A Cryptosystem Based on Hidden Order Groups and Its Applications in Highly Dynamic Group Key Agreement

Amitabh Saxena and Ben Soh
Department of Computer Science and Computer Engineering
La Trobe University
VIC, Australia 3086

February 23, 2006

Abstract

Let G_1 be a cyclic multiplicative group of order n. It is known that the Diffie-Hellman problem is random self-reducible in G_1 with respect to a fixed generator g if $\phi(n)$ is known. That is, given $g, g^x \in G_1$ and having oracle access to a 'Diffie-Hellman Problem' solver with fixed generator g, it is possible to compute $g^{1/x} \in G_1$ in polynomial time. On the other hand, it is not known if such a reduction exists when $\phi(n)$ is unknown. We exploit this "gap" to construct a cryptosystem based on hidden order groups by presenting a practical implementation of a novel cryptographic primitive called *Strong Associative One-Way Function* (SAOWF). SAOWFs have interesting applications like one-round group key agreement. We demonstrate this by presenting an efficient group key agreement protocol for dynamic ad-hoc groups. Our cryptosystem can be considered as a combination of the Diffie-Hellman and RSA cryptosystems.

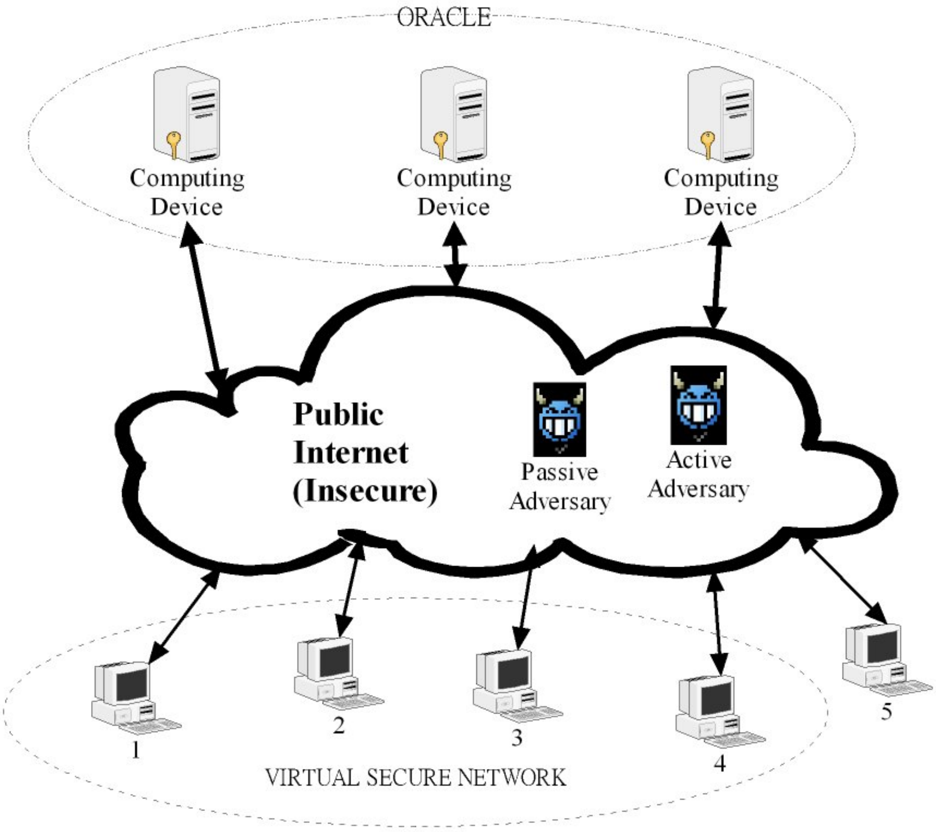
1 Introduction

The problem of efficient key agreement in ad-hoc groups is a challenging problem, primarily because membership in such groups does not follow any specified pattern. We refer the reader to [1, 2] for a survey of key agreement protocols for ad-hoc groups. In the literature, most group key agreement protocols are classified in three categories (a) Centralized, (b) Distributed (i.e. tree based) and (c) Fully Contributory. In this paper, we describe a method that is fully contributory, yet it uses a central controller. We elaborate this below.

The original two-party Diffie-Hellman key exchange [3] can be extended to fully contributory multi-party key exchange as demonstrated in [4] using the Group Diffie-Hellman (GDH) protocol. However, most protocols based on GDH require many rounds of sequential messages to be exchanged between members.

Centralized protocols, on the other hand have their own disadvantages; the central controller needs to maintain a large amount of state information for the groups it is managing. Our approach is to combine the two methods and design an efficient one-round key agreement protocol where the central controller does not maintain any state information.

Our protocol uses a central controller in computing the shared group key. However, the central controller is not responsible for key distribution and is only used as an "oracle" (i.e. a computing device) with public access. Users do not require secure channels in communicating with this oracle. Additionally, we provide a method to verify that the oracle is performing correctly. In our protocol, this oracle has some trapdoor information that can be efficiently used to compute partial public keys that are sent to users over an insecure public channel. Thus, our protocol can be directly converted into a de-centralized (or distributed) one simply by sharing this trapdoor information between a number of trusted authorities and allowing multiple "copies" of this oracle to function simultaneously. In effect, we present an entirely new way of doing secure group communication which is illustrated in figure 1.



In our model, secure group communication is facilitated by the Oracle. Assuming that public keys are known in advance, users can use this Oracle to compute a shared secret group key independently of the other users such that no active or passive adversary has the ability to compute the this key. Essentially, users use the oracle as a "verifiable computing device". We will use both the active and passive adversaries to relay messages between the users and the computing device.

Figure 1: Secure group communication in our model.

Our basic idea arises due to the key agreement protocols of Rabi and Sherman [5] based on hypothetical primitives which we briefly discuss in the next section.

2 Preliminaries

Rabi and Sherman first proposed the idea of one-round group key agreement using hypothetical cryptographic primitives called Strong Associative One-Way Functions (SAOWFs) [5] and our group-key protocol is based on theirs. Before presenting our protocol, we first discuss some important properties of SAOWFs.

