeding to the actual analysis we consider (and exclude) all those executions where, for some pair $(f, \hat{y})$, used by **an instance** A^i and either the adversary or an instance B^j, there exist two distinct passwords pw_β, for $\beta = 0, 1$, (and, correspondingly, two values PW_β, as $\mathsf{PW}_\beta = \mathcal{G}(f, pw_\beta)$), such that the tuples $(f, \hat{y}, pw_\beta, f^{-1}(\hat{y} \oslash_f \mathsf{PW}_\beta))$ are both in **H-List**. We denote such an event as CollH_6. Clearly, by these positions we have:

$$|\Pr[\mathsf{AskH}_6] - \Pr[\mathsf{AskH}_5]| \leq \Pr[\mathsf{CollH}_6]. \tag{5}$$

With the next technical lemma we actually provide a precise upper-bound to the probability of event CollH_6

Lemma 4. *Under the assumption that $\mathcal{F}$ is a family of trapdoor hard-to-invert group isomorphisms, for any pair $(f, \hat{y})$, used in a communication with an instance A^i, there is at most one valid element* PW*, obtained as output of $\mathcal{G}$, such that $(f, \hat{y}, pw, f^{-1}(\hat{y} \oslash_f \mathsf{PW}))$ is the list* **H-List***. Formally,*

$$\Pr[\mathsf{CollH}_6] \leq 2(q_a + q_p)\mathsf{SuccInSet}(q_h^2, t + 2q_h^2\tau_{law}). \tag{6}$$

Proof. First notice that, since the execution which uses φ is perfectly indistinguishable from any other execution, we have that $f = \varphi$ with probability $1/(q_a + q_p)$. Now, assume we are in the execution where φ is used, we proceed by *reductio ad absurdum*: we assume that there exists a couple $(\varphi, \hat{y})$, for which $\mathsf{PW}_0 = \varphi(u_0) \otimes_\varphi b_0 \cdot \rho = \mathcal{G}(\varphi, pw_0)$ and $\mathsf{PW}_1 = \varphi(u_1) \otimes_\varphi b_1 \cdot \rho = \mathcal{G}(\varphi, pw_1)$ are both obtained from the $\mathcal{G}$ oracle and such that the tuples $(\varphi, \hat{y}, pw_i, \varphi^{-1}(\hat{y} \oslash_\varphi \mathsf{PW}_i))$ are both in **H-List**, for $i = 0, 1$. We need to show that, in such a case, one can invert φ, thus reaching a contradiction.

To this aim, we consider the following quantity:

$$Z \stackrel{\mathsf{def}}{=} \varphi^{-1}(\hat{y} \oslash_\varphi \mathsf{PW}_1) \ominus_\varphi \varphi^{-1}(\hat{y} \oslash_\varphi \mathsf{PW}_0) = \varphi^{-1}(\mathsf{PW}_1 \oslash_\varphi \mathsf{PW}_0).$$

With probability $1/2$, the bits b_0 and b_1 flipped to generate PW_0 and PW_1 respectively, are distinct. In particular, and w.l.o.g., we may assume that $b_i = i$ for $i = 0, 1$. This means that

$$Z = \varphi^{-1}(\varphi(u_1) \otimes_\varphi \rho \oslash_\varphi \varphi(u_0)) = u_1 \ominus_\varphi u_0 \oplus_\varphi \varphi^{-1}(\rho) = u_1 \ominus_\varphi u_0 \oplus_\varphi \sigma.$$

As a consequence, $\sigma = Z \oplus_\varphi u_0 \ominus_\varphi u_1$.

More concretely, if a pair of passwords (pw_0, pw_1) satisfying the conditions above exists, then one can compute a σ which is in the list with probability $1/(2(q_a + q_p))$. Indeed, first one computes all the Z's satisfying the relation above from all the pairs of queries to $\mathcal{H}$. This determines all the possible candidates for the value of σ. Notice that there are at most q_H^2 of such candidates and computing each of them requires 2 (group) operations. Thus assuming that $b_0 \neq b_1$, and the correct instance for A has been guessed, the correct σ is in the list.

