A New Cryptosystem Based On Hidden Order Groups

Amitabh Saxena and Ben Soh
Email: {asaxena, ben}@cs.latrobe.edu.au
Department of Computer Science and Computer Engineering
La Trobe University
VIC, Australia 3086

March 5, 2007

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 (see Theorem 3.2). On the other hand, it is not known if such a reduction exists when $\phi(n)$ is unknown (see Conjuncture 3.1). We exploit this "gap" to construct a cryptosystem based on hidden order groups and present a practical implementation of a novel cryptographic primitive called an *Oracle Strong Associative One-Way Function* (O-SAOWF). O-SAOWFs have applications in multiparty protocols. We demonstrate this by presenting a key agreement protocol for dynamic ad-hoc groups.

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 envisage an ad-hoc group as a broadcast group where members do not have one-to-one channels; rather they share the communication medium such that everyone within range is able to receive any broadcast message. An efficient group key agreement protocol in this scenario should satisfy the property that the shared group key is computable without interaction with the other members. Such protocols are often called *one-round* key agreement protocols where the only round consists of the initial key distribution phase. Two notable examples of one-round key agreement protocols are the classic two-party Diffie-Hellman key exchange [1] and the Joux tripartite key exchange using bilinear maps [2]. However, till date constructing a generalized one-round n-party key agreement protocol has remained a challenging and open problem.

1.1 Our Contribution

In this paper, we present the first practical example of a one-round key agreement protocol for arbitrary size groups. Although our construction enables the group key to be computed non-interactively, it comes with a caveat; a trusted third party is required to do most of the computation.

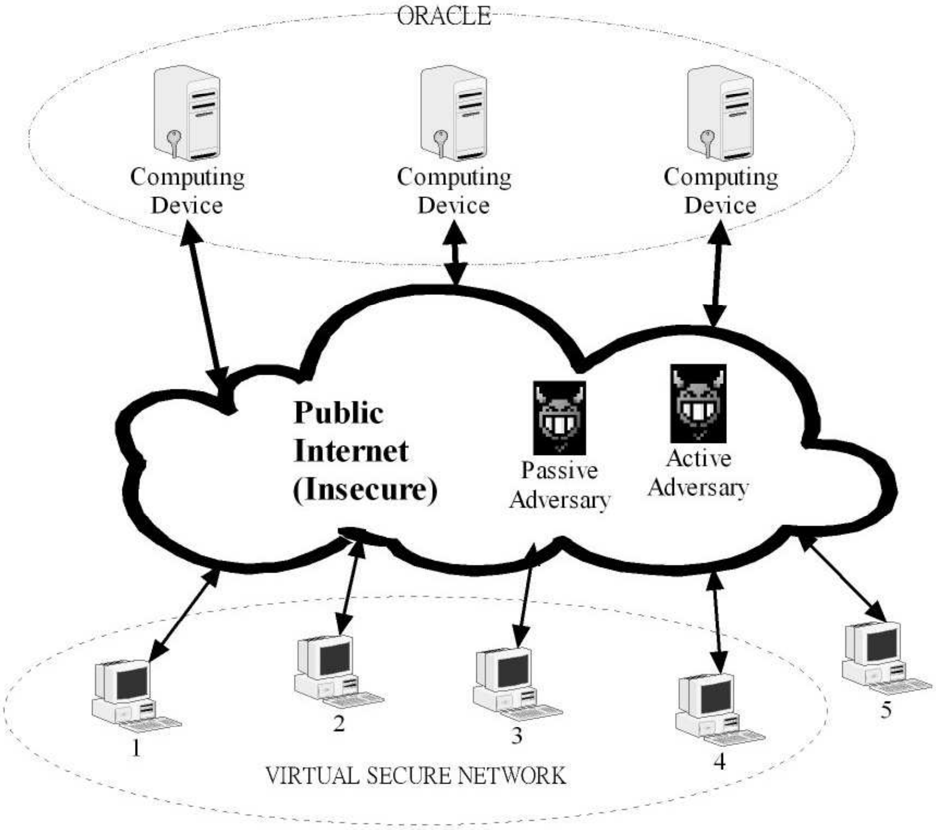
We refer the reader to [3, 4] for a survey of key agreement protocols for ad-hoc groups. In the literature, most group key agreement protocols are categorized as (a) Centralized, (b) Distributed and (c) Fully Contributory. Our proposed method is fully contributory, yet it uses a central authority. We elaborate on this below.