2.1 Strong Associative One Way Functions

Let G be a finite abelian group with respect to the operation $\star$. The mapping $f : G \times G \mapsto G$ defined by $f(a, b) = a \star b$ for $a, b \in G$ has the following properties:

1. $f(f(a, b), c) = f(a, f(b, c)) \forall a, b, c \in G$ [Associativity]

2. $f(a, b) = f(b, a) \forall a, b \in G$ [Commutativity]

3. There exists a unique identity element $i \in G$ such that $f(a, i) = a \ \forall a \in G$.

4. For each $a \in G$, there exists a unique element $b \in G$ such that $f(a, b) = i$. We say b is the inverse of a and denote it by a^{-1} when there is no ambiguity in the notation.

The above properties come for "free" in any abelian group. We now additionally want to enforce the following properties on f:

1. Computability: For all $a, b \in G$, $f(a, b)$ must be efficiently computable.

2. Strong Non-Invertibility: Let $a, b \stackrel{R}{\leftarrow} G$ and $c = f(a, b) \in G$. Given a, c, computing $b = f(c, a^{-1})$ must be infeasible.

A function of the above type is called a *Strong Associative One-Way Function* (SAOWF).[1] Although functions exhibiting such behavior in the *worst case* can be easily constructed under the $P \neq NP$ assumption, we require the above conditions to hold in the *average case* (significant to cryptography).

The strong non-invertibility condition requires that for any $c \stackrel{R}{\leftarrow} image(f)$, inverting f with respect to a **given** preimage a must be infeasible in the average case. However, this condition does not say anything about *weak non-invertibility*, which requires that computing **any** preimage of c must be infeasible. In fact, the results of [6] prove that there exists a one-way function that is strongly non-invertible but not weakly non-invertible.[2] In this paper, we do not enforce the weak non-invertibility requirement. Rather, we allow the function to be weakly invertible. It turns out that our construction of SAOWF is strongly non-invertible, yet it is weakly invertible. Our construction also demonstrates an example of a Group with Infeasible Inversion (GII) [8].

Clearly, the above restrictions imply that computing inverses in G must be infeasible. Since we are working in a finite group, the only way to achieve this is to keep the order of the group hidden. This is the main idea behind our construction.

2.2 Oracle Based Construction of SAOWFs

We demonstrate a practical implementation of SAOWFs using a technique called *Verifiable Oracles*. Informally, this is described as follows:

1. There is a central authority (the oracle) responsible for setting up the parameters of G.

2. Only the oracle has the ability to compute f. Thus we must give all instances of pairs (a, b) to the oracle to compute $f(a, b)$. The inputs must be of a special form and only the oracle has the ability to decide if the input pairs are valid or not.

3. In certain special cases (when the elements of G have been sampled by us), we can compute f directly without help from the oracle.

4. We can mask the pairs (a, b) by creating new pairs (a', b') and ask the oracle to compute $f(a', b')$. From the oracle's output, $f(a, b)$ can be directly computed. Additionally, even the oracle does not know the value of $f(a, b)$. We use Chaum's blinding technique [9] for this.

5. The output of the oracle can be verified.

[1] Most researchers differentiate between commutative and non-commutative SAOWFs. For simplicity, in this paper we drop the non-commutativity requirement and assume that all SAOWFs considered are necessarily commutative.

[2] For completeness, we also define a Weak Associative One-Way Function (WAOWF) as one which is weakly non-invertible. A WAOWF may not be a SAOWF and vice-versa. For a discussion on the worst case analysis of Associative One-Way Functions, see [7].

Such oracle based constructions are often called "black box" constructions and until now it has been an open problem to present a practical example of a black box (or oracle-based) construction of an SAOWF [8]. In this paper, we present the first practical example of a black box SAOWF using a cryptosystem based on hidden order groups. The construction is possible due to certain homomorphic properties of the group used in the Diffie-Hellman key exchange [3] and the the group used in the RSA cryptosystem [10] for appropriately chosen parameters.

3 Our Construction

Our construction makes use of composite order groups that support a bilinear map. The bilinearity is required to verify the outputs of the oracle for protection against active adversaries. On the other hand, if protection is only required against passive adversaries, our construction works with any finite field having a composite order multiplicative subgroup. We use the following notation.

3.1 Bilinear Pairings

Let G_1 and G_2 be two cyclic multiplicative groups both of the same composite order n such that computing discrete logarithms in G_1 and G_2 is intractable. A bilinear pairing is a map $\hat{e} : G_1 \times G_1 \mapsto G_2$ that satisfies the following properties:

1. *Bilinearity*: $\hat{e}(a^x, b^y) = \hat{e}(a, b)^{xy}\ \forall a, b \in G_1$ and $x, y \in \mathbb{Z}_n$.
2. *Non-degeneracy*: If g is a generator of G_1 then $\hat{e}(g, g)$ is a generator of G_2.
3. *Computability*: The map $\hat{e}$ is efficiently computable.

See, for example [11] for details on generating composite order bilinear maps for any given n that is square free. We will assume that $n = pq$ where p, q are large primes such that given the product $n = pq$, factoring n is intractable. Let g be some **fixed** generator of G_1. We define the following problems in G_1.

A. *Diffie-Hellman Problem (DHP$_{(g,G_1)}$)*: Given $g^x, g^y \in G_1$ output $g^{xy} \in G_1$.

B. *Decision Diffie-Hellman Problem (DDHP$_{(g,G_1)}$)*: Given $g^x, g^y, g^z \in G_1$ output 1 if $z = xy \in \mathbb{Z}_n$; otherwise output 0.

C. *Inverse Diffie-Hellman Problem (IDHP$_{(g,G_1)}$)*: Given $g^x \in G_1$ output $g^{1/x} \in G_1$.

Lemma 1. *DDHP$_{(g,G_1)}$ (the Decision Diffie-Hellman Problem) is easy*

Proof. Clearly, from the properties of the mapping $\hat{e}$ we have $z = xy \in \mathbb{Z}_n$ if and only if $\hat{e}(g, g^z) = \hat{e}(g^x, g^y)$. Thus, solving DDHP$_{(g,G_1)}$ is equivalent to computing the mapping $\hat{e}$ twice.

The security of our scheme depends on the following two hardness assumptions. We will return to the security of our construction in section 6.

1. IDHP$_{(g,G_1)}$ is intractable.
2. Breaking RSA with modulus n and public exponent e is intractable for anyone who does not know the factorization of n. That is, for some $e \in \mathbb{Z}^*_{\phi(n)}$ computing x from $x^e \in \mathbb{Z}^*_n$ is infeasible. We call this the RSA$_{(e,n)}$ problem.