This concludes the proof. □

As a final step, we study, separately, the three sub-cases of $\mathsf{AskH1}$ and then $\mathsf{AskH0w1}$ (recall that now we do not have to worry about collisions involving partial transcripts, queries to $\mathcal{G}$, or passwords pw in $\mathcal{H}$-queries):

– AskH1-Passive: For this event (where both f and $\hat{y}$ are simulated), we give the following lemma:

Lemma 5. *For any pair* $(f, \hat{y})$*, involved in a passive transcript, there is no valid element* PW *such that* $(f, \hat{y}, pw, f^{-1}(\hat{y} \oslash_\varphi \mathsf{PW}))$ *is in* **H-List***, unless one can invert* $\mathcal{F}$*:*

$$\Pr[\mathsf{AskH1\text{-}Passive}_6] \leq 2(q_a + q_p)\mathsf{SuccInSet}(q_h, t + 2q_h\tau_{law}). \tag{7}$$

Proof. Assume there exist $(f, \hat{y})$ involved in a passive transcript, and $\mathsf{PW} = f(u) \otimes_\varphi b \cdot \rho$ such that the tuple $(f, \hat{y} = f(\hat{x}), pw, Z \overset{\mathsf{def}}{=} f^{-1}(\hat{y} \oslash_f \mathsf{PW}))$ is in **H-List**. Then, as above, with probability $1/2(q_a + q_p)$, $f = \varphi$ and $b = 1$:

$$Z = \varphi^{-1}(\varphi(\hat{x}) \oslash_\varphi \varphi(u) \oslash_\varphi \rho) = \hat{x} \ominus_\varphi u \ominus_\varphi \varphi^{-1}(\rho) = \hat{x} \ominus_\varphi u \ominus_\varphi \sigma.$$

As a consequence, $\sigma = \hat{x} \ominus_\varphi u \ominus_\varphi Z$. By outputting all the candidates for σ (computed from all the Z in the $\mathcal{H}$-queries) the correct value is in the list (with a similar probability as above). □

– AskH1-WithB: this may correspond to an attack where the adversary tries to impersonate A to B (break unilateral authentication). However, each authenticator sent by the adversary has been computed with at most one pw. Since we choose pw at the very end only, the latter is among the involved passwords with probability:

$$\Pr[\mathsf{AskH1\text{-}WithB}_6] \leq \mathcal{U}_N(q_b) = \frac{q_b}{N}. \tag{8}$$

– AskH1-WithA: Lemma 4 above applied to games where the event CollH_6 did not happen (and without $\mathcal{G}$-collision), states that for each pair $(f, \hat{y})$ involved in a transcript with an instance A^i, there is at most one element pw such that for $\mathsf{PW} = \mathcal{G}(f, pw)$, the corresponding tuple is in **H-List**: the probability over a random password chosen at the very end only is thus less than $\mathcal{D}(q_s)$. As a consequence,

$$\Pr[\mathsf{AskH1\text{-}WithA}_6] \leq \mathcal{U}_N(q_a) = \frac{q_a}{N}. \tag{9}$$

About AskH0w1 (when the three above events did not happen), it means that only executions with an instance of A (and either B or the adversary) may lead to acceptance. Exactly the same analysis as for AskH1-Passive and AskH1-WithA leads to

$$\Pr[\mathsf{AskH0w1}_6] \leq \frac{q_a}{N} + 2(q_a + q_p)\mathsf{SuccInSet}(q_h, t + q_h\tau_{law}). \tag{10}$$

However, one can note that the latter upper-bound in inequality (10) actually used the same bad cases as for upper-bounding events $\mathsf{AskH1\text{-}Passive}_6$ and $\mathsf{AskH1\text{-}WithA}_6$, since **H-List** contains all the queries to both $\mathcal{H}_0$ and $\mathcal{H}_1$. Then we get an upper-bound for the probability of AskH_6:

$$\Pr[\mathsf{AskH}_6] \leq \frac{q_a + q_b}{N} + 2(q_a + q_p)\mathsf{SuccInSet}^{\mathsf{ow}}_{\mathcal{F}}(q_h, t + q_h\tau_{law}). \tag{11}$$