1.2 The Main Idea

The original two-party Diffie-Hellman key exchange [1] can be extended to fully contributory multi-party key exchange as demonstrated in [5] using the Group Diffie-Hellman (GDH) protocol. However, all 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 authority in computing the shared group key. However, the central authority 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 model for secure group communication (see 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 key independently of the other users such that no (active or passive) adversary has the ability to compute this key. Essentially the oracle is used as a "verifiable computing device" and the adversary as the communication medium.

Figure 1: Secure group communication in our model.

Our basic idea arises due to the paper of Rabi and Sherman [6], where they described a cryptographic primitive called a *Strong Associative One-Way Function (SAOWF)*, and discussed as an application a one-round key agreement protocol in ad-hoc groups. In related work, Boneh and Silverberg also proposed a one-round key agreement protocol for ad-hoc groups based on a similar primitive called a *multilinear map* [7]. However, as of now no practical construction of either primitive is known. In this paper we extend the work of Rabi and Sherman and give a practical construction of a SAOWF under a restricted model of computation, namely *black-box computation*. Our construction also gives an example of a *Trapdoor Group with Infeasible Inversion* (TGII) discussed in [8].

The rest of the paper is organized as follows. In section 2 we give some background and notation. We define SAOWFs in section 2.1 and extend this definition to include black-box computation in section 2.4. Our construction is presented in section 4 and some applications are given in section 5. Finally, we discuss implementation issues in section 6.

2 Preliminaries

Around 1984, Rivest and Sherman suggested the idea of one-round key agreement in ad-hoc groups using a class of cryptographic primitives that they called *Associative One-Way Functions (AOWFs)* [9, 10]. Later in 1993, Rabi and Sherman suggested the use of AOWFs in digital signatures [11]. In subsequent work, Rabi and Sherman [6] gave an existence proof of complexity theoretic AOWFs under the $P \neq NP$ hypothesis. Other authors studied complexity theoretic AOWFs with respect to different properties such as low ambiguity, strong invertibility, totality and commutativity [12, 13, 14]. Finally, in [15], Hemaspaandra, Rothe and Saxena gave a complete characterization of complexity theoretic AOWFs.

In all the above works, however, the AOWFs considered are *complexity theoretic*, that is, they exhibit useful characteristics only in the *worst case* and not in the *average case*. Such constructions do not have much practical significance in the context of cryptography. In this work we focus on *cryptographic* AOWFs - that exhibit useful characteristics even in the average case. Additionally, we study only a small family of AOWFs, namely those that are commutative, total and strongly non-invertible. We call this the class of *Strong Associative One-Way Functions (SAOWFs)*.

2.1 Strong Associative One-Way Functions

Let $(\mathbb{G}, \star)$ be a finite abelian group. The mapping

$$\begin{array}{rcl} f : \mathbb{G} \times \mathbb{G} & \mapsto & \mathbb{G} \\ (A, B) & \mapsto & A \star B \end{array}$$

has the following four properties (we use the notation $f(A, B)$ and $A \star B$ interchangeably):

P1. **Associativity**: $f(f(A, B), C) = f(A, f(B, C)) \ \forall A, B, C \in \mathbb{G}$.

P2. **Commutativity**: $f(A, B) = f(B, A) \ \forall A, B \in \mathbb{G}$.

P3. **Identity**: There exists a unique element $I \in \mathbb{G}$ such that $f(A, I) = A \ \forall A \in \mathbb{G}$. We say I is the identity element. Denote by $\mathbb{G}^*$ the set $\mathbb{G} \backslash \{I\}$.

P4. **Inverses**: For each $A \in \mathbb{G}^*$, there exists a unique $B \in \mathbb{G}^*$ such that $f(A, B) = I$. We say B is the inverse of A and denote it by A^{-1}.

We now additionally want to enforce the following three properties on $(\mathbb{G}, \star)$:

P5. **Samplability**: Elements of $\mathbb{G}$ must be efficiently samplable.