Our goal is to construct an implementation of an SAOWF using this setup. The oracle is responsible for generating the system parameters.

3.2 Initial Setup (Parameter Generation)

1. The oracle $\mathcal{O}$ sets a security parameter τ and generates $p, q \xleftarrow{R} \mathbb{N}$ where p, q are large distinct primes of $\approx \tau$ bits each. Let $n = pq$. The oracle generates the parameters $(\hat{e}, G_1, G_2)$ where G_1, G_2 are descriptions of two groups both of order n and $\hat{e} : G_1 \times G_1 \mapsto G_2$ is a bilinear map as defined in section 3.1. It then picks a random $g \xleftarrow{R} G_1$ such that g is a generator of G_1.

2. The oracle computes $\phi(n) = (p-1)(q-1)$ and generates a pair $e, d \xleftarrow{R} \mathbb{Z}^*_{\phi(n)}$ such that $ed = 1 \in \mathbb{Z}^*_{\phi(n)}$. Here, (e, d) will be used exactly like an RSA key pair.

3. The oracle generates $\alpha \xleftarrow{R} \mathbb{Z}^*_n$, computes $h = g^\alpha \in G_1$ and $\beta = \alpha^e \in \mathbb{Z}^*_n$.

4. The system parameters $(\hat{e}, G_1, G_2, n, g, e, h, \beta)$ are made public in an authentic way. The oracle keeps d secret.

3.3 Constructing a SAOWF

We construct an SAOWF using the oracle in four steps:

1. First define the mapping $f_1 : \mathbb{Z}^*_n \mapsto \mathbb{Z}^*_n$ as $f_1(x) = x^e \in \mathbb{Z}^*_n$ for any $x \in \mathbb{Z}^*_n$. This mapping is the well known RSA function and is bijective. Thus the inverse mapping f_1^{-1} is also well defined (even if not efficiently computable).

2. Now consider the set $\mathbb{S} \subsetneq G_1$ defined as

$$\mathbb{S} = \{x | x = g^y \in G_1 \text{ for some } y \in \mathbb{Z}^*_n\}$$

Clearly, $|\mathbb{S}| = \phi(n) = |\mathbb{Z}^*_n|$ and $\mathbb{S}$ is exactly the set of elements of G_1 having order n. We will now attempt to define a group structure using $\mathbb{S}$.

3. Define the mapping $f_2 : \mathbb{Z}^*_n \mapsto \mathbb{S}$ as $f_2(y) = g^{f_1^{-1}(y)} \in G_1$ for any $y \in \mathbb{Z}^*_n$. We note that f_2 is bijective and so the inverse map f_2^{-1} is also well defined. We see that f_2 has the property $f_2(f_1(a)) = g^a$ or $f_2^{-1}(g^a) = f_1(a)$. Additionally,

   (a) Both $f_2(f_1(a))$ and $f_1(a)$ are efficiently computable but neither of $f_2(a)$ or $f_1^{-1}(a)$ are efficiently computable without knowledge of the factors of n.

   (b) For any $x \in \mathbb{S}$ computing $f_2^{-1}(x)$ is also infeasible if computing discrete logarithms in G_1 to base g is intractable (because an algorithm that computes f_2^{-1} can be directly converted into an algorithm that computes discrete logarithms in G_1 to base g).

4. Define the set $\mathbb{G} \subsetneq \mathbb{S} \times \mathbb{Z}^*_n$ as

$$\mathbb{G} = \{(x, y) | x = f_2(y)\}$$

We define a binary commutative operation $\star$ on $\mathbb{G}$ using the mapping $\star : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}$ as follows. Given any two pairs $A, B \in \mathbb{G}$, we let $A = (x_a, y_a)$ and $B = (x_b, y_b)$ where $x_a, x_b \in \mathbb{S}$ and $y_a, y_b \in \mathbb{Z}^*_n$. Then $C = (x_c, y_c) = A \star B$ is defined as follows:

$$y_c = y_a y_b \in \mathbb{Z}^*_n$$

and

$$x_c = {x_a}^{f_1^{-1}(y_b)} = g^{f_1^{-1}(y_a) f_1^{-1}(y_b)} = {x_b}^{f_1^{-1}(y_a)} \in G_1 \quad (1)$$

The reader can verify that $\star$ is associative and commutative and that $(g, 1) \in \mathbb{G}$ is the identity element. Also, since all the exponents are in $\mathbb{Z}^*_n$, every element of $\mathbb{G}$ is invertible with respect to $\star$ (i.e. every element has an inverse). In other words, $\mathbb{G}$ forms an abelian group with respect to the $\star$ operation. The order of this group $|\mathbb{G}| = \phi(n)$ is effectively hidden from anyone who does not know the factors of n.

For any $A \in \mathbb{G}$, let the symbol A^i denote $A \star A \star \ldots A$ (i times). The inverse of A is denoted by A^{-1}. It can be trivially verified that the following are also true: $A^i \star A^j = A^{i+j}$; $(A^i)^j = A^{ij}$; $A \star A^{-1} = A^0 = (g, 1)$ and; $(A^i \star B^j)^k = A^{ij} \star B^{jk}$, for all $A, B \in \mathbb{G}$ and all i, j, k in $\mathbb{Z}$.

3.4 Computing the SAOWF

We now enumerate some important properties of $(\mathbb{G}, \star)$ which enable us to construct an SAOWF that allows efficient "forward" computation without allowing the corresponding "reverse" computation.

1. $\mathbb{G}$ is efficiently sampleable. To sample from $\mathbb{G}$, first generate random $r \xleftarrow{R} \mathbb{Z}_n^*$. Then compute $x = f_2(f_1(r)) = g^r \in G_1$ and $y = f_1(r) = r^e \in \mathbb{Z}_n^*$. We see that $(x, y) \in \mathbb{G}$. In this case we call r, the *sampling information* for (x, y).

2. Let $A, B \in \mathbb{G}$. Anyone who has sampled either one of A or B can compute $A \star B$ efficiently. To see this, let $A = (x_a, y_a)$ be sampled using the sampling information $r_a \in \mathbb{Z}_n^*$. That is, $x_a = g^{r_a} \in G_1$ and $y_a = r_a^e \in \mathbb{Z}_n^*$. Also let $B = (x_b, y_b)$. We can compute $C = A \star B$ as follows. Let $C = (x_c, y_c)$. Then $x_c = x_b^{r_a} \in G_1$ and $y_c = y_a y_b \in \mathbb{Z}_n^*$.