Combining all the above equations, one gets

$$\Pr[\mathsf{A}_0] \leq \frac{q_b}{2^{\ell_1}} + \Delta \qquad \Pr[\mathsf{SwA}_0] \leq \frac{1}{2} + \Delta,$$

where

$$\begin{aligned}
\Delta &\leq 2(q_a + q_p)\left(\mathsf{SuccInSet}(q_h^2, t + 2q_h^2\tau_{law}) + \mathsf{SuccInSet}(q_h, t + 2q_h\tau_{law})\right) \\
&\quad + \frac{q_a + q_b}{N} + \mathsf{Adv}^{\mathsf{sim}}(T) + q_b\mathsf{Succ}^{\mathsf{forge}}(t) + \frac{Q^2}{2q} + \frac{Q_P^2}{2s} \\
&\leq \frac{q_a + q_b}{N} + 4(q_a + q_p)\mathsf{SuccInSet}(q_h^2, t + 2q_h^2\tau_{law}) + \mathsf{Adv}^{\mathsf{sim}}(T) + q_b\mathsf{Succ}^{\mathsf{forge}}(t) \\
&\quad + \frac{Q^2}{2q} + \frac{Q_P^2}{2s}.
\end{aligned}$$

We conclude the proof by noticing that $\Pr[\mathsf{S}_0] \leq \Pr[\mathsf{SwA}_0] + \Pr[\mathsf{A}_0]$.

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 Fig. 6. Here we describe the details of this specific implementation.

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

Fig. 6. 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 $\{0,1\}^k$, where $2^{k-1} < n < 2^k$, and we iterate it until we get an output in J_n. The details of this technique are presented in Appendix B. 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 (see Appendix A for the full proof) we show that if n is a composite modulus (with at least two different prime factors) then the proposed function is an isomorphism.

Theorem 6. *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.*

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 $\mathcal{H}_3$ (resp. $\mathcal{H}_5$) is a random oracle onto $\{0,1\}^k$ (resp. $\{0,1\}^\ell$) 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 $i = 1, \ldots, \ell$,	
$(\beta_i, \alpha_{i,2c_i}, \alpha_{i,2c_i+1}, \{h_{i,j}\})$	(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. 7. Proof of the Correct Modulus

The theorem above leads to the protocol `Prove-Surjective` (see in Fig. 7). 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 in Fig. 7). The correctness (completeness, soundness and zero-knowledge properties) of these protocols is presented in Appendix C.

Remark 7. 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.

7 Conclusion

In this paper, we introduced a new useful primitive we called the "trapdoor hard-to-invert isomorphism family", and we provided some examples, based on the Diffie-Hellman assumption, the RSA assumption and factoring. Then, we presented a generic password-based authenticated key exchange protocol, based on any such primitive, with a formal security proof. The instantiation with the signed squaring function is definitely the most interesting one, since this is, to our knowledge, the first password-based authenticated key exchange scheme secure under the intractability assumption for factoring. Furthermore, the computational cost is minimal for one of the parties. We also present some other concrete instantiations in Appendix D, based on the Diffie-Hellman problem (which is actually one of the AuthA variants, proposed in IEEE P1363 by Bellare and Rogaway [3], or PAK [6]) and RSA (which is similar to the RSA-OKE schemes [20, 22, 21, 31], but very efficient).

Acknowledgments

We thank the anonymous referees for their fruitful comments. We thank Ivan Visconti for several discussions on zero-knowledge proofs.