P6. **Computability**: For all $A, B \in \mathbb{G}$, $f(A, B)$ must be efficiently computable.

P7. **Strong Non-Invertibility**: Let $A, B \overset{R}{\leftarrow} \mathbb{G}^*$ and $C \leftarrow f(A, B) \in \mathbb{G}$. Given A, C, computing $B = f(C, A^{-1})$ must be infeasible in the average case.

Definition 2.1. *We say that f is a* Strong Associative One-Way Function *(SAOWF) if properties P1-P7 are satisfied.*[1]

Remark 2.2. A SAOWF as defined above is analogous to a Group with Infeasible Inversion (GII) defined in [8].

Although SAOWFs have many applications as demonstrated in [6, 8, 16], exhibiting a practical construction of a SAOWF is still an open problem. We make a positive progress in this direction by presenting a practical black-box construction of a SAOWF.

We note that it is possible to construct a SAOWF f under the $P \neq NP$ hypothesis if we replace "average case" by "worst case" in the statement of property P7 [14, 15]. However, for applications significant to cryptography we require property P7 to be defined in the average case.

For completeness, we also define weak non-invertibility as follows.

P8. **Weak Non-Invertibility**: Let $C \stackrel{R}{\leftarrow} \mathbb{G}^*$. Given C, computing any pair $(A, B) \in \mathbb{G}^{*2}$ such that $C = f(A, B)$ must be infeasible in the average case.

Definition 2.3. *We say that f is a* Weak Associative One-Way Function *(WAOWF) if properties P1-P6 and P8 are satisfied.*

The strong non-invertibility condition (P7) 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 (P8), which requires that computing **any** preimage of C must be infeasible. In fact, the results of [17] prove that there exists an associative one-way function that is strongly non-invertible but not weakly non-invertible.[2]

Thus, a WAOWF may not be a SAOWF and vice-versa. In this work, 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 a SAOWF is strongly non-invertible, yet it is weakly invertible.

Clearly, property P7 implies that computing inverses in $\mathbb{G}$ must be infeasible. Since the group $(\mathbb{G}, \star)$ is of finite order, the only way to achieve this is to keep the order of this group hidden. This is the main idea behind our construction.

2.2 Black-Box Constructions

Although the original objective of our research was to exhibit a practical construction of a SAOWF, in this work, we focus on a slightly different but related problem: exhibiting a practical *black-box* construction of a SAOWF by extending the definition of "computation" in property P6 to include *oracle computation.*

In our black-box model although the group $(\mathbb{G}, \star)$ is easily samplable, we we do not have access to the algorithm for computing f. Instead, access to the computing algorithm is only provided via a "black-box" with public access. This is illustrated in figure 2.

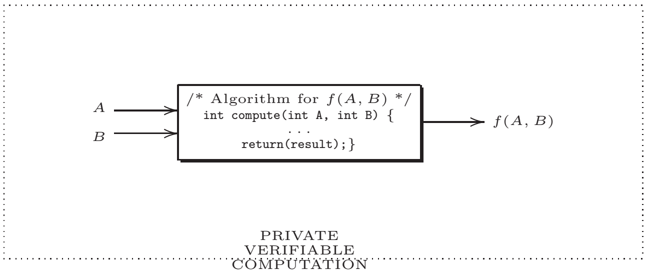
However, for a black-box construction to have any practical significance it must support (a) verifiable and (b) private computation as elaborated next.

2.3 PV-Oracles

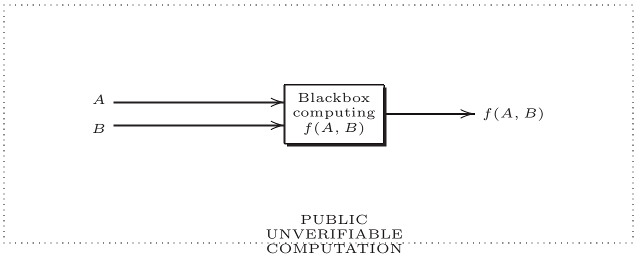
In complexity theory, a black-box with public access is referred to as an *oracle.* In this work, we restrict ourselves to *constructible* oracles (i.e. oracles that can be constructed using some trapdoor), since we want our system to be practical. Additionally, to justify the use of a (constructible) oracle as one-way function in a cryptographic protocol, we must provide the same guarantees that a real function provides.

