Mutual Authentication for Low-Power Mobile Devices

Markus Jakobsson[1] and David Pointcheval[2]

[1] Information Sciences Research Center, Bell Labs
Murray Hill, New Jersey 07974
http://www.bell-labs.com/user/markusj
[2] Dépt d'Informatique, École Normale Supérieure
75230 Paris Cedex 05, France
http://www.di.ens.fr/users/pointche

Abstract. We propose methods for mutual authentication and key exchange. Our methods are well suited for applications with strict power consumption restrictions, such as wireless medical implants and contactless smart cards. We prove the security of our schemes based on the discrete log gap problem.
Keywords: Low power, medical informatics, mutual authentication, gap problem.

1 Introduction

Computers can be separated into *wired* and *wireless* devices, where no particular power restrictions are typically placed on the former, and the restrictions on wireless devices (typically cellular phones) relate mostly to the battery form factors. The use of wireless devices for medical applications – such as insulin meters and pacemakers – create a new category in terms of power restrictions, in which the power limitations are taken to their extreme. While traditional design of such devices have not relied on communication with nearby devices, there are great benefits associated with allowing this. Examples of such benefits include more accurate control of medical conditions, allowing doctors to constantly monitor health conditions; possibilities to detect inconsistent operation before it becomes a threat to the patient; and general collection of statistics for the improvement of the product.

At the same time, these are applications where errors and inconsistencies, whether due to interference or malice, may be fatal. In order to avoid security vulnerabilities, authentication methods and key exchange methods become crucial components in such systems. Authentication has traditionally been of an asymmetric nature, namely, an untrusted entity identifying itself to a trusted entity. With a trend towards decentralization, there is a greater need for symmetric or *mutual* authentication. The need for mutual authentication becomes particularly obvious in situations where users carry small wireless devices that monitor *and* control the operation of other wireless devices residing in the user's body.

P. Syverson (Ed.): FC 2001, LNCS 2339, pp. 178–195, 2002.

A situation with similar restrictions involves contact-free smart cards, whose advantages over standard smart cards range from the convenience they offer to their increased security – where the latter is due to the increased defense against power and timing attacks. Due to the absence of a local power source for such devices, electricity to perform computation is obtained by induction over a field moving in relation to the card. Only minute amounts of computation can be performed under such premises, severely restricting the choice of schemes that can be employed.

We propose two closely related schemes that allow for mutual authentication and key exchange, and which lower the computational requirements (and therefore the power consumption) by means of careful protocol design. One common technique we employ is that of precomputation, which allows for both the shifting of computation to another entity, and for a lower "peak performance" (and therefore a lower average power consumption). For applications in which devices are unable to perform such precomputation, and where the memory resources are limited, we show how trusted auxiliary devices can perform the computation and wirelessly upload this to the devices in question (after a successful mutual authentication, of course.) Our solutions have applications within a large set of seemingly unrelated fields, such as payment schemes, access control schemes, medical surveillance, and cellular billing schemes.

Outline: We begin by reviewing related work (section 2), followed by a discussion of our model (section 3). We then present two related schemes (section 4), both of which perform mutual authentication and key exchange. Not counting the amount of precomputation, we have that in the first scheme, the computational load for the client amounts to one modular multiplication and addition, while in the second scheme, we even avoid the modular reduction. Following this, we model the protocol and possible attacks on it (section 5), to prepare for the analysis of our solutions. We end by a careful security analysis of the two schemes, with further improvements (section 6 resp. section 7). We prove the schemes secure based on the *gap Diffie–Hellman* problem (which requires the standard Diffie–Hellman assumption.)

2 Related Work

2.1 Key Exchange and Mutual Authentication

Our paper hails back to the work on Diffie–Hellman key exchange [8], and the use of a shared key for purposes of authentication. While many methods can be employed in this later step – symmetric as well as asymmetric – we focus on asymmetric methods based on Schnorr signatures [17]. The reason is purely one of efficiency: Taking this approach, we can shift almost all the computational work to a preprocessing stage. One could use other methods for this second part, though, such as those proposed by Bellare and Rogaway [5].

Another direction for key exchange is that of Needham and Schroeder [11], later evolving into Kerberos (see [12] for a description.) There, a mutually trusted

third party is involved in the key exchange. Under such a trust model, an alternative to our protocols is to use a trusted third party for key exchange or precomputation. In the latter case, one could use a simple table based method, in which the TTP distributes pairwise matching lists to the participants. One part of an entry could correspond to a request, the second to a response, and a third to the key to be used. However, and as noted, such a solution requires the TTP to be *mutually trusted* by the parties involved, and not only trusted by its client. Another important difference is that such a solution is not necessarily easy to distribute. The (unilaterally) trusted third party in our solution – if used at all – may perform all the exponentiation using quorum action, and send the portions of the result to the device, which then computes the corresponding database entry.

Coming back to the former type of model, we have that a key exchange scheme (without TTP) involves two participants, a *client* and a *server*, who want to share a secret session key in order to achieve confidentiality. They therefore communicate on a public channel and eventually compute a value that they both know but which nobody else knows. Many security models have been defined to cover this kind of schemes. Of these, the following two models have received the most consideration:

- The first model was proposed by Bellare and Rogaway [4,5], and refined in [2] (furthermore considering dictionary attacks). Here, the adversary can interact with all the participants, with an aim to learn some information about one session key. Therefore, one tries to prove the indistinguishability of the session key (from a random key) for the adversary.
- The second model was proposed by Bellare, Canetti, and Krawczyk [1], and is based on the multi-party simulatability technique. This means that one first defines an idealized version of a key exchange scheme. Then, to prove that the real-world scheme is secure, one shows that any adversary in the real world has to behave like an adversary in the ideal world.

Shoup [19] recently showed that the two models (with some refinements) are equivalent in preventing active adversaries to break forward secrecy: An adversary who can see all the public communication and has access to all the session keys *but one*, cannot obtain any information about *that last* session key, even if he later learns the long-term secrets of the parties.

When parties have established a common secret session key, most of the key exchange protocols, such as the Diffie–Hellman [8] key exchange scheme using public keys, *implicitly* ensure that any party is really partnered (sharing the session key) with the party he wanted, or with nobody. Indeed, if an adversary uses the public key of Alice, Bob will run the key exchange process, and at the end he thinks that the actual session key is shared with Alice. However, the adversary cannot extract the session key from the communication. Therefore, nobody but Alice can be partnered with Bob as a result of this process.