3. Similarly, anyone who has sampled $A \in \mathbb{G}$ can also compute $A^{-1} \star B$ for any $B \in \mathbb{G}$ because if $r \in \mathbb{Z}_n^*$ is the sampling information for A then $r^{-1} \in \mathbb{Z}_n^*$ is the sampling information for A^{-1}. Also, for any $i \in \mathbb{N}$, $r^i \in \mathbb{Z}_n^*$ is the sampling information for $A^i \in \mathbb{G}$.

4. Let $A, B \in \mathbb{G}$ be given. For anyone who has *not sampled* at least one of $\{A, B, A^{-1}, B^{-1}\}$ computing $A \star B$ is intractable if breaking RSA is hard.

5. Let $A, B \in \mathbb{G}$ be given. For anyone who has *not sampled* at least one of $\{A, A^{-1}\}$ computing $A^{-1} \star B$ is intractable if breaking RSA is hard.

6. It is infeasible to decide if any given pair $(x, y) \in \mathbb{G}$ unless breaking RSA is easy. However, it appears that even the ability to decide membership of $\mathbb{G}$ does not help in breaking RSA.

7. Let $A, B \in \mathbb{G}$. The oracle $\mathcal{O}$ has the ability to compute $A \star B$ since it knows the secret information d (effectively the factors of n) which can be used to compute f^{-1}. Computing $A \star B$ is then straightforward using equation 1.

8. Additionally, it is possible to use the oracle to compute $A \star B$ for any $A, B \in \mathbb{G}$ such that the oracle does not know either of A or B using the following blinding technique.

    1. The input is $A, B \in \mathbb{G}$ such that none of $\{A, B, A^{-1}, B^{-1}\}$ have been sampled by us.
    2. Sample $A_1, B_1 \xleftarrow{R} \mathbb{G}$. Thus we can compute $A' = A \star A_1$ and $B' = B \star B_1$. [See items 1 and 2 in the above discussion].
    3. Use the oracle to output $C' = \mathcal{O}(A', B') = A' \star B'$. [See item 7 above].
    4. Compute $C = A_1^{-1} \star B_1^{-1} \star C'$ and output $C = A \star B$. [See item 3 above].

9. Due to the properties of bilinear maps, it is possible to verify the outputs of the oracle as follows. Let $A = (x_a, y_a)$ and $B = (x_b, y_b)$ such that $A, B \in \mathbb{G}$. Let $C = (x_c, y_c) \in \mathbb{S} \times \mathbb{Z}_n^*$ be the output of the oracle with input (A, B). Clearly, $C = A \star B$ if and only if $y_c = y_a y_b \in \mathbb{Z}_n^*$ and $\hat{e}(g, x_c) = \hat{e}(x_a, x_b) \in G_2$. We will use this technique to protect against active adversaries.

10. Of the public values in the initial setup, the pair $(h, \beta) \in \mathbb{G}$.

3.5 Oracle Functionality

The oracle $\mathcal{O}$ works as follows.

1. For any inputs A, B, if $(A, B) \notin \mathbb{G}^2$, the oracle sets $C = (g, 1) \in \mathbb{G}$.
2. On the other hand, if $(A, B) \in \mathbb{G}^2$ (i.e. the inputs are valid), the oracle computes $C = A \star B \in \mathbb{G}$ using the method defined above.
3. It replies with $C = \mathcal{O}(A, B)$.

Thus, the oracle can also be used to decide if any given pair $(x, y) \in \mathbb{G}$. Additionally, given any $A \in \mathbb{G}$, we can use the oracle to compute A^i for any $i \in \mathbb{N}$ using the "repeated squaring" method. The replies of the oracle are assumed to be instantaneous.

The main idea behind out schemes is that the oracle allows us to compute $\star$ (i.e. multiply in $\mathbb{G}$) but does not allows us to invert $\star$ (i.e. divide). From this point onwards, we will use the notation $\mathcal{O}(A, B)$ and $A \star B$ interchangeably for any $A, B \in \mathbb{G}$.

We emphasize that throughout this discussion we will assume that calls to the oracle are public and any passive adversary is allowed to observe both the input and output. In case we require to use the oracle to compute $\star$ on secret information, we **must** blind the inputs using the method described in section 3.4, item 8 to keep the values hidden from passive adversaries. Since we can verify the outputs of the oracle (see section 3.4, item 9) we are also protected from active adversaries.

4 Group Key Agreement

We will use the protocol of Rabi and Sherman [5] to compute a shared group key. The operation $\star$ on $\mathbb{G}$ acts as a SAOWF. For any $A, B \in \mathbb{G}$, if neither of A or B have been sampled by the us, we will use the oracle to compute $A \star B$. Without loss of generality, we demonstrate how four users can compute a shared key. The reader may find it useful to refer back to figure 1 at this time.

4.1 Key Agreement Protocol

The four users (1, 2, 3, 4) can use the oracle to compute a secret key as follows:

1. Recall that, from the set of public parameters, $(h, \beta) \in \mathbb{G}$. Denote this value by P. The four users $(1, 2, 3, 4)$ generate private keys $X_1, X_2, X_3, X_4 \overset{R}{\leftarrow} \mathbb{G}$ respectively using the sampling method described above. The sampling information is also kept as part of the private key.
2. Each user i computes the public key $Y_i = X_i \star P$. This computation is possible because each private key X_i has been sampled by user i. Thus, there are four public keys $Y_1, Y_2, Y_3, Y_4 \in \mathbb{G}$. These keys are long term (i.e. any smaller subset of the four users can use the same public keys to compute their shared key).
3. User 1 uses the oracle to compute the partial public key $Y_{234} = Y_2 \star Y_3 \star Y_4 = X_2 \star X_3 \star X_4 \star P^3$ by making 2 calls; i.e. by computing $Y_{234} = \mathcal{O}(\mathcal{O}(Y_2, Y_3), Y_4)$. We note that everyone is allowed to compute this partial public key Y_{234}.
4. The shared key $K_{1234} = X_1 \star Y_{234} = X_1 \star X_2 \star X_3 \star X_4 \star P^3$ can then be directly computed by user 1 without the help of the oracle using the secret sampling information for generating X_1. User 2 computes K_{1234} independently of user 1 as $K_{1234} = X_2 \star Y_{134} = X_2 \star \mathcal{O}(\mathcal{O}(Y_1, Y_3), Y_4)$.
5. Likewise, the remaining two users can compute the same secret key K_{1234} using the oracle and their secret sampling information. No other user except the oracle $\mathcal{O}$ has the ability to compute this key.