[1] Most researchers differentiate between commutative and non-commutative SAOWFs [15]. For simplicity, we will enforce the commutativity property (P2) in our definition.

[2] We note that the terminology used in this paper is slightly non-standard (but more intuitive). For instance, "weak non-invertibility" as defined here is simply referred to as "non-invertibility" in the literature [17]. Additionally, "weak" in the literature is used to refer to non-total functions [14]. However, since we are working in finite abelian groups, we can dispense off with definitions such as *honesty*, *non-commutativity* and *totality* used in [14, 15] for describing SAOWFs.



(a) A real computable function



(b) A black-box with public access

Figure 2: Comparing a real and black-box computation.

Specifically, a real function is private and verifiable. We define similar properties for oracles. We will restrict ourselves to an oracle that computes a binary commutative function using two inputs.

Verifiable Oracles. Let f be the binary commutative function computed by an oracle. We say that the oracle supports verifiable computation if for all $A, B \in domain(f)$ and all $C \in image(f)$, there exists a PPT verification algorithm Verify that outputs 1 if $C = f(A, B)$ and 0 otherwise. An oracle supporting verifiable computation is called a *Verifiable Oracle (V-Oracle).* A V-Oracle is illustrated in figure 3.[3]

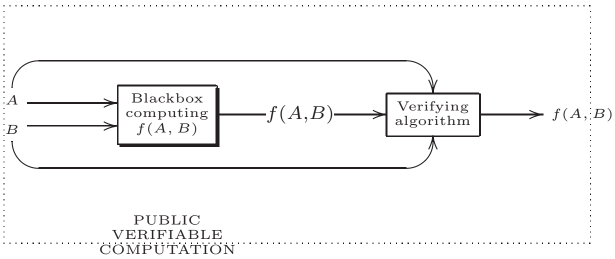


Figure 3: Public, Verifiable Black-box computation (V-Oracle).

Private And Verifiable Oracles. Let f be the binary commutative function computed by a V-Oracle. We say that the V-Oracle supports private computation if the inputs and outputs of the compu-

[3]As an example of a V-Oracle with one input, consider an existentially unforgeable signature scheme. The signing oracle is a V-Oracle since the signature can obviously be verified.

tation can be *blinded* from the V-Oracle such that the blinding algorithm provides information theoretic [18] secrecy. Formally, there must exist two PPT algorithms Blind and $\mathsf{Unblind}$ as follows.

Blind is a randomized algorithm and takes as input an element $A \in domain(f)$ and outputs a tuple (A', σ), where $A' \in domain(f)$, σ is some auxiliary information, and the distributions $\{A\}$ and $\{A'\}$ are independent and identical. We say that Blind is the *Blinding Algorithm* and σ is the *Unblinding Value*.

$\mathsf{Unblind}$ takes as input a tuple (C', σ), where $C' \in image(f)$ and σ is some auxiliary information. It outputs a value $C \in image(f)$ such that the following homomorphic property holds (We call $\mathsf{Unblind}$ the *Unblinding Algorithm*).

$$\Pr\left[\begin{array}{rcl} A, B & \stackrel{R}{\leftarrow} & domain(f); \\ (A', \sigma) & \stackrel{R}{\leftarrow} & \mathsf{Blind}(A); \\ C & \leftarrow & \mathsf{Unblind}(f(A', B), \sigma) : \\ C & = & f(A, B) \end{array}\right] = 1 \tag{1}$$

We call a V-Oracle supporting a private computation a *Private V-Oracle (PV-Oracle)*. See figure 4 for an illustration of a PV-Oracle.[4]