Thus, apart from performing the key agreement, one usually wants to verify the actual partner. This latter property for a key exchange scheme is called *mutual authentication*. However, as presented in [2], an implicitly authenticated

key exchange protocol can be easily transformed into a scheme that provides mutual authentication, merely by adding one more flow, with a *key confirmation* step.

2.2 The Gap Problems

Very recently, Okamoto and Pointcheval [14] introduced a new class of problems to deal with the security of very efficient schemes. Informally, it considers the gap between a decision problem and its computational counterpart. More precisely, a gap-problem is a computational problem to solve given access to a decision oracle. Let us see what it means for the Diffie–Hellman family of problems, where all the elements belong in a group $\mathcal{G}$ of prime order q:

- *The Computational Diffie–Hellman Problem* (a.k.a. C-DH): given a triple (g, g^a, g^b), find the element $C = g^{ab}$.
- *The Decision Diffie–Hellman Problem* (a.k.a. D-DH): given a quadruple (g, g^a, g^b, g^c), decide whether $c = ab \bmod q$ or not.
- *The Gap Diffie–Hellman Problem* (a.k.a. G-DH): given a triple (g, g^a, g^b), find the element $C = g^{ab}$ with the help of a Decision Diffie–Hellman Oracle (which answers whether a given quadruple is a Diffie–Hellman quadruple or not).

Using the notation from the complexity theory, one could define the Gap Diffie–Hellman problem as the Computational Diffie–Hellman Problem with access to a Decision Diffie–Hellman oracle: $\mathsf{G\text{-}DH} = \mathsf{C\text{-}DH}^{\mathsf{D\text{-}DH}}$. Thereafter, some relations between these problems become clear: first, if the C-DH problem is easy, so is G-DH; secondly, if the G-DH problem is easy, then C-DH = D-DH, which is very unlikely. This latter remark justifies the current assumption that the Gap Diffie–Hellman problem is hard to solve. The assumption of its hardness seems very similar to the Decision Diffie–Hellman assumption. Thus, the class of the gap-problems can be considered a dual to the class of the decision-problems.

This class of problems is already believed to be yield to many secure and efficient schemes. Indeed, it helped to prove the security of an undeniable signature scheme, the very old and well-known scheme proposed by Chaum [7,6,14], for which no security proof was previously known. It is also the basis of very efficient chosen-ciphertext secure cryptosystems [13].

3 Model

We have two primary types of participants, the *client* and the *server*. Although we strive to limiting the computational burden for both of these participants, it is the *client* that we assume have the strictest limitations. It is the purpose of our protocols to allow a client and a server to perform mutual authentication and to establish a shared key.

Our schemes can be used with a standard public key infrastructure. The use of certificates is straightforward; however, we must assume that these are

verified beforehand to reduce the computational complexity. This only has to be assumed for the clients, given that the servers are assumed to have sufficient computational power to verify certificates. We note that this fits well into a model where many clients know of a few servers, but the servers do not know about any clients.

Furthermore, we may have *trusted devices*, who perform computation on behalf of clients and servers. A trusted device interacts with either a client or a server, but not both, as is only trusted by the entity it interacts with. The amount of trust that a device has to place in such a *trusted device* can be reduced by means of standard methods for distribution.

We assume that the entire communication network is managed by the adversary, who may schedule interactions arbitrarily, and who may inject and drop messages arbitrarily. We assume that all participants, and any adversary, can be modeled by poly-time Turing Machines.

Informally, we want our protocols to satisfy the following requirements.

- From a computational point of view, as said above, the on-line workload of the *client* must be minimal. Namely, we avoid the use of modular exponentiation, and avoid or reduce modular additions and multiplications.
- From the security point of view, we want to prevent active adversaries to learn any information about a session key. Forward-secrecy is also an important issue. However, under the above computational restriction, it seems impossible to achieve a forward-secrecy from both sides. We can assume a strong physical security level for the *server*, while the *client* may be a weak device. Therefore, the corruption of this device, and thus the leakage of the long-term secret key of the client, should not make public all the previous secret communication. Thus, the most important aspect is that all the session keys remain secret after the leakage of a *client* long-term key.

We will define the corresponding security requirements in more detail in the analysis section.

4 Solutions

We introduce two closely related protocols for mutual authentication and key exchange. While the protocols differ only on a few points in terms of their description, the security analysis differs substantially between the two. Still, the protocols are shown secure based on the same assumption: the intractability of the gap Diffie–Hellman problem.

Both protocols are based on the Diffie–Hellman key distribution scheme [8] together with the Schnorr's authentication scheme [17] (and the GPS scheme for the optimized version [9,16].) Thanks to the latter, much precomputation can be performed so that no on-line computation is required of the client. Therefore, the client can be any low-cost device.

The first scheme is presented in figure 1, the second differs on only a few points:

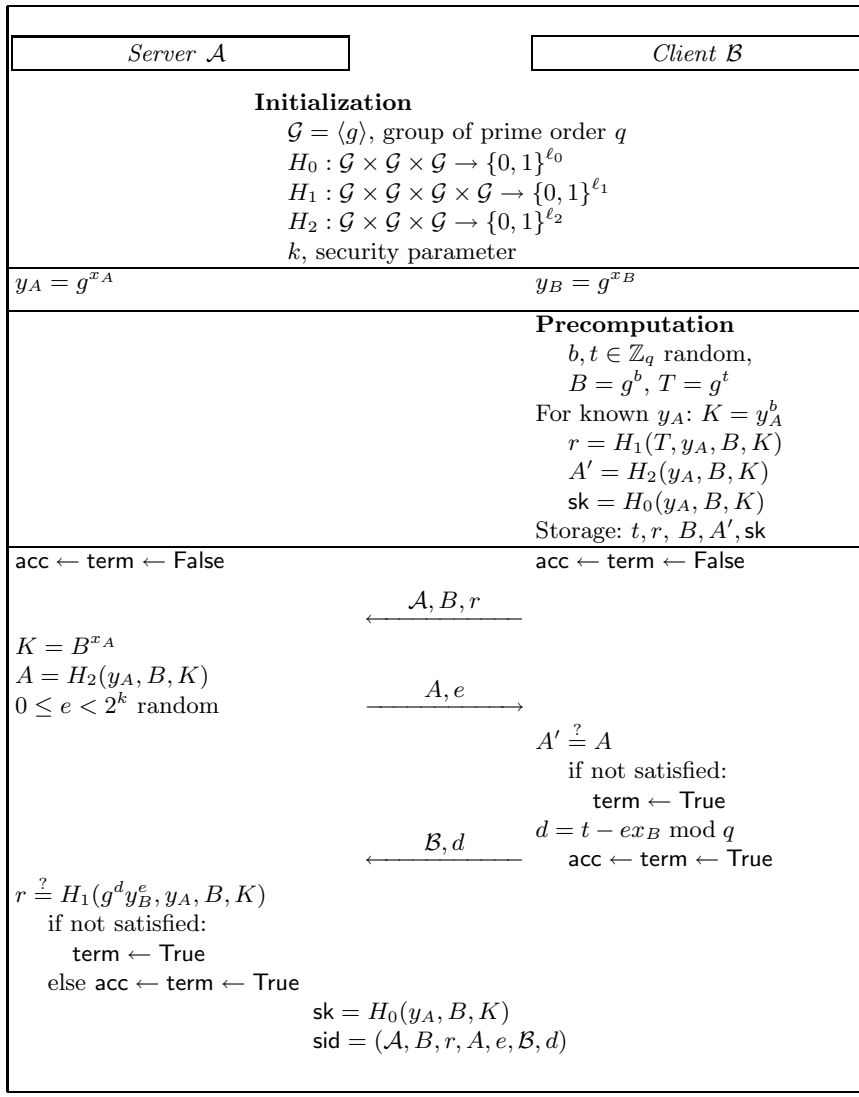

Fig. 1. Mutual Authentication

- It introduces a new security parameter, k'.
- Instead of selecting t uniformly at random from $\mathbb{Z}_q$, t is selected uniformly at random from $\mathbb{Z}_{q'}$, where $q' = q\,2^{k+k'}$.
- Instead of computing d as $d = t - ex_B \bmod q$, it is computed as $d = t - ex_B$. (Note the absence of the modular reduction.)

5 Modeling the Protocol

For this proof of security, we use the Bellare and Rogaway model [4,5] revisited by Shoup [19] to handle the forward-secrecy. In this model (see figure 2), any

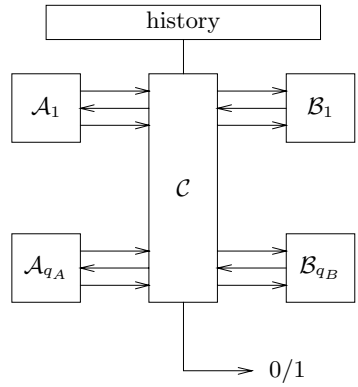

Fig. 2. Security Model

instance of each party, $\mathcal{A}$ or $\mathcal{B}$, is seen as an oracle. At the end of each protocol, when any party $\mathcal{U}_i$ has accepted, he gets a session key, denoted by sk_U^i, and a session ID, denoted by sid_U^i which is the concatenation of all the flows. The session ID's are made public, while the session keys clearly remain secret. Indeed, the session keys are the common secret shared by the two parties at the end of the protocol. The session ID's have a technical significance: they are used to define partnership. The partner of a party is an instance which has a similar session ID. Since the session ID's are public, the partnership is also public. With such a definition of partnership, one can remark that a party may have many partners, although we will show that it is very unlikely.

The adversary can interact, as a man-in-the-middle, with the parties, or more formally with many instances of them ($\mathcal{A}_i$ for the server and $\mathcal{B}_j$ for the client) as many times as he wants in a concurrent way. He can ask them the following queries

- $\mathsf{Send}\,(\mathcal{U},\ i,\ \mathit{string})$ – which means that the adversary sends the message *string* to the oracle $\mathcal{U}_i$ (either a server or a client). The oracle makes some computation according to the protocol and gives the answer back.
- $\mathsf{Reveal}\,(\mathcal{U},\ i)$ – if the oracle $\mathcal{U}_i$ has accepted (the tag acc has been set to True), he returns the session key sk_U^i. It models the misuse of a session key by the parties after having established it.
- $\mathsf{Test}\,(\mathcal{U},\ i)$ – if the oracle $\mathcal{U}_i$ has accepted, one tosses a coin b. If $b = 1$ then the session key sk_U^i is returned, else a random string is returned. The aim of the attack is then to guess this bit b. Therefore, there are trivial restrictions about this query:

  - it can just be asked once;
  - no Reveal-query has been asked to $\mathcal{U}_i$;
  - no Reveal-query has been asked to $\mathcal{V}_j$, where $\mathcal{V}_j$ is partnered with $\mathcal{U}_i$.
- Corrupt $(\mathcal{U})$ – in order to deal with the forward-secrecy, one allows the adversary to corrupt the parties. Then, he obtains the secret key (the long-term secret key x_U) of the corrupted party $\mathcal{U}$. Therefore, the Test-query will have to be asked to a party which had accepted before any corruption.

The above game, with the Test-query, just deals with the key agreement property but not with authentication. We will say that the protocol provides mutual authentication if no instance accepts and not exactly one partner exists. Otherwise, it would mean that the adversary has impersonated a party. More precisely, if an instance $\mathcal{A}_i$ of the server accepts with no partner, it means that the adversary had impersonated the client, and therefore broken the client-to-server authentication.

In the other direction, if an instance $\mathcal{B}_j$ of the client accepts with no partner, it means that the adversary had impersonate the server, and therefore broken the server-to-client authentication.

A key exchange protocol guarantees mutual authentication if for any adversary, her probabilities in breaking the client-to-server authentication or the server-to-client authentication are both negligible. This is usually guaranteed by implicit authentication together with key confirmations from both parties [2].

6 Analysis of the First Scheme

6.1 Presentation

This section deals with the security of the scheme presented in figure 1. We prove that it achieves the security requirements:

- an adversary cannot learn any information about a session key which has not been revealed. This is proven by the fact that any adversary can just obtain a negligible advantage in guessing the bit b involved in the Test-query;
- an adversary cannot impersonate any of the parties, which guarantees the mutual authentication;
- forward-secrecy is ensured as long as the server is not corrupted.

As usual, some assumptions have to be made to provide the security result. The following proof just runs in the random oracle model [3] and assume the intractability of the gap Diffie–Hellman problem [14].

Indeed, we cannot hope to weaken the computational assumption, but can prove that it is sufficient.