4.2 Join and Merge Operations

Clearly, members can join groups arbitrarily and groups can merge arbitrarily. Rather than giving a formal model for this, we demonstrate this by the following examples.

Example 1: *User 5 joins the group of users (1, 2, 3, 4) with group key created above.*

1. User 5 has private key $X_5 \in \mathbb{G}$ and public key $Y_5 = X_5 \star P \in \mathbb{G}$.

2. To join the group, user 5 computes $K_{12345} = X_5 \star \mathcal{O}(\mathcal{O}(\mathcal{O}(Y_1, Y_2), Y_3), Y_4) \in \mathbb{G}$.

3. Users 1 computes $K_{12345} = X_1 \star \mathcal{O}(Y_{234}, Y_5)$. Similarly users (2, 3, 4) compute K_{12345} using their private keys, partial public keys and Y_5 by making one oracle call each.

Example 2: *The group of users (1, 2, 3, 4) merges with the group of users (4, 5, 6).*

4. Denote the set of users $\{1, 2, 3, 4\}$ by s_a and the set of users $\{4, 5, 6\}$ by s_b. The private key of s_a is $K_a = K_{1234}$ and the public key is $Y_a = Y_{1234} = Y_1 \star Y_2 \star Y_3 \star Y_4$. Similarly, the private key of s_b is $K_b = K_{456}$ and the public key is $Y_b = Y_{456} = Y_4 \star Y_5 \star Y_6$.

5. Each member of s_a computes $K_{ab} = K_a \star Y_b$ using the oracle after blinding K_a, while each member of K_b computes $K_{ab} = K_b \star Y_a$ using the oracle after blinding K_b. User 4, who is common to s_a and s_b can choose to compute K_{ab} either way. The group public key corresponding to the group private key K_{ab} is $Y_{ab} = Y_a \star Y_b$.

4.3 Forward Secrecy

Observe that due to the above mentioned merge procedure, the compromise of the group key of a set s_a of users compromises the group key of any other set $s_c \supsetneq s_a$. To overcome this weakness, if the private key of group s_a is compromised, at least one member of s_a must compute a new public-private key pair. We also note that compromise of a group private key **does not** compromise the private keys of any members of that group.

4.4 Overview Of The Above Scheme

The above protocol is a constructive existence proof of a SAOWF because we *construct* the one-way function using an oracle. Additionally, an attacker is allowed to make unlimited calls to the oracle (limited only by time). Thus, we call our primitive an Oracle based-SAOWF (or O-SAOWF). We note the following points about the cryptosystem.

1. *Complexity*: For a group of m users, a total of $m - 2$ oracle calls are required for each user to compute the shared key. Thus a total of $m(m - 2)$ calls are required for all the m users. However, no specific ordering is required between the users. A user i may choose to compute the shared key *after* a ciphertext is received. Additionally, oracle calls can be sent in a batch.

2. *Universal Key Escrow*: The oracle has universal escrow capability. Given a public key $Y_i = X_i \star P$ for some private key $X_i \in \mathbb{G}$, the oracle can invert $\star$ and compute X_i.

3. *Non-interactivity*: Assuming that all the public keys Y_i are known in advance, any user can compute the shared key without interacting with the other users.

4. *Verifiability of the Oracle*: If verifiability of the oracle is not required (i.e. we need protection only from passive adversaries) then instead of the bilinear groups G_1, G_2, we can use a finite field having a multiplicative subgroup of order n. The set $\mathbb{S}$ is then the $\phi(n)$ elements of this field of order n.

5. *Decentralizing the Oracle*: We observe that an arbitrary number of "copies" of the oracle can be run without any compromise in security. The ability to do more computations of $\star$ does not give any additional advantage.

5 Other Applications

The above discussion makes it evident that we do not provide a key agreement protocol. Rather, we provide a practical implementation of a new cryptographic primitive called Strong Associative One-Way Function (SAOWF). As shown in the literature, SAOWFs have many applications [8, 5]. To demonstrate this, we present two more applications; (a) signatures and (b) multi-user (ring) signatures. Throughout this discussion, we will assume that the oracle is trusted, the public parameters of the oracle are used in constructing the function $\star$ and that $\alpha \in \mathbb{Z}_n^*$, the sampling information of $P = (h, \beta)$ is kept secret by the oracle.

5.1 Single-User Signatures

Let $m \in \mathbb{N}$ be some message. We compute a signature for user 1 with private key (X_1, r_1) (where $X_1 \in \mathbb{G}$ and $r_1 \in \mathbb{Z}_n^*$ is the sampling information for X_1) and public key $Y_1 = X_1 \star P \in \mathbb{G}$.

1. *Sign*: To sign message m, compute $S = X_1^m \star P$. The signature of user 1 on m is S.

2. *Verify*: To verify a message-signature pair (m, S), we check if the equality $Y_1^m \stackrel{?}{=} S \star P^{m-1}$ holds.

We note that S can be directly computed by user 1 without help of the oracle because the sampling information for X_1^m is $r_1^m \in \mathbb{Z}_n^*$. On the other hand, we need the oracle's help to compute Y_1^m, P^{m-1} and $S \star P^{m-1}$ using the repeated squaring method, which will amount to $< k \cdot log_2(m)$ oracle-calls for a verifier (for some small constant k). The above protocol demonstrates the connection between the Strong Diffle-Hellman assumption and the Strong RSA assumption.

5.2 Multi-User And Ring Signatures

Our construction of group signatures is not that efficient, in that the signer is also required to use the oracle. To sign messages, members of a group must share a secret group key which is computed using the method given in section 4. For simplicity, we will only demonstrate (without loss of generality) how three users (1, 2, 3) can sign messages once they have agreed on the group private key $K_{123} = X_1 \star X_2 \star X_3 \star P^2$. This group also has the group public key $Y_{123} = \mathcal{O}(\mathcal{O}(Y_1, Y_2), Y_3) = X_1 \star X_2 \star X_3 \star P^3$ which can be computed using the help of the oracle.