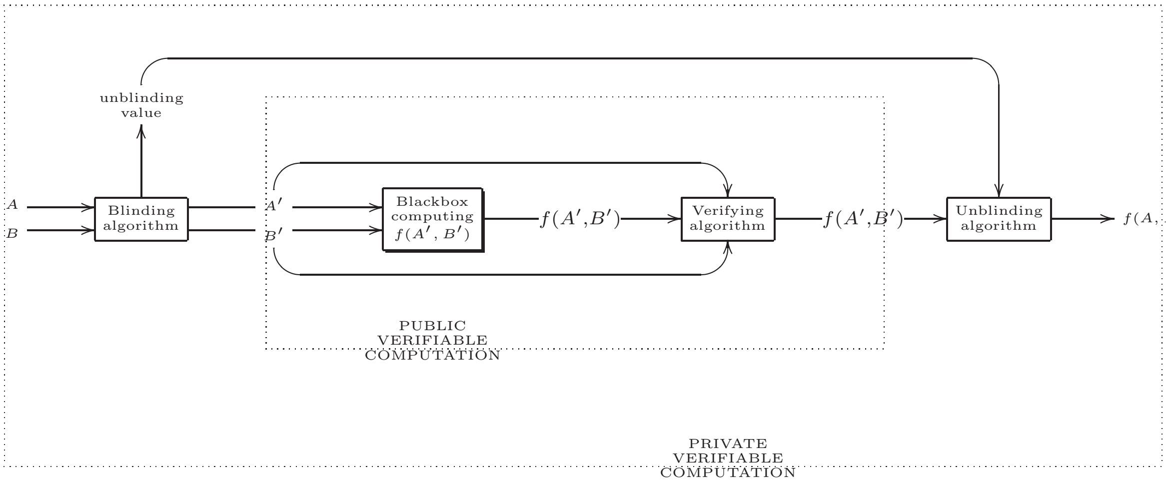


Figure 4: Private and Verifiable Black-box computation (PV-Oracle).

2.4 Oracle SAOWFs (O-SAOWFs)

We now extend the definition of computation in property P6 of section 2.1 to include computation by PV-Oracles. We call such a construction an *Oracle-SAOWF (O-SAOWF)* and formally define it below.

Definition 2.4. *A black-box construction of a SAOWF implemented using a PV-Oracle is called an Oracle-SAOWF (O-SAOWF). An O-SAOWF construction has four PPT "algorithms" as described below (we use quotes here because one of the algorithms* $\mathsf{PV\text{-}Compute}$ *is not a real algorithm in the usual sense; it involves a call to a PV-Oracle).*

Setup This is a randomized algorithm and takes in as input a security parameter τ. It outputs the system parameters params for the group $(\mathbb{G}, \star)$ and a master key $\mathsf{master\text{-}key}$.

[4]As an example of a PV-Oracle with one input, consider a RSA decryption oracle w.r.t. a given RSA public key. Information theoretic privacy for inputs to the decryption oracle can be achieved using Chaum's blinding technique [19].

Sample This is a randomized algorithm and takes in the parameter params. It outputs a uniformly selected element $A \xleftarrow{R} \mathbb{G}$ along with some auxiliary information σ_A, which we will call the *sampling information* in our construction.

Compute This algorithm takes as input the parameter params, the master key $\mathsf{master\text{-}key}$ and two values A, B. If $(A, B) \notin \mathbb{G}^2$, it sets $C \leftarrow I$ (recall that I is the identity element). On the other hand, if $(A, B) \in \mathbb{G}^2$ it computes $C \leftarrow f(A, B) = A \star B$. The output is $C \in \mathbb{G}$. We assume that $\mathsf{master\text{-}key}$ acts like a trapdoor that enables computation of f.

Define a PV-Oracle $\mathcal{O}$ having access to $\mathsf{master\text{-}key}$ and implementing the compute algorithm. We assume that $\mathsf{master\text{-}key}$ is not known to anybody else. The fourth algorithm involves a call to this oracle.

PV-Compute This algorithm takes as input the parameter params and two elements $A, B \in \mathbb{G}$. It uses the Verify, Blind and $\mathsf{Unblind}$ algorithms defined in section 2.3 as sub-routines to compute $C \leftarrow f(A, B) = A \star B$ privately and verifiably by querying the PV-Oracle $\mathcal{O}$ that implements the $\mathsf{Compute}$ procedure. It outputs $C \in \mathbb{G}$.