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 Exact Security of Digital Signatures – How to Sign with RSA and Rabin. In *Eurocrypt '96*, LNCS 1070, pages 399–416. Springer-Verlag, Berlin, 1996.
3. M. Bellare and P. Rogaway. The AuthA Protocol for Password-Based Authenticated Key Exchange. Contributions to IEEE P1363. March 2000.
4. 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.
5. 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.
6. 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/`.
7. 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.
8. E. Bresson, O. Chevassut, and D. Pointcheval. New Security Results on Encrypted Key Exchange. In *PKC '04*, LNCS 2947, pages 145–159. Springer-Verlag, Berlin, 2004.
9. D. Catalano, D. Pointcheval, and T. Pornin. IPAKE: Isomorphisms for Password-based Authenticated Key Exchange. In *Crypto '04*, LNCS 3152, pages 477–493. Springer-Verlag, Berlin, 2004.
10. A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano and A. Sahai Robust Non-interactive Zero-Knowledge In *Crypto '01*, LNCS 2139, pages 566–598. Springer-Verlag, Berlin, 2001.
11. W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT–22(6):644–654, November 1976.
12. Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong Key-Insulated Signature Schemes. In *PKC '03*, LNCS 2567, pages 130–144. Springer-Verlag, Berlin, 2003.
13. T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, IT–31(4):469–472, July 1985.
14. 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.
15. O. Goldreich and Y. Lindell. Session-Key Generation Using Human Passwords Only. In *Crypto '01*, LNCS 2139, pages 408–432. Springer-Verlag, Berlin, 2001.
16. S. Halevi and H. Krawczyk. Public-Key Cryptography and Password Protocols. In *Proc. of the 5th CCS.* ACM Press, New York, 1998.
17. 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.
18. N. Koblitz. *A Course in Number Theory and Cryptography.* Number 114 in Graduate Texts in Math. Springer-Verlag, New York, 1994. Second edition.
19. E. Landau. *Handbuch der Lehre von der Verteilung der Primzahlen.* Teubner, Berlin, 1909.
20. S. Lucks. Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys. In *Proc. of the Security Protocols Workshop*, LNCS 1361, pages 79–90. Springer-Verlag, Berlin, 1997.
21. 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.
22. P. MacKenzie and R. Swaminathan. Secure Network Authentication with Password Identification. Submission to IEEE P1363a. August 1999.
23. 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.
24. P. Paillier. Public-Key Cryptosystems Based on Discrete Logarithms Residues. In *Eurocrypt '99*, LNCS 1592, pages 223–238. Springer-Verlag, Berlin, 1999.

25. 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.
26. 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.
27. J. N. Rosser and L. Schoenfeld. Approximate Formulas for some Functions of Prime Numbers. *Illinois Journal on Mathematics*, 6(1):64–94, 1962.
28. A. Sahai. Non-Malleable Non-Interactive Zero-Knowledge and Chosen-Ciphertext Security. In *Proc. of the 40th FOCS*, pages 543–553. IEEE, New York, 1999.
29. V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In *Eurocrypt '97*, LNCS 1233, pages 256–266. Springer-Verlag, Berlin, 1997.
30. M. Zhang. New Approaches to Password Authenticated Key Exchange Based on RSA In *Proc. of Asiacrypt '04*, LNCS 3329, pages 230–244. Springer-Verlag, Berlin, 2004.
31. 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.

A Proof of Theorem 6

Assume f_n is surjective, we prove it is a bijection by proving that (where $|S|$ for a set S denotes the cardinality of the set):

$$|\{-1,1\} \times Q_n| \leq |J_n|.$$

Let us denote with $\bar{J}_n$ the set of elements having Jacobi symbol -1 in $\mathbb{Z}_n^*$. We claim that $|J_n| \geq |\bar{J}_n|$. We prove this claim by considering the following function g. Let x be an element in $\bar{J}_n$ (if $\bar{J}_n$ is empty, the statement is trivial) we set $g : \bar{J}_n \to J_n$ as

$$g(y) = xy \bmod n.$$

It is easy to check that this function is injective. This fact immediately implies the claim. Moreover since $\bar{J}_n \cup J_n = \mathbb{Z}_n^\star$ and $\bar{J}_n \cap J_n = \emptyset$, it has to be the case that $|J_n| \geq |\mathbb{Z}_n^\star|/2$.

Observe that $|\{-1,+1\} \times Q_n| \leq |\mathbb{Z}_n^\star|/2$ simply because if n is a composite with (at least) two different prime factors every quadratic residue in $\mathbb{Z}_n^\star$ has at least four different square roots. This means that $|Q_n| \leq |\mathbb{Z}_n^\star|/4$, and thus $|\{-1,+1\} \times Q_n| \leq 2 \times |\mathbb{Z}_n^\star|/4 = |\mathbb{Z}_n^\star|/2$.