6.2 The Gap Diffie–Hellman Problem: A Necessary Assumption

First, let us specify more formally the Diffie–Hellman problems we will use. In the protocol, $\mathcal{G}$ is any group of prime order q. For any pair (g, h) of $\mathcal{G}$-elements, we define the following Diffie–Hellman problems, which are particular instances of the general problems presented previously.

- $\mathsf{C\text{-}DH}_{g,h}$: given an element a, find the element $b = \mathsf{C\text{-}DH}(g, h, a)$.
- $\mathsf{D\text{-}DH}_{g,h}$: given a pair (a, b), decide whether $b = \mathsf{C\text{-}DH}(g, h, a)$, which is equivalent to decide whether $\mathsf{D\text{-}DH}(g, h, a, b)$ is true or not.
- $\mathsf{G\text{-}DH}_{g,h}$: given an element a, find the element $b = \mathsf{C\text{-}DH}(g, h, a)$ with the help of a $\mathsf{D\text{-}DH}_{g,h}$ oracle.

First, it is clear that if the discrete logarithm problem can be broken, then this authentication scheme is no longer secure. Furthermore, for the server, the computational Diffie–Hellman problem $\mathsf{C\text{-}DH}_{g,y_A}$ is enough to be broken so that the security of the overall scheme vanishes. However, one may also remark that the adversary has access to a kind of oracle $\mathsf{D\text{-}DH}_{g,y_A}$ that answers to any query $\mathsf{D\text{-}DH}_{g,y_A}(a, b)$, by saying whether $b = \mathsf{C\text{-}DH}(g, y_A, a)$ or not, for any pair (a, b) of her choice: indeed, the adversary chooses a random r and sends $\mathcal{A}, a, r$ to the server. This latter answers A, e. The adversary stops the game and simply checks whether $A = H_2(y_A, a, b)$, which answers the $\mathsf{D\text{-}DH}_{g,y_A}(a, b)$ query.

Therefore, if one can break the Gap Diffie–Hellman problem $\mathsf{G\text{-}DH}_{g,y_A}$, which is exactly to compute $\mathsf{C\text{-}DH}_{g,y_A}(B) = \mathsf{C\text{-}DH}(g, y_A, B)$ for a non-negligible part of B with non-negligible probability, with an access to a $\mathsf{D\text{-}DH}_{g,y_A}$ oracle, which answers whether $C = \mathsf{D\text{-}DH}(g, y_A, B)$ or not, for any pair (B, C), then one can use the server for simulating the $\mathsf{D\text{-}DH}_{g,y_A}$ oracle, as shown above.

Now, let us prove that this mathematical assumption is enough for the security of this scheme, which would prove the equivalence of the security and the Gap Diffie–Hellman problem [14]. Let us do it step by step. Whereas we want to prove the security of the key exchange protocol and of the mutual authentication, we do not proceed as usual. Indeed, we first study the client-to-server authentication, then the security of the key agreement (no leakage of information about any session key) and finally we complete the mutual authentication by proving the server-to-client authentication.

In all the following claims and proofs, we denote by

- q_A (resp. q_B), the number of instances of the server (resp. client) involved in the game;
- ℓ_0, ℓ_1 and ℓ_2, the output size of the oracles H_0, H_1 and H_2;
- q_0, q_1 and q_2, the number of queries asked to the oracles H_0, H_1 and H_2;
- q_H, the total number of queries asked to the oracles H_0, H_1 and H_2;
- k, the size of the challenge e.

6.3 Client-to-Server Authentication

Let us first deal with the authentication of the parties to each other. In this aim, we denote by $\mathsf{Event}^{\mathsf{c2s}}$ the event that, at the end of the attack, there exists an instance $\mathcal{A}_i$ of the server which has accepted without exactly one partner. This event defines the violation of the client-to-server authentication. Respectively, we denote by $\mathsf{Event}^{\mathsf{s2c}}$ the event that, at the end of the attack, there exists an instance $\mathcal{B}_j$ of the client which has accepted without exactly one partner. This latter event defines the violation of the server-to-client authentication.

The following lemma states that the protocol provides client-to-server authentication, relative to the discrete logarithm problem.

Lemma 1. *Let us assume that an adversary can violate the client-to-server authentication with probability ε within a time bound t. Then the discrete logarithm can be solved within an expected time*

$$t' \leq t \times \left(\frac{1}{\nu} + \left(\frac{\nu}{4q_A} - \frac{1}{2^k} \right)^{-1} \right), \text{ where } \nu = \varepsilon - \left(\frac{1}{2^{\ell_1}} \times \left(\frac{q_B^2}{q} + q_1^2 \right) + \frac{q_A q_1}{q} \right).$$

Proof. Let us assume that, for some ν,

$$\varepsilon = \Pr[\mathsf{Event}^{\mathsf{c2s}}] \geq \nu + \frac{1}{2^{\ell_1}} \times \left(\frac{q_B^2}{q} + q_1^2 \right) + \frac{q_A q_1}{q}.$$

First, one can easily simulate any client $\mathcal{B}$ instance without the secret key, thanks to the random oracle used to commit the first flow. On the other hand, there is no need to simulate the server instances, since we have the secret key. Let see the figure 3. On may remark that this simulation is perfectly indistinguishable from a real game, excepted in the case the definition $H_1(g^d y_B^e, y_A, B, K) \leftarrow r$ cannot be done in the $\mathsf{Send}\,(\mathcal{B},\ j,\ (A, e))$-query. Indeed, H_1 may have already been defined at that point before. But since d is randomly chosen in $\mathbb{Z}_q$, the simulation fails with probability less than $q_A q_1/q$.

Therefore, one can consider this simulation as the game to study, which is indistinguishable from a real game. Thus, one can remark that the probability for an $\mathcal{A}_i$ to have many partners is bounded by $q_B^2/q2^{\ell_1}$, since B and r are randomly chosen by the client instances. Furthermore, we condition, using Pr_H, all the probabilities to the event $\neg\mathsf{Event}^{\mathsf{ColH}}$, where $\mathsf{Event}^{\mathsf{ColH}}$ denotes a collision for H_1. Therefore,

$$\begin{aligned} \nu + \frac{q_1^2}{2^{\ell_1}} \leq \varepsilon - \frac{q_A q_1}{q} - \frac{q_B^2}{q \cdot 2^{\ell_1}} \leq \Pr[\exists i \ \ \mathsf{Event}_i] &\leq \mathsf{Pr}_H[\exists i \ \ \mathsf{Event}_i] + \Pr[\mathsf{Event}^{\mathsf{ColH}}] \\ &\leq \mathsf{Pr}_H[\exists i \ \ \mathsf{Event}_i] + \frac{q_1^2}{2^{\ell_1}}, \end{aligned}$$