1. *Sign*: As before, let $m \in \mathbb{N}$ be the message to be signed. User 1 computes $S = K_{123}^m \star P$ using the oracle. Since calls to the oracle are public, user 1 must use the the blinding technique mentioned in section 3.4, item 8 to keep the value of K_{123} (and of the powers of K_{123}) hidden from a passive adversary. The signature on m is S.

2. *Verify*: To verify a message-signature pair (m, S), we check that $Y_{123}^m = S \star P^{m-1}$.

In the above construction, group signature are "closed". In other words, it is not possible for any group controller to revoke the anonymity of the signer (since there is no group controller). Thus, the above scheme also demonstrates an example of ring signatures [12].

6 Security

The oracle is primarily used as a "computing device" in our proofs. We assume that the oracle always functions correctly and keeps the trapdoor information d secret. Our initial task is to somehow "extract" d from the oracle. However, this is equivalent to factoring n so we look at the next task; being able to "divide" by $P \in \mathbb{G}$ (or in other words, invert $\star$ with respect to P). We start by proving the following theorem that shows that being able to "divide" by P allows us to compute P^{-1} directly.

Theorem 1. *Let $X, P \stackrel{R}{\leftarrow} \mathbb{G}$ where $X \neq (g, 1)$ such that the sampling information of P is unknown. An algorithm $\mathcal{A}_1$ that computes $X \star P^{-1} \in \mathbb{G}$ can be converted into an algorithm $\mathcal{A}_2$ that computes $P^{-1} \in \mathbb{G}$.*

Proof. $\mathcal{A}_2$ runs $\mathcal{A}_1$ as follows. The input to $\mathcal{A}_2$ is $P \in \mathbb{G}$. Algorithm $\mathcal{A}_2$ samples $X \leftarrow \mathbb{G}$ such that $X \neq (g, 1)$. It gives the pair (X, P) as input to $\mathcal{A}_1$ which outputs $B = X \star P^{-1} \in \mathbb{G}$. Algorithm $\mathcal{A}_2$ then outputs $P^{-1} = B \star X^{-1} \in \mathbb{G}$.

6.1 The Group Inversion Problem

Due to theorem 1, the security of all the above schemes reduces to the following problem.

Group Inversion Problem. *($GIP_{\mathbb{G}}$). Let $P = (h, \beta) \xleftarrow{R} \mathbb{G}$ be uniformly sampled. (using secret $\alpha \xleftarrow{R} \mathbb{Z}_n^*$ such that $h = g^\alpha \in G_1$ and $\beta = \alpha^e \in \mathbb{Z}_n^*$). Given P, compute $P^{-1} = (h', \beta') \in \mathbb{G}$ where $h' = g^{1/\alpha} \in G_1$ and $\beta' = (1/\alpha)^e \in \mathbb{Z}_n^*$, possibly by using the oracle $\mathcal{O}$.*

Clearly $\beta' = 1/\beta \in \mathbb{Z}_n^*$ can be efficiently computed. However, to compute h' we need to know the factorization of n in addition to having access to the oracle. Note that computing $1/\alpha$ is out of question since this will be equivalent to both breaking RSA **and** computing discrete logarithms in G_1 to base g. So we should look at indirect methods of computing $g^{1/\alpha} \in G_1$ without having the ability to compute $1/\alpha \in \mathbb{Z}_n^*$. Formally, we define the advantage of an algorithm solving this as follows.

Definition 1. *For any algorithm $\mathcal{A}$, the advantage of $\mathcal{A}$ in solving the group inversion problem GIP-Adv$_{\mathcal{A}}(\tau)$ for some security parameter τ is defined as:*

$$\textit{GIP-Adv}_{\mathcal{A}}(\tau) = \textbf{\textit{Pr}}\left[\ \mathcal{A}(\hat{e}, n, G_1, G_2, g, e, h, \beta, \mathcal{O}) = g^{1/\alpha} :\ \begin{array}{c} (\hat{e}, G_1, G_2, p, q) \leftarrow \mathcal{O}(\tau), \\ n = pq, e \leftarrow \mathbb{Z}_{\phi(n)}^*, g \leftarrow G_1, \\ \alpha \leftarrow \mathbb{Z}_n^*, h = g^\alpha, \beta = \alpha^e \end{array}\right] \tag{2}$$

where $\mathcal{O}$ is an oracle with the functionality defined in section 3.5.

Our security is based on the following conjuncture.

Conjuncure 1. *For any random $P \in \mathbb{G}$ such that the sampling information for P is unknown, computing P^{-1} (with respect to $\star$) is infeasible unless the factors of n are known. This assumption holds for any PPT adversary having access to oracle $\mathcal{O}$. In other words, for any algorithm $\mathcal{A}$, we have that GIP-Adv$_{\mathcal{A}}(\tau)$ is a negligible function in τ.*

6.2 Relationship With Other Problems

To get more confidence in the above problem, we show its relation to the RSA problem and the Inverse Diffie-Hellman Problem. The reader is referred back to section 3.1 for the notation used here. First we define the following problems.

1. *Discrete Log Problem ($DLG_{(g,G_1)}$)*: Let g be a fixed generator of G_1. For any input $g^x \in G_1$, output $x \in \mathbb{Z}_n$.
2. *Extended RSA Problem ($ERSA_{(g,e,n,G_1)}$)*: Let g be a fixed generator of G_1 and let e be a fixed RSA public exponent with the modulus n. For any input $x^e \in \mathbb{Z}_n^*$, output $g^x \in G_1$.

6.2.1 The Extended RSA Problem

The Group Inversion Problem reduces to the Extended RSA problem (theorem 4 below). We give a sufficient condition for the Extended RSA problem to be hard in theorem 5.

Theorem 2. *$GIP_{\mathbb{G}} \Rightarrow ERSA_{(g,e,n,G_1)}$.*

Proof. Given a GIP$_{\mathbb{G}}$ instance $(g^x, x^e) \in \mathbb{G}$, we compute $x' = 1/x^e \in \mathbb{Z}_n^*$. Then x' forms an instance of the ERSA$_{(g,e,n,G_1)}$ problem, the solution of which will be a solution to the GIP$_{\mathbb{G}}$ instance.