From the above is then clear that $|\{-1,+1\} \times Q_n| \leq |\mathbb{Z}_n^\star|/2 \leq |J_n|$.

For the second statement, assume that f_n is not surjective. Since it is a morphism the image set produced by f_n

$$G_n = f_n\langle\{-1,+1\} \times Q_n\rangle$$

is actually a subgroup of J_n (which of course is a group in the first place). By the well-known Lagrange's theorem, this fact implies that the order of G_n divides that of J_n, thus $|G_n| \leq |J_n|/2$ (note that being f_n not surjective, G_n has to be a proper subgroup of J_n). □

B Generation of Random Elements in J_n

Landau's Theorem [19] states that $\varphi(n) \geq n/\log_2(n)$. However better results exist [27],

$$\text{For } n \geq 5, \quad \varphi(n) \geq \frac{n}{6 \ln\ln n} \qquad \text{and for } n \geq 646, \quad \varphi(n) \geq \frac{n}{2.5 \ln\ln n}.$$

Therefore, if one first checks that n is in the correct range, that is between 2^{k-2} and 2^k, for $k > 12$, then

$$\varphi(n) \geq \frac{n}{2.5 \ln\ln n} \geq \frac{2^k}{10 \ln\ln n} \geq \frac{2^k}{3.5(2\log_2 k - 1)}.$$

For $k = 1024$, the average number of iterations is less than 67.

In this way it is possible to produce outputs that are uniformly distributed in $\mathbb{Z}_n^\star$. Furthermore, under the assumption that the elements with Jacobi symbols equal to $+1$ are uniformly distributed—which is widely believed to be true—, the average number of iterations required to obtain outputs uniformly distributed in the subset of elements in $\mathbb{Z}_n^\star$ having Jacobi symbol equal to $+1$, can be upper-bounded by $3.5 \times (2\log_2 k - 1)$.

As a consequence, one can set the parameters q and s (see Section 5) as $q = 2^{k-3}$, and $s = k^2/4\ln^2 k$ (using the standard approximation for the density of primes and the assumption that Blum primes represent half the number of the primes).

C Proof of Compositeness for SQRT-IPAKE

In this section we prove that the protocol `Prove-Composite` is actually a perfect zero-knowledge one.

Completeness. It is easy to check that the protocol is complete (i.e. a correct statement can always be verified successfully).

Soundness. We claim that a dishonest client can cheat only with probability $1/2^\ell$. Note that, in order to succeed, the prover should be able to convince the verifier that n is a composite modulus of the required form (i.e. with at least two different prime factors) while it is not. This means that an invalid n can be either $n = p$ or $n = p^r$ where p is prime. In both cases $\mathbb{Z}_n^\star$ is a cyclic group. It is a well known fact from number theory (see [18] for example) that in every finite cyclic group G the equation $x^d = a$ has $\gcd(d, ord(G))$ different solutions. In our case this means that, for each quadratic residue, the malicious prover can find only two different square roots and thus, in order to be successful, he has to guess all the random bits c_i sent by the verifier in step 3 [7]. This leads to an error probability of $1/2$ for each c_i and thus to global success probability of $1/2^\ell$.

Zero Knowledge The simulator $\mathcal{S}$ goes as follows. It receives on input a valid modulus n (which in our protocol can be seen as a common parameter). Since we are in the random-oracle model $\mathcal{S}$ is able to simulate the three oracles $\mathcal{H}_2$, $\mathcal{H}_3$ and $\mathcal{H}_5$. Actually, it needs to define some values for $\mathcal{H}_2$ and $\mathcal{H}_5$ only, so that it is able to answer future queries. Whenever a query $(n, \mathsf{seed}, \star)$, for any seed but the specific (valid) challenge n, is asked to either $\mathcal{H}_2$ or $\mathcal{H}_5$, $\mathcal{S}$ anticipates the entire execution of the corresponding protocol: for $i = 1, \ldots, \ell$, it chooses at random