where Event_i denotes the event that, at the end of the attack in the simulated game, the instance $\mathcal{A}_i$ has accepted without any partner. Then $\mathsf{Pr}_H[\exists i\ \mathsf{Event}_i]$ is lower-bounded by ν. The end of the proof works exactly as the security proof of the signature schemes studied in [15], thanks to the *forking lemma*. Using this technique, we make a fork on the execution $\mathsf{sid} = (\mathcal{A}, B, r, A, e, \mathcal{B}, d)$, on which occurred the violation of the client-to-server authentication, by changing e into e' at the right time. We then obtain a new violation on the execution $\mathsf{sid}' = (\mathcal{A}, B, r, A', e', \mathcal{B}, d')$. This uses the same values for B and r: note that the correctness of B, and the knowledge of K, are both verified in the test $r = H_1(g^d y_B^e, y_A, B, K)$[1]. More precisely, let us group inside the set $\mathcal{I}$ all the

[1] We note that the absence of such a construction would allow a reuse of transcripts, which opens up to serious abuse. We refer to [20] for a description of how such vulnerabilities can be taken advantage of. Therein, a weakness of a previous version of our protocol is described and exploited.

Initialization	
Input	g and y
Keys	$x_A \in \mathbb{Z}_q$, $y_A = g^{x_A}$, $y_B \leftarrow y$
Hash functions H_0, H_1, H_2	
$H_0(\star)$, $H_1(\star)$, $H_2(\star)$	if the value is not determined at that point, one chooses a random value in the corresponding range and returns it.
Any hash value used below is implicitly obtained with this simulation, unless something else is specified.	
Instance $\mathcal{A}_i$ of $\mathcal{A}$	
$\mathsf{Send}\ (\mathcal{A},\ i,\ (\mathcal{A}, B, r))$	one computes $K = B^{x_A}$ and $A = H_2(y_A, B, K)$. Then one chooses a random challenge $0 \leq e < 2^k$ and returns (A, e).
$\mathsf{Send}\ (\mathcal{A},\ i,\ (\mathcal{B}, d))$	one checks whether $r = H_1(g^d y_B^e, y_A, B, K)$. If satisfied, one accepts and terminates, else one just terminates, while still not accepting.
Instance $\mathcal{B}_j$ of $\mathcal{B}$	
$\mathsf{Send}\ (\mathcal{B},\ j,\ \text{“start”})$	one chooses random $b \in \mathbb{Z}_q$ and $r \in \{0,1\}^{\ell_1}$. Then, one computes $B = g^b$, $K = y_A^b$ and returns (B, r).
$\mathsf{Send}\ (\mathcal{B},\ j,\ (A, e))$	one checks whether $A = H_2(y_A, B, K)$. If satisfied, one chooses a random $0 \leq d < q$, computes $T = g^d y_B^e$, defines $H_1(T, y_A, B, K) \leftarrow r$ and returns $(\mathcal{B}, d)$ while accepting and terminating, else one just terminates, while still not accepting.
Other queries	
$\mathsf{Reveal}\ (\mathcal{U},\ i)$	if the oracle $\mathcal{U}_i$ has accepted, one returns the corresponding $H_0(y_A, B, K)$.
$\mathsf{Test}\ (\mathcal{U},\ i)$	if $\mathcal{U}_i$ has accepted, one flips a coin and either returns the corresponding $H_0(y_A, B, K)$ or a random string.

Fig. 3. Game A: Client-to-server authentication

most likely indices i: $\mathcal{I} = \{i \mid \mathsf{Pr}_H[\mathsf{Event}_i \mid \mathsf{Event}^{\mathsf{c2s}}] \geq 1/2q_A\}$. Then one can easily prove that we have $\mathsf{Pr}_H[\exists i \in \mathcal{I},\ \mathsf{Event}_i] \geq 1/2$.

Let us call $\mathsf{Event}_i^{\mathsf{Partial}}$ the event defined by the following property: when the instance $\mathcal{A}_i$ receives the $\mathsf{Send}\ (\mathcal{A},\ i,\ (\mathcal{A}, B, r))$ query,

$$\mathsf{Pr}_H[\mathsf{Event}_i \mid \mathsf{Event}_i^{\mathsf{Partial}}] \geq \nu/4q_A.$$

Then, using the *splitting lemma* [15] one can claim that for any index $i \in \mathcal{I}$, $\mathsf{Pr}_H[\mathsf{Event}_i^{\mathsf{Partial}} \mid \mathsf{Event}_i] \geq 1/2$. Indeed,

$$\begin{aligned}\mathsf{Pr}_H[\mathsf{Event}_i] &= \mathsf{Pr}_H[\mathsf{Event}_i \wedge \mathsf{Event}^{\mathsf{c2s}}] \\ &= \mathsf{Pr}_H[\mathsf{Event}_i \mid \mathsf{Event}^{\mathsf{c2s}}] \times \mathsf{Pr}_H[\mathsf{Event}^{\mathsf{c2s}}] \geq \frac{1}{2q_A} \times \nu.\end{aligned}$$

Therefore, if one runs the attack, until the event $\mathsf{Event}^{\mathsf{c2s}}$ occurs, which requires an expected number of iterations bounded by $1/\nu$. In that case, with probability of $1/2$, we furthermore have Event_i with an instance $i \in \mathcal{I}$. That event

means that the adversary (since it is not an instance of $\mathcal{B}$) has answered d which satisfies $r = H_1(g^d y_B^e, y_A, B, K)$. Therefore, with probability $1/2$, $\mathsf{Event}_i^{\mathsf{Partial}}$ occurs too. One rewinds the game up to the $\mathsf{Send}\ (\mathcal{A},\ i,\ (\mathcal{A}, B, r))$ query, answering with a random challenge e'. One resumes and rewinds with new challenges e' until another event $\mathsf{Event}^{\mathsf{c2s}}$ occurs, or at most $(\nu/4q_A - 1/2^k)^{-1}$ times. If $\mathsf{Event}_i^{\mathsf{Partial}}$ occurred, we obtain a second success Event_i with probability greater than $1/2$.

Globally, after at most $1/\nu + (\nu/4q_A - 1/2^k)^{-1}$ iterations of the game, we have obtained two answers d, d' to two distinct challenges $e \neq e'$ with probability greater than $1/8$, for the same $(\mathcal{A}, B, r, A)$.