2.4.1 Security Of O-SAOWFs

Assume that the $\mathsf{PV\text{-}Compute}$ algorithm performs correctly (that is, the Verify algorithm is correct and the $\mathsf{Blind}/\mathsf{Unblind}$ algorithms provide information theoretic secrecy). Also assume that access to the $\mathsf{Compute}$ algorithm is available only in a black-box manner via oracle $\mathcal{O}$ that knows the parameter $\mathsf{master\text{-}key}$. We can then define the security of the O-SAOWF as follows. We say that a PPT algorithm $\mathcal{A}$ breaks the O-SAOWF if it is able to "strongly invert" the O-SAOWF and compute inverses in $\mathbb{G}$ having only black-box access to the $\mathsf{Compute}$ algorithm. We call this the *Group Inversion Problem (GIP$_{\mathbb{G}}$)*. Formally, the advantage of $\mathcal{A}$ in solving GIP$_{\mathbb{G}}$ is defined as

$$\text{GIP-Adv}_{\mathcal{A}}(\tau) = \Pr \left[\begin{array}{l} \mathcal{A}^{\mathcal{O}(\mathsf{Compute}(),\ \mathsf{master\text{-}key})}(P, \mathsf{params}) \to P^{-1} : \\ \\ (\mathsf{params}, \mathsf{master\text{-}key}) \xleftarrow{R} \mathsf{Setup}(\tau), \\ (P, \sigma_P) \xleftarrow{R} \mathsf{Sample}(\mathsf{params}) \end{array} \right] \tag{2}$$

Definition 2.5. *We say that algorithm $\mathcal{A}$ $(k_{\mathcal{O}}, \delta, \epsilon)$-breaks the O-SAOWF f if $\mathcal{A}$ runs at most time δ; $\mathcal{A}$ makes at most $k_{\mathcal{O}}$ adaptive queries to the oracle $\mathcal{O}$ implementing the* $\mathsf{Compute}$ *algorithm; and GIP-Adv$_{\mathcal{A}}(\tau)$ is at least ϵ. Alternatively, we say that the O-SAOWF is $(k_{\mathcal{O}}, \delta, \epsilon)$-secure under an adaptive attack if no such algorithm $\mathcal{A}$ exists.*

It is clear that a black-box SAOWF f where we extend the definition of computation in property P6 to include computation by PV-Oracles, is identical to a "real" computable SAOWF f in terms of functionality. However, until now it had been an open question to present even a black-box construction of SAOWFs using PV-Oracles. In this work, we present the first practical construction of a black-box SAOWF based on a PV-Oracle. In other words, our construction allows private (in the information-theoretic sense) and verifiable computation.[5]

Remark 2.6. It should be noted that the above model of an O-SAOWF f that allows black-box computation of the group operation $\star$ on $\mathbb{G}$ using a PV-Oracle is different from a *black-box group*, a notion introduced by Babai and Szemerédi [20] (see also [21]), where access to the entire group $(\mathbb{G}, \star)$ is provided through black-box routines and the representation of group elements is opaque. In contrast, the above model is an example of a *semi black-box group*, since the representation of group elements is not opaque and certain operations like blinding/unblinding, sampling and verification of composition can be done outside of the black-box.

[5]It is noteworthy that our construction of a black-box SAOWF using a PV-Oracle also serves an existence proof of real computable SAOWFs in a way because we achieve almost identical functionality using a black-box construction.

3 The Underlying Primitives

In this section, we give a brief overview of the two main underlying primitives of our construction: (i) composite order bilinear maps, and (ii) the Paillier cryptosystem.

3.1 Bilinear Maps

Let G_1 and G_2 be two cyclic multiplicative groups both of the same 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.

The above properties also imply:

$\hat{e}(ab, c) = \hat{e}(a, c) \cdot \hat{e}(b, c)\ \forall a, b, c \in G_1$

$\hat{e}(a, bc) = \hat{e}(a, b) \cdot \hat{e}(a, c)\ \forall a, b, c \in G_1$

Additionally, we assume that it is easy to *sample* elements from G_1. In a practical implementation, the group G_1 is the set of points on an elliptic curve and G_2 is the multiplicative subgroup of a finite field. The map $\hat{e}$ is derived either from the modified Weil pairing [22, 23] or the Tate pairing [24]. We will assume that the smallest prime factor of n is $\geq 2^{171}$ so that the fastest algorithm for computing discrete logarithms in G_1 (Pollard's rho method [25, p.128]) takes $\geq 2^{85}$ iterations [22].