$$\alpha_{i,0}, \alpha_{i,2} \overset{R}{\leftarrow} \mathbb{Z}_n^\star \quad c_i \overset{R}{\leftarrow} \{0,1\} \quad \beta_i \overset{R}{\leftarrow} \{-1,+1\}$$

and sets

$$\alpha_{i,1} = -\alpha_{i,0} \bmod n \quad \alpha_{i,3} = -\alpha_{i,2} \bmod n \quad y_i = \alpha_{i,2c_i}^2 \beta_i \bmod n,$$

$$h_{i,j} = \mathcal{H}_3(n, \alpha_{i,j}), \text{ for } j = 0, \ldots, 3.$$

It finally defines

$$\mathcal{H}_2(n, \mathsf{seed}, i) \leftarrow y_i, \text{ for } i = 1, \ldots, \ell, \quad \mathcal{H}_5(n, \mathsf{seed}, \{h_{i,j}\}) \leftarrow c_1 \ldots, c_\ell.$$

It is clear that with this simulation of the random oracles, the simulator can perfectly simulate the view of Bob. Furthermore, this simulation works even if one uses a fixed modulus n: several concurrent proofs can be simulated since $\mathcal{S}$ can answer successfully all the queries (by construction).

[7] Note that for this argument to be correct it is crucial that Bob actually checks that the $h_{i,j}$ (step 1 in the verification process) are all different. Otherwise Alice could just use the two square roots she owns and simply duplicate them

D Other Concrete Examples

D.1 Diffie-Hellman

The most natural instantiation is of course with the Diffie-Hellman family. The obtained protocol is presented in Fig. 8. It is very similar to one of the AuthA variants [3], or to PAK [6]. Note that the

$$
\begin{array}{rcl}
\underline{\textit{Alice}} & & \underline{\textit{Bob}} \\
\multicolumn{3}{c}{\text{Shared password: } pw} \\
\multicolumn{3}{c}{\text{Shared public elements: } g \in \mathbb{G}, \text{ of prime order } q} \\
\multicolumn{3}{c}{\mathcal{G} : \mathbb{G} \times \mathsf{Password} \to \mathbb{G}} \\
\mathsf{accept} \leftarrow \mathsf{false} & & \mathsf{accept} \leftarrow \mathsf{false} \\
 & & \mathsf{terminate} \leftarrow \mathsf{false} \\
f \overset{R}{\leftarrow} \mathbb{Z}_q^\star, F \leftarrow g^f & \xrightarrow{\mathsf{Alice}, F} & F^q = 1? \\
 & & x \overset{R}{\leftarrow} \mathbb{Z}_q, X \leftarrow g^x, Y \leftarrow F^x \\
\mathsf{PW} \leftarrow \mathcal{G}(F, pw) & & \mathsf{PW} \leftarrow \mathcal{G}(F, pw) \\
Y' \leftarrow \hat{Y}/\mathsf{PW} & \xleftarrow{\mathsf{Bob}, \hat{Y}} & \hat{Y} \leftarrow Y \times \mathsf{PW} \\
X' \leftarrow Y'^{1/f} & \xrightarrow{\mathsf{Auth}} & \\
 & & \mathsf{Auth}\ \text{valid?} \Rightarrow \mathsf{accept} \leftarrow \mathsf{true} \\
\mathsf{accept} \leftarrow \mathsf{true} & & \mathsf{terminate} \leftarrow \mathsf{true} \\
\multicolumn{3}{c}{\mathsf{Auth} \leftarrow \mathcal{H}_1(\mathsf{Alice}\|\mathsf{Bob}\|F\|\hat{Y}\|pw\|X)} \\
\multicolumn{3}{c}{sk \leftarrow \mathcal{H}_0(\mathsf{Alice}\|\mathsf{Bob}\|F\|\hat{Y}\|pw\|X)}
\end{array}
$$

Fig. 8. Diffie-Hellman – **IPAKE** protocol

general security result, here, seems to lead to the following variant of the standard Diffie-Hellman problem: given g, $F = g^f$ and $Y = g^{fx}$, find $X = g^x$. However a small change suffices to obtain back to the classical Diffie-Hellman problem: $h \leftarrow F = g^f$, $X' \leftarrow g = h^z$ where $z = 1/f$, and $Y' \leftarrow Y = g^{fx} = h^x$, find $Z' = h^{zx} = g^x = X$.