Thanks to $e \neq e', d, d'$, since we have assumed that no collision has been found for H_1, we have the relation $g^e y_B^d = g^{e'} y_B^{d'}$, which leads to the discrete logarithm of y_B in basis g. □

Let us postpone the study of mutual authentication and study right now the security of the key agreement. Indeed, the proof relies on the previous result, and will be useful for the server-to-client authentication.

6.4 Key Agreement

Theorem 2. *Let us assume that an adversary can guess the bit involved in the* Test*-query with advantage ε within a time bound t. Then the computational Diffie–Hellman problem can be solved with probability $\varepsilon' \geq \varepsilon/2 - p_{\mathsf{c2s}}$, within almost the same time, where p_{c2s} is the maximal probability for an adversary to violate the client-to-server authentication within a time bound t (*cf. *Lemma 1), with at most q_H queries to the decision Diffie–Hellman oracle. Thus the security relies on the gap Diffie–Hellman problem.*

Proof. Let us first remark that because of the randomness of the hash function, to gain any advantage in guessing correctly the coin involved in the Test-query, the adversary must ask the query (y_A, B, K) to H_0: $\Pr[\mathsf{AskK}] \geq \mathsf{Adv}/2$, where AskK denotes the event that the query (y_A, B, K) corresponding to the sid of the Test-query has been asked to H_0. Therefore, because of the constraints on the Test-query,

$$\Pr[\mathsf{AskK} \wedge \exists i\ \mathsf{Test}\ (\mathcal{A},\ i) \wedge \mathsf{Event}^{\mathsf{c2s}}] + \Pr[\mathsf{AskK} \wedge \exists i\ \mathsf{Test}\ (\mathcal{A},\ i) \wedge \neg\mathsf{Event}^{\mathsf{c2s}}] + \Pr[\mathsf{AskK} \wedge \exists j\ \mathsf{Test}\ (\mathcal{B},\ j)] \geq \mathsf{Adv}/2.$$

If one denotes by p_{c2s} the probability to break the client-to-server authenticity, one can claim that

$$\Pr[\mathsf{AskK} \wedge \exists i\ \mathsf{Test}\ (\mathcal{A},\ i) \wedge \neg\mathsf{Event}^{\mathsf{c2s}}] + \Pr[\mathsf{AskK} \wedge \exists j\ \mathsf{Test}\ (\mathcal{B},\ j)] \geq \mathsf{Adv}/2 - p_{\mathsf{c2s}}.$$

Let us now consider the simulation of the parties, as described on figure 4. Thanks to the Decision Diffie–Hellman Oracle $\mathsf{D\text{-}DH}_{g,\alpha}$, one can perfectly simulate all the parties and the random oracles. Indeed, the tables H_0^{DH}, H_1^{DH} and H_2^{DH} are managed using this decision Diffie–Hellman oracle, and record the answers of the oracles H_0, H_1 and H_2, when inputs are Diffie–Hellman triples.

The simulation may just fail, in the $\mathsf{Test}\ (\mathcal{A},\ i)$ query, since this latter simulation requires a client-partner, if the event $\mathsf{Event}^{\mathsf{c2s}}$ occurs. Anyway, with this simulation, the event "$(\exists i)\ \mathsf{Test}\ (\mathcal{A},\ i) \wedge \neg\mathsf{Event}^{\mathsf{c2s}}$" implies event "$(\exists j)\mathsf{Test}\ (\mathcal{B},\ j)$":

$$\Pr[\mathsf{AskK} \wedge (\exists j)\mathsf{Test}\ (\mathcal{B},\ j)] \geq \mathsf{Adv}/2 - p_{\mathsf{c2s}}.$$

Because of the simulation of $\mathcal{B}_j$, we have

$$\mathsf{Adv}/2 - p_{\mathsf{c2s}} \leq \Pr[\mathsf{AskK} \text{ for } (y_A = \alpha, B = \beta^b, K = \mathsf{C\text{-}DH}(g, y_A, B))].$$

Therefore, the AskK event says that $K = \mathsf{C\text{-}DH}(g, \alpha, \beta^b)$ can be extracted from the queries asked to the H_0 oracle, while verifying the correctness thanks to the Decision Diffie–Hellman Oracle (the $\mathsf{D\text{-}DH}_{g,\alpha}$), with probability greater than $\mathsf{Adv}/2 - p_{\mathsf{c2s}}$. Thus, $\mathsf{C\text{-}DH}(g, \alpha, \beta) = K^d$, where $d = b^{-1} \bmod q$.

To conclude the proof, one can just remark that if the Gap Diffie–Hellman problem $\mathsf{G\text{-}DH}_{g,y_A}$ is intractable, so do is the discrete logarithm problem too, which guarantees that p_{c2s} is small. □

6.5 Mutual Authentication

Since we have already proven the client-to-server authentication, we just need to prove the server-to-client authentication to ensure mutual authentication.

Lemma 3. *Let us assume that an adversary can violate the server-to-client authentication (without any violation of the client-to-server authentication) of the protocol with probability π within a time bound t. Then the computational Diffie–Hellman problem can be solved with probability π' within almost the same time, where*

$$\pi' \geq \pi - \left(\frac{q_B}{2^{\ell_2}} + \frac{q_B^2}{q}\right).$$

Proof. As we have seen above, the simulation presented on figure 4 is perfect unless the event $\mathsf{Event}^{\mathsf{c2s}}$ occurs. Therefore, let us study the event $\mathsf{Event}^{\mathsf{s2c}}$, knowing $\neg\mathsf{Event}^{\mathsf{c2s}}$. It means that at some point, after having sent $(\mathcal{A}, B = \alpha^b, r)$ and received (A, e), a client accepts the proof whereas it has not been produced by a server:

- either the adversary guessed the value A (probability less than $q_B/2^{\ell_2}$)
- or the value B occurred in an other session (probability less than q_B^2/q, since it is randomly chosen by the client)
- or the adversary has asked for (y_A, B, K) to the oracle H_2