3.1.1 Problems in Bilinear Maps

It is clear that irrespective of whether n is prime or composite, both G_1 and G_2 have generators. Fix some generator g of G_1 and define the following problems.

Computational Diffie-Hellman Problem [CDHP$_{(g,G_1)}$]: Given $g^x, g^y \in G_1$ for unknowns $x, y \in \mathbb{Z}$, output $g^{xy} \in G_1$.

Decision Diffie-Hellman Problem [DDHP$_{(g,G_1)}$]: Given $g^x, g^y, g^z \in G_1$ for unknowns $x, y, z \in \mathbb{Z}$, output 1 if $z \equiv xy \pmod n$; otherwise output 0.

Inverse Diffie-Hellman Problem [IDHP$_{(g,G_1)}$]: Given $g^x \in G_1$ for unknown $x \in \mathbb{Z}_n^*$, output $g^{1/x} \in G_1$.

The following result was noted by Joux and Nguyen [26].

Lemma 3.1. *DDHP$_{(g,G_1)}$ (the decision Diffie-Hellman problem) is easy.*

Proof. Clearly, from the properties of the mapping, $z \equiv xy \pmod n$ 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 next theorem shows that the computational Diffie-Hellman problem is random self-reducible in the group G_1 if $\phi(n)$ is known.

Theorem 3.2. *IDHP$_{(g,G_1)}$ $\Rightarrow$ CDHP$_{(g,G_1)}$ if $\phi(n)$ is known.*

Proof. We must show that given an IDHP$_{(g,G_1)}$ instance $g^x \in G_1$ for some $x \in \mathbb{Z}_n^*$ and access to a CDHP$_{(g,G_1)}$ oracle, we can efficiently compute $g^{1/x} \in G_1$. This follows from the following facts.

1. **Fact.** From Euler's Theorem [25, p.69] we know that $\forall u \in \mathbb{Z}_n^*\ u^{\phi(n)} \equiv 1 \mod n$. Equivalently, $u^{\phi(n)-1} \equiv 1/u \bmod n$.

2. **Fact.** Given any pair $g^u, g^v \in G_1$ for arbitrary $u, v \in \mathbb{N}$ we can use the $\text{CDHP}_{(g,G_1)}$ oracle to compute $g^{uv} \in G_1$

3. **Fact.** Given any value $g^u \in G_1$, we can use the $\text{CDHP}_{(g,G_1)}$ oracle to compute $g^{u^{2^i}}$ for any $i \in \mathbb{N}$ by the "repeated squaring" method (see [27, p.23] for an example).

Therefore, from g^x we can efficiently compute $h = g^{x^{\phi(n)-1}} \in G_1$ using the $\text{CDHP}_{(g,G_1)}$ oracle and the "repeated squaring and multiply" algorithm of [25, p.71] (via facts 2 and 3). Then from fact 1, $h = g^{1/x}$, and thus h is the required solution. □

Although Theorem 3.2 says that $\text{IDHP}_{(g,G_1)} \Rightarrow \text{CDHP}_{(g,G_1)}$ if $\phi(n)$ is known, it is not clear if the same reduction holds when $\phi(n)$ is unknown. In light of this we make the following hypothesis, necessary for the security of our construction.

Conjuncture 3.1. *$IDHP_{(g,G_1)} \not\Rightarrow CDHP_{(g,G_1)}$ if $\phi(n)$ is unknown.*

3.1.2 BDH Parameter Generator

We will further assume that $n = |G_1| = |G_2| = pq$ where p, q are large primes such that given the product $n = pq$, factoring n is intractable. We refer the reader to [28] for details on generating composite order bilinear maps for any given n that is square free.

Using the idea of [23], we define a Bilinear Diffie-Hellman (BDH) parameter generator as a randomized PPT algorithm $\mathcal{BDH}$ that takes a single parameter $\tau \in \mathbb{N}$ and outputs a tuple $(\hat{e}, G_1, G_2, p, q)$ such that p, q are distinct primes of τ bits each, G_1, G_2 are two cyclic multiplicative groups of the same order pq, and $\hat{e} : G_1 \times G_1 \mapsto G_2$ is a bilinear mapping as defined in section 3.1.