D.2 RSA

Description. For the case of RSA the function f is defined by a modulus n, with a universal exponent e, by $f(x) = x^e \bmod n$. For this function to be one-way, Alice is assumed to choose an RSA modulus of size k (and at least $k-2$), and thus the product of two exactly $k/2$-bit primes, between $2^{k/2-1}$ and $2^{k/2}$ (the set of which is denoted $\mathit{Prime}(k/2)$). From his side, Bob wants this function to be a permutation, which means that $\varphi(n)$ must be co-prime with respect to the exponent e. This fact has to be proven by Alice.

The protocol is instantiated by setting $X_f = Y_f = \mathbb{Z}_n^\star$, (the internal group law is multiplication in $\mathbb{Z}_n^\star$). The hash function $\mathcal{G}$ is supposed to output k-bit elements, uniformly distributed in $\mathbb{Z}_n^\star$ (see Appendix B).

The full protocol appears in Fig. 9. It is very similar to OKE (Open Key Exchange [20] and variants, like SNAPI [22, 21] and [31]). The main difference is the efficient proof for the co-primality of e and $\varphi(n)$, even for a small exponent e.

Proof of Correct Modulus. The proof about n consists in proving that n is in the correct range, and e is co-prime to $\varphi(n)$ (which can be done by proving that any element admits an e-th root). Methods provided in [22, 21] are very inefficient since they ask e to be a very large prime, in such a way that it is co-prime to any integer smaller than n, and thus to $\varphi(n)$. A much more efficient technique is proposed in [31], but with no complete security proof. Here we describe our solution.