Then

$$\Pr[\mathsf{Event}^{\mathsf{s2c}} \mid \neg\mathsf{Event}^{\mathsf{c2s}}] \leq \Pr\left[\begin{array}{l}(y_A, B, K) \text{ asked, with } y_A = \alpha,\\ B = \beta^b, K = \mathsf{C\text{-}DH}(g, y_A, B))\end{array}\right] + \frac{q_B}{2^{\ell_2}} + \frac{q_B^2}{q},$$

which completes the proof of the lemma. □

Initialization	
Input	g, α and β
Keys	$y_A \leftarrow \alpha$, $x_B \in \mathbb{Z}_q$, $y_B \leftarrow g^{x_B}$
Hash functions H_0, H_1, H_2	
$H_0(a,b,c)$	two different situations may appear. – $a = \alpha$ and $c = \mathsf{C\text{-}DH}(g,a,b)$, checked by the $\mathsf{D\text{-}DH}_{g,\alpha}$ oracle: if $H_0^{\mathsf{DH}}(a,b)$ has been defined, to say d, (which occurs iff H_0 has been defined to d in the point (a,b,c), then returns d, else, (*i.e.* H_0 is undefined at the point (a,b,c)) then one chooses a random value $d \in \{0,1\}^{\ell_0}$, defines $H_0^{\mathsf{DH}}(a,b) \leftarrow d$ and returns d. – otherwise: if H_0 is undefined at the point (a,b,c), then one chooses a random value in $\{0,1\}^{\ell_0}$ and returns it.
$H_1(T,a,b,c)$	same as for H_0, but using ℓ_1 and $H_1^{\mathsf{DH}}(T,a,b)$.
$H_2(a,b,c)$	same as for H_0, but using ℓ_2 and $H_2^{\mathsf{DH}}(a,b)$.
$H_0^{\mathsf{DH}}(a,b)$	if the query (a,b) has not been asked to H_0^{DH} then one chooses a random value in $\{0,1\}^{\ell_0}$ and returns it.
$H_1^{\mathsf{DH}}(T,a,b)$	same as for H_0^{DH}, but using ℓ_1, and queries of the form (T,a,b).
$H_2^{\mathsf{DH}}(a,b)$	same as for H_0^{DH}, but using ℓ_2.
Any hash value used below is implicitly obtained with this simulation, unless something else is specified. Furthermore, only the simulated parties have access to the H_0^{DH}, H_1^{DH} and H_2^{DH} oracles.	
Instance $\mathcal{A}_i$ of $\mathcal{A}$	
$\mathsf{Send}\ (\mathcal{A},\ i,\ (\mathcal{A},B,r))$	one asks for $A = H_2^{\mathsf{DH}}(y_A,B)$, chooses a random challenge $0 \le e < 2^k$ and returns (A,e).
$\mathsf{Send}\ (\mathcal{A},\ i,\ d)$	one checks whether $r = H_1^{\mathsf{DH}}(g^d y_B^e, y_A, B)$. If satisfied, one accepts and terminates, else one just terminates, while still not accepting.
Instance $\mathcal{B}_j$ of $\mathcal{B}$	
$\mathsf{Send}\ (\mathcal{B},\ j,\ \text{“}\mathsf{start}''\text{)}$	one chooses random $b,t \in \mathbb{Z}_q$ and computes $B = \beta^b$, $T = g^t$ as well as $r = H_1^{\mathsf{DH}}(T, y_A, B)$, and returns (B,r).
$\mathsf{Send}\ (\mathcal{B},\ j,\ (A,e))$	if $A = H_2^{\mathsf{DH}}(y_A,B)$ then one computes $d = t - ex_B \bmod q$ and returns d while accepting and terminating, else one just terminates, while still not accepting.
Other queries	
$\mathsf{Reveal}\ (\mathcal{U},\ i)$	if $\mathcal{U}_i$ has accepted, one returns the corresponding $H_0^{\mathsf{DH}}(y_A,B)$.
$\mathsf{Test}\ (\mathcal{B},\ j)$	if $\mathcal{U}_i$ has accepted, one flips a coin and either returns the corresponding $H_0^{\mathsf{DH}}(y_A,B)$ or a random string.
$\mathsf{Test}\ (\mathcal{A},\ i)$	let us denote by $\mathcal{B}_j$ the partner of $\mathcal{A}_i$ (abort if not uniquely defined), and run $\mathsf{Test}\ (\mathcal{B},\ j)$.
$\mathsf{Corrupt}\ (\mathcal{B})$	one returns x_B.

Fig. 4. Game B: Key agreement protocol

Thanks to both lemma 1 and lemma 3, one can easily claim the following theorem.

Theorem 4. *Let us assume that an adversary can violate the mutual authentication of the protocol with probability ε within a time bound t. Then the computational Diffie–Hellman problem can be solved within an expected time bound*

$$t' \leq t \times \left(\frac{1}{\nu} + \left(\frac{\nu}{4q_A} - \frac{1}{2^k} \right)^{-1} + \left(\frac{\varepsilon}{2} - \frac{q_B}{2^{\ell_2}} - \frac{q_B^2}{q} \right)^{-1} \right),$$

where

$$\nu = \frac{\varepsilon}{2} - \left(\frac{1}{2^{\ell_1}} \times \left(\frac{q_B^2}{q} + q_1^2 \right) + \frac{q_A q_1}{q} \right).$$

Proof. Simply adding results of both Lemmas 1 and 3, one gets the expected result, since $\varepsilon \leq \Pr[\mathsf{Event}^{\mathsf{ma}}] = \Pr[\mathsf{Event}^{\mathsf{c2s}}] + \Pr[\mathsf{Event}^{\mathsf{s2c}} \mid \neg\mathsf{Event}^{\mathsf{c2s}}]$, and therefore either $\Pr[\mathsf{Event}^{\mathsf{c2s}}] \geq \varepsilon/2$ or $\Pr[\mathsf{Event}^{\mathsf{s2c}} \mid \neg\mathsf{Event}^{\mathsf{c2s}}] \geq \varepsilon/2$. □

6.6 Forward Secrecy

This protocol furthermore provides partial forward-secrecy. Indeed, it is clear that if the server is corrupted, then all the session keys can be recovered from the transcript. However, the corruption of the client may not help to recover the session keys: the forward-secrecy just deals with the key agreement property which can be perfectly simulated by the game presented on figure 4. This simulation provides the Corrupt-query, since the client secret key is known. Then the theorem 2 still holds, since the Test-query has to be asked for a session which occurs before the corruption.

7 Improvements

7.1 Analysis of the Second Scheme

Without the q-modular reduction, the simulation of the client-to-server authentication, while choosing $0 \leq d < q \cdot 2^{k+k'}$, is not perfect [16]. However the distance of the distribution of the transcripts is less than $1/2^{k'}$ (statistical indistinguishability). Therefore, all the security results still remain, under the condition that $1/2^{k'}$ is negligible.