Theorem 3. *A sufficient (but not necessary) condition that $ERSA_{(g,e,n,G_1)}$ problem is hard is that $RSA_{(e,n)} \not\Rightarrow DLG_{(g,G_1)}$.*

Proof. Let the $\mathrm{ERSA}_{(g,e,n,G_1)}$ problem be easy. Then given $x^e \in \mathbb{Z}_n^*$, we can compute $g^x \in G_1$. Thus, $\mathrm{RSA}_{(e,n)} \Rightarrow \mathrm{DLG}_{(g,G_1)}$. Therefore, if $\mathrm{RSA}_{(e,n)} \not\Rightarrow \mathrm{DLG}_{(g,G_1)}$ then $\mathrm{ERSA}_{(g,e,n,G_1)}$ must be hard. The converse may not be true; even if $\mathrm{RSA}_{(e,n)} \Rightarrow \mathrm{DLG}_{(g,G_1)}$ it may be that $\mathrm{ERSA}_{(g,e,n,G_1)}$ is still hard.

6.2.2 The Inverse Diffie-Hellman Problem

Clearly, the group inversion problem reduces to the inverse Diffie Hellman problem, $\mathrm{IDHP}_{(g,G_1)}$ (see section 3.1). Although, it is not known if $\mathrm{DHP}_{(g,G_1)}$ reduces to the oracle $\mathcal{O}$, we conjure that any method of reducing $\mathrm{IDHP}_{(g,G_1)}$ to $\mathrm{DHP}_{(g,G_1)}$ will yield a method of reducing $\mathrm{GIP}_{\mathbb{G}}$ to the oracle $\mathcal{O}$. We give a series of conjunctures on the relation between the inverse Diffie-Hellman problem and the group inversion problem.

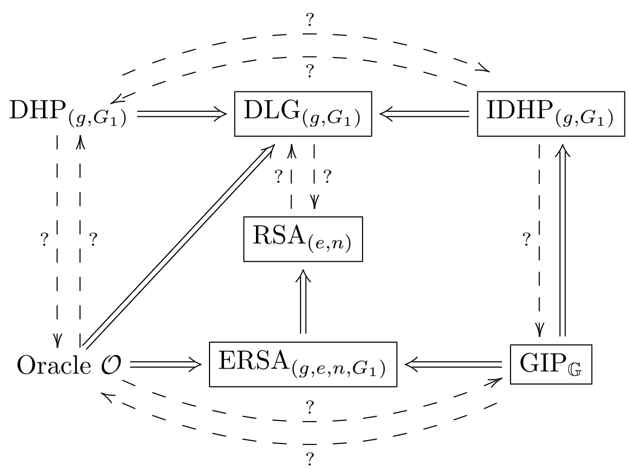
Conjuncure 2a. *Any method of reducing* $\mathit{IDHP}_{(g,G_1)} \Rightarrow \mathit{DHP}_{(g,G_1)}$ *also provides a method of reducing* $\mathit{GIP}_{\mathbb{G}} \Rightarrow$ *oracle* $\mathcal{O}$*.*

Conjuncure 2b. *Any method of reducing* $\mathit{DHP}_{(g,G_1)} \Rightarrow$ *oracle* $\mathcal{O}$ *also provides a method of reducing* $\mathit{IDHP}_{(g,G_1)} \Rightarrow \mathit{GIP}_{\mathbb{G}}$*.*

We also give a "stronger" version of the above conjuncture which states that it is highly unlikely that the Diffie-Hellman Problem can be reduced to the oracle $\mathcal{O}$.

Conjuncure 2c. *Any method of reducing* $\mathit{DHP}_{(g,G_1)} \Rightarrow$ *oracle* $\mathcal{O}$ *also provides a method of reducing* $\mathit{DLG}_{(g,G_1)} \Rightarrow \mathit{RSA}_{(e,n)}$

We can summarize our results using the following diagram. The boxes indicate problems relevant to us. The solid double arrows indicate known reductions while the single dashed arrows indicate that a reduction is not known. There is no reduction from the oracle $\mathcal{O}$ to $\mathrm{DHP}_{(g,G_1)}$ because the oracle $\mathcal{O}$ also needs to decide if the inputs are elements of $\mathbb{G}$ or not, which a $\mathrm{DHP}_{(g,G_1)}$ oracle cannot do.



7 Implementation And Efficiency

In this section, we will briefly touch upon issues relating to implementation and efficiency of our primitive. The security of the above protocols is based on the intractability of factoring n. Based on the current state of the art factoring algorithms, we suggest using an RSA modulus of about 616 decimal digits ($\approx$ 2048 bits) for high security applications.[3] This also makes computing discrete logarithms in G_1 to base g intractable using Pollard's lambda method [13, p.128].

[3] See for example, the RSA factoring challenge (http://www.rsasecurity.com/rsalabs/node.asp?id=2092) and the article "TWIRL and RSA key size" (http://www.rsasecurity.com/rsalabs/node.asp?id=2004). It is thought that 2048 bit keys will be secure till the year 2030.

Although, our construction of O-SAOWF has other applications as demonstrated, we feel that the primary use of our scheme will be for highly dynamic group key agreement in applications like "secure chat". Our system offers the advantage that the group key need not be precomputed for communication between group members. Thus, there is no specific ordering (unlike the Group Diffie-Hellman (GDH) protocol [4]) between the users. Additionally, since there is no need for a secure channel between any of the participants, all messages may be directly broadcast. The blinding method described above allows us to mask secret keys when using the oracle to compute with them.

For increased efficiency in partial public key computation, we will assume that calls to the oracle can be batched as follows, for any i inputs $A_1, A_2, \ldots A_i \in \mathbb{G}$, the oracle outputs $A_1 \star A_2 \star \ldots A_i$. In this case, for key computation in a group of m users each user must make a batch call requiring a message of size $2log_2(n)(m-1)$ bits to be sent to the oracle. The reply of the oracle constitutes just one element of size $2log_2(n)$. However, we lose the ability to verify the output of the oracle in a "batch query".

To increase efficiency in applications where "exponentiation" in $\mathbb{G}$ is required using the oracle via the repeated squaring method (i.e., we need to compute A^i for some $A \in \mathbb{G}$ and $i \in \mathbb{N}$ such that A has not been sampled by us), the oracle can also accept "exponentiation" queries.