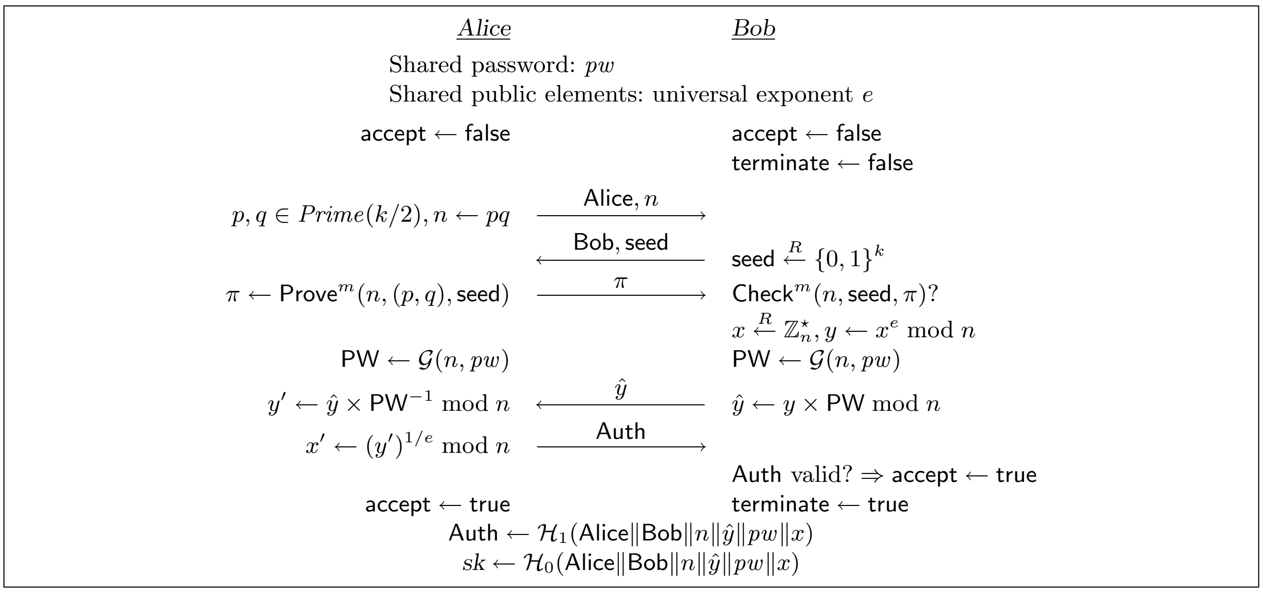
Alice
Bob
Shared password: pw
Shared public elements: universal exponent e
accept $\leftarrow$ false
accept $\leftarrow$ false
terminate $\leftarrow$ false
$p, q \in Prime(k/2), n \leftarrow pq$
$\xrightarrow{\mathsf{Alice}, n}$
$\xleftarrow{\mathsf{Bob}, \mathsf{seed}}$
$\mathsf{seed} \overset{R}{\leftarrow} \{0,1\}^k$
$\pi \leftarrow \mathsf{Prove}^m(n, (p,q), \mathsf{seed})$
$\xrightarrow{\pi}$
$\mathsf{Check}^m(n, \mathsf{seed}, \pi)$?
$x \overset{R}{\leftarrow} \mathbb{Z}_n^\star, y \leftarrow x^e \bmod n$
$\mathsf{PW} \leftarrow \mathcal{G}(n, pw)$
$\mathsf{PW} \leftarrow \mathcal{G}(n, pw)$
$y' \leftarrow \hat{y} \times \mathsf{PW}^{-1} \bmod n$
$\xleftarrow{\hat{y}}$
$\hat{y} \leftarrow y \times \mathsf{PW} \bmod n$
$x' \leftarrow (y')^{1/e} \bmod n$
$\xrightarrow{\mathsf{Auth}}$
Auth valid? $\Rightarrow$ accept $\leftarrow$ true
accept $\leftarrow$ true
terminate $\leftarrow$ true
$\mathsf{Auth} \leftarrow \mathcal{H}_1(\mathsf{Alice}\|\mathsf{Bob}\|n\|\hat{y}\|pw\|x)$
$sk \leftarrow \mathcal{H}_0(\mathsf{Alice}\|\mathsf{Bob}\|n\|\hat{y}\|pw\|x)$

Fig. 9. RSA – **IPAKE** protocol

Let ℓ be a security parameter, the protocol, for $n \leftarrow pq$, e such that $\gcd(e, \varphi(n)) = 1$, and a random seed seed, goes as follows, for $i \leftarrow 1$ to ℓ:

1. Alice computes $y_i = \mathcal{H}(n, \mathsf{seed}, i)$ (where $\mathcal{H}(n, \cdot)$ is a full-domain hash function onto $\mathbb{Z}_n^\star$, modeled as a random oracle);
2. Alice computes $x_i = \sqrt[e]{y_i} \bmod n$;

Alice then sends the x_i's to Bob whom accepts if for all $i = 1, \ldots, \ell$, $x_i^e \bmod n = \mathcal{H}(n, \mathsf{seed}, i)$.

Note that this costs roughly ℓ exponentiations (for the server Bob) with exponent e. It is not hard to see that the protocol is complete (i.e. if e is co-prime with $\varphi(n)$ every element is invertible). On the other hand if e is not co-prime with $\varphi(n)$ the probability that a random element is an e-residue can be bounded as follows. Let $d = \gcd(\varphi(n), e)$ a random element a is an e-residue modulo n with probability at most $1/d$. Thus the probability that ℓ random and independently chosen elements are all e-residues is at most $1/d^\ell$. Finally it is very easy to check that the protocol is a perfect zero-knowledge one, in the random-oracle model, even also in a concurrent setting, using a simulation of $\mathcal{H}$ as in the FDH signature proof [2]: for any query (n, seed, i) to $\mathcal{H}$, for the correct challenge n, one chooses r at random in $\mathbb{Z}_n^\star$ and defines $\mathcal{H}(n, \mathsf{seed}, i) \leftarrow r^e \bmod n$. This simulation works exactly for the same reasons as in the Appendix C.