7.2 Hash Functions

Using the proof technique proposed by Girault and Stern [10], one can still prove the client-to-server authentication even with a short hash function H_1, just considering the multi-collision resistance. Indeed, if one can avoid ℓ-collisions for H_1, with probability greater than p_ℓ, then the lemma 1 is slightly modified, as follows.

Lemma 5. *Let us assume that an adversary can violate the client-to-server authentication with probability ε within a time bound t, then the discrete logarithm can be solved within an expected time*

$$t' \leq t \times \left(\frac{1}{\nu} + (\ell - 1) \times \left(\frac{\nu}{4q_A} - \frac{1}{2^k} \right)^{-1} \right), \text{ where } \nu = \varepsilon - \left(\frac{q_B^2}{2^{\ell_1} q} + \frac{q_A q_1}{q} + p_\ell \right).$$

The main modification appears in the time complexity, since in the forking lemma, one has to rewind many times to obtain ℓ values, so that at least 2 are distinct.

7.3 Size of the Parameters

One can use the following sizes for achieving a good security level, assuming that the adversary cannot ask more than $q_A, q_B \leq 2^{30}$ queries to the instance-oracles and $q_0, q_1, q_2 \leq 2^{64}$ queries to the random oracles:

- a 160-bit order q for the group $\mathcal{G}$ prevents baby-step/giant-step attacks [18] or any other generic attack. Then a convenient group as to be chosen to avoid any other kind of attack (*e.g.* $\mathcal{G} = \langle g \rangle \subset \mathbb{Z}_p^\star$, or an elliptic curve). The integer n will denote the bit-size of the encoding of the elements in $\mathcal{G}$;
- $k = k' = 64$ make the simulation indistinguishable but with a very small distance (less than 2^{-64});
- $\ell_1 = 80$, for providing a 5-collision resistant hash function; $\ell_2 = 64$ or 128; and ℓ_0, whatever needed for a session key, say 64.

7.4 Storage and Computation

With these parameters, the client can precompute anything required during the protocol:

- two exponentiations before knowing the server;
- one exponentiation and three hashings when he knows the server;

Then, he has to store

- B, a group element (of size n);
- t and r, where t is a $|q| + k + k' = 288$-bit long integer and r a 80-bit hash value;
- A' and sk, two 64-bit hash values.

The total memory required for one authenticated key agreement is $n + 496$ bits. Using an elliptic curve group, this is less than 82 bytes.

Thereafter, the client will just have to perform on-line,

- one test of equality between two 64-bit elements;
- one multiplication between a 64-bit and a 160-bit integers;
- one addition between a 224-bit and a 288-bit integers.

One can even decrease the storage-memory by choosing and storing $0 \leq t < q$, but computing $d = t + \rho \cdot q + e \cdot x_B$, where ρ is a random 128-bit element.

Then the storage-memory required for one authenticated key agreement is $n + 368$ bits. Using an elliptic curve group, this is less than 66 bytes. But one multiplication and one addition more have to be performed on-line.

8 Conclusion

In this paper, we have proposed a key exchange scheme which achieves mutual authentication and forward-secrecy (but just for the leakage of the client long-term key). The main interest of this scheme is the computational efficiency. Indeed, it requires the client to perform only a few additions and multiplications of short integers, and a few comparisons between 64-bit strings. The storage requirements are less than 70 bytes per process, which allows more than 15 precomputed tuples per kilobyte.

Acknowledgments

We are grateful to Wong and Chan [20] for detecting a vulnerability in the presented version. Our paper, as it appears here, takes their findings into consideration by ensuring that transcripts are not inappropriately reused. We refer to their publication for a detailed description of the importance of protecting against such attacks.

References

1. M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. In *Proc. of the 30th STOC*. ACM Press, New York, 1998.
2. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Eurocrypt 2000*, LNCS 1807, pages 139–155. Springer-Verlag, Berlin, 2000.
3. M. Bellare and P. Rogaway. Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. In *Proc. of the 1st CCS*, pages 62–73. ACM Press, New York, 1993.
4. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Crypto'93*, LNCS 773, pages 232–249. Springer-Verlag, Berlin, 1994.
5. M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: The Three Party Case. In *Proc. of the 27th STOC*. ACM Press, New York, 1995.
6. D. Chaum. Zero-Knowledge Undeniable Signatures. In *Eurocrypt'90*, LNCS 473, pages 458–464. Springer-Verlag, Berlin, 1991.
7. D. Chaum and H. van Antwerpen. Undeniable Signatures. In *Crypto'89*, LNCS 435, pages 212–216. Springer-Verlag, Berlin, 1990.
8. W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT–22(6):644–654, November 1976.
9. M. Girault. Self-Certified Public Keys. In *Eurocrypt'91*, LNCS 547, pages 490–497. Springer-Verlag, Berlin, 1992.

10. M. Girault and J. Stern. On the Length of Cryptographic Hash-Values used in Identification Schemes. In *Crypto'94*, LNCS 839, pages 202–215. Springer-Verlag, Berlin, 1994.
11. R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21:993–999, 1978.
12. B.C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, 32(9):33–28, September 1994.
13. T. Okamoto and D. Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In *RSA 2001*, LNCS 2020, pages 159–175. Springer-Verlag, Berlin, 2001.
14. T. Okamoto and D. Pointcheval. The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes. In *PKC 2001*, LNCS 1992, pages 104–118. Springer-Verlag, Berlin, 2001.
15. D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, 13(3):361–396, 2000.
16. G. Poupard and J. Stern. Security Analysis of a Practical "on the fly" Authentication and Signature Generation. In *Eurocrypt'98*, LNCS 1403, pages 422–436. Springer-Verlag, Berlin, 1998.
17. C.P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *Crypto'89*, LNCS 435, pages 235–251. Springer-Verlag, Berlin, 1990.
18. D. Shanks. Class Number, a Theory of Factorization, and Genera. In *Proceedings of the Symposium on Pure Mathematics*, volume 20, pages 415–440. AMS, 1971.
19. V. Shoup. On Formal Models for Secure Key Exchange. Technical Report RZ 3120, IBM Research, April 1999.
20. D.S. Wong and A.H. Chan. Efficient and Mutually Authenticated Key Exchange for Low-Power Computing Devices In *Asiacrypt 2001*, LNCS. Springer-Verlag, Berlin, 2001. To appear.