Finally, we note that it is possible to share the RSA decryption key (known only to the oracle) between different trusted authorities with the weakness that compromise of even one would compromise the entire system. We close this section with a comparison of our scheme with previously proposed group key agreement methods in table 1.

Membership size is m	**O-SAOWF**	**GDH** basic [4]	**AGKE** [14]	**GKE**[15]
Number of rounds	1	$m-1$ sequential	2 sequential	2 sequential
Synchronization/ ordering needed ?	No	Yes	Yes	Yes
Controller Required ?	No	No	Yes (for initial key distribution)	Yes (for group key distribution)
Interaction Required ?	No	Yes	Yes	Yes (for synchroniza- -tion) otherwise no
Key Agreement Method	Oracle	Self (interactive)	Self (broadcast)	Controller
Message size per user (sent)*	$(m-1)k_1$	$(m-1)k_2$	k_3(broadcast only) otherwise mk_3	$2k_4$ (to controller)
Message size per user (rcvd)	k_1 (no verification) $(m-2)k_1$ (verification)	$(m-1)k_2$	mk_3	k_4
Merge/Part with m_1 users	1 round (total $2(m+m_1)$ messages)	$O(m \pm m_1)$ rounds (fresh key)	2 rounds (total $m+m_1$ broadcasts)	2 rounds (total $2m_1+m$ messages)
Partial Public keys reusable ?	Yes	No	No	No
Optimization Possible?	Yes**	Not likely	Not likely	Not likely
Protection under active attack	Yes# Verifiable Oracle	Susceptible to man in the middle attack	Authentication after 2nd round	Insecure under an active attack [16]
Protection under passive attack	Group Inversion Problem	Diffie-Hellman Problem	Diffie-Hellman Problem	Diffie-Hellman Problem

* We assume that $k_1, k_2 \ldots k_n$ are constants.

** Assuming that intermediate controllers are used and partial public keys are cached.

\# Assuming that public keys are known in advance, the verifiability of the oracle ensures *implicit group key authentication* and an active attack cannot be mounted on the key agreement protocol. On the other hand, other protocols require the intermediate messages to be explicitly authenticated.

Table 1: Comparison of our group key agreement scheme

8 Conclusion

In this paper, we presented a practical implementation of a new cryptographic primitive known as an Oracle-based Strong Associative One-Way Function (O-SAOWF) using the idea of hidden order groups. As some practical applications of this primitive, we presented a one-round key agreement scheme for dynamic ad-hoc groups based on a combination of the $n(n-1)$-round Group Diffie-Hellman (GDH) [4] and the 1 round protocol using Strong Associative One-Way Functions (SAOWFs) due to Rabi and Sherman [5]. Our scheme can be extended to group signatures as demonstrated in section 5. In reality, our scheme also demonstrates a "pay-per-use" cryptographic primitive using the oracle. The advantage of our scheme in comparison with other centralized schemes is that the central controller does not maintain any state information of the groups it is managing. It just acts as a "computing device" for users registered with it and the only information needed for this computation is d, the private key corresponding to the RSA public key (e, n). We envisage several interesting applications of our primitive in the near future.

As we demonstrate, the ability to "multiply" using the oracle does not give us the ability to "divide" in $\mathbb{G}$ because its order is unknown. This ensures that an "Euclidean"-like Algorithm does not work here. The curious property of the SAOWF demonstrated in this paper is that it is *weakly invertible*. In other words, given $A \in \mathbb{G}$, it is possible to compute two pairs $(B, B') \in \mathbb{G}^2$ such that $A = B \star B'$ even without using the oracle.[4] It may be that all average-case SAOWFs exhibit this property. We conclude this paper with three open questions (in addition to the conjunctures mentioned earlier).

1. Can we use the oracle $\mathcal{O}$ to factor n?
2. Is it possible to reduce $\text{IDHP}_{(g,G_1)}$ to $\text{DHP}_{(g,G_1)}$ without knowing the factorization of n?
3. Construct a group of hidden order where "multiplication" can be done without using an oracle.

Acknowledgment

We would like to thank Ronald Rivest, Virendra Sule and Chunbo Ma for useful feedback and for pointing out some errors and missing citations in the original draft.

References

[1] Sandro Rafaeli and David Hutchison. A survey of key management for secure group communication. *ACM Comput. Surv.*, 35(3):309–329, 2003.

[2] Xukai Zou, Byrav Ramamurthy, and Spyros S. Magliveras. *Secure Group Communications Over Data Networks.* Springer, New York, NY, USA, 2005.

[3] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[4] Michael Steiner, Gene Tsudik, and Michael Waidner. CLIQUES: A new approach to group key agreement. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98)*, pages 380–387, Amsterdam, 1998. IEEE Computer Society Press.

[5] Muhammad Rabi and Alan T. Sherman. An observation on associative one-way functions in complexity theory. *Inf. Process. Lett.*, 64(5):239–244, 1997.

[6] Lane A. Hemaspaandra, Kari Pasanen, and Jörg Rothe. If p $\neq$ np then some strongly noninvertible functions are invertible. In *FCT '01: Proceedings of the 13th International Symposium on Fundamentals of Computation Theory*, pages 162–171. Springer-Verlag, 2001.

[4] To see this, sample $B \leftarrow \mathbb{G}$. Then $B' = B^{-1} \star A$.

[7] Lane A. Hemaspaandra, Jörg Rothe, and Amitabh Saxena. Enforcing and defying associativity, commutativity, totality, and strong noninvertibility for one-way functions in complexity theory. In *ICTCS*, 2005.

[8] Susan Hohenberger. The cryptographic impact of groups with infeasible inversion. Master's thesis.

[9] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.

[10] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[11] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

[12] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. *Lecture Notes in Computer Science*, 2248:552–??, 2001.

[13] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[14] Hyun-Jeong Kim, Su-Mi Lee, and Dong Hoon Lee. Constant-round authenticated group key exchange for dynamic groups. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2004.

[15] E. Bresson, O. Chevassut, A. Essiari, and D. Pointcheval. Mutual authentication and group key agreement for low-power mobile devices, 2003.

[16] Seungjoo Kim Junghyun Nam and Dongho Won. Attacks on bresson-chevassut-essiari-pointcheval's group key agreement scheme for low-power mobile devices. Cryptology ePrint Archive, Report 2004/251, 2004.