For any PPT algorithm $\mathcal{A}$, denote by $\text{CDHP-Adv}_{\mathcal{A}}(\tau)$, the advantage of $\mathcal{A}$ in solving $\text{CDHP}_{(g,G_1)}$ for some security parameter τ. Formally,

$$\text{CDHP-Adv}_{\mathcal{A}}(\tau) = \Pr \left[\begin{array}{l} \mathcal{A}(\hat{e}, n, G_1, G_2, g, u, v) = g^{xy} : \\ \\ \qquad\qquad (\hat{e}, G_1, G_2, p, q) \xleftarrow{R} \mathcal{BDH}(\tau) \text{ s.t. } |G_1| = |G_2| = pq, \\ n = pq, \quad g \xleftarrow{R} G_1 \text{ s.t. } \langle g \rangle = G_1, \quad (x, y) \xleftarrow{R} {\mathbb{Z}_n}^2, \quad u = g^x, \quad v = g^y \end{array} \right] \tag{3}$$

Similarly, we denote by $\text{IDHP-Adv}_{\mathcal{A}}(\tau)$ the advantage of $\mathcal{A}$ in solving $\text{IDHP}_{(g,G_1)}$. Formally,

$$\text{IDHP-Adv}_{\mathcal{A}}(\tau) = \Pr \left[\begin{array}{l} \mathcal{A}(\hat{e}, n, G_1, G_2, g, u) = g^{1/x} : \\ \\ \qquad\qquad (\hat{e}, G_1, G_2, p, q) \xleftarrow{R} \mathcal{BDH}(\tau) \text{ s.t. } |G_1| = |G_2| = pq, \\ \qquad\qquad n = pq, \quad g \xleftarrow{R} G_1 \text{ s.t. } \langle g \rangle = G_1, \quad x \xleftarrow{R} \mathbb{Z}_n^*, \quad u = g^x \end{array} \right] \tag{4}$$

We will make the following two assumptions for all our constructions.

Diffie-Hellman Assumption: The computation Diffie-Hellman problem ($\text{CDHP}_{(g,G_1)}$) is intractable. In other words, for all PPT algorithms $\mathcal{A}$, $\text{CDHP-Adv}_{\mathcal{A}}(\tau)$ is a negligible function of τ.

Inverse Diffie-Hellman Assumption: The inverse Diffie-Hellman problem ($\text{IDHP}_{(g,G_1)}$) is intractable. In other words, for all PPT algorithms $\mathcal{A}$, $\text{IDHP-Adv}_{\mathcal{A}}(\tau)$ is a negligible function of τ.

3.2 The Paillier Cryptosystem

Our idea of constructing the O-SAOWF is to use an oracle as a "Diffie-Hellman problem solver" in the bilinear group G_1 of composite order n. Since the only known way to solve the Diffie-Hellman problem is to compute discrete logarithms, we provide the discrete logarithms to the oracle in an encrypted form using an asymmetric cryptosystem. The requirement here is that the encryption algorithm $\mathbf{E}$ must possess the following multiplicative homomorphic property: for any messages $m_1, m_2 \in \mathbb{Z}_n^*$, given $\{\mathbf{E}(m_1), m_2\}$ or $\{m_1, \mathbf{E}(m_2)\}$, it must be possible to compute $\mathbf{E}(m_1 m_2 \bmod n)$ directly without knowing the corresponding decryption algorithm $\mathbf{D}$. The Paillier cryptosystem [29] has this property.[6]

The following facts form the basis of the Paillier cryptosystem. Let $n = pq$, where p, q are distinct odd primes. Let $\lambda = \mathrm{lcm}(p-1, q-1)$ and $\phi(n) = (p-1)(q-1)$. For any integer $x \equiv 1 \pmod{n}$ define $L(x) = (x-1)/n$.

1. **Fact 1.** $|\mathbb{Z}_{n^2}^*| = n\phi(n)$
2. **Fact 2.** For all $w \in \mathbb{Z}_{n^2}^*$ it is true that $w^{n\lambda} \equiv 1 \pmod{n^2}$; and $(w^\lambda \bmod n^2) \equiv w^\lambda \equiv 1 \pmod{n}$.
3. **Fact 3.** Let $t \in \mathbb{Z}_{n^2}^*$ such that the order of t is $n\nu$ for some $1 \leq n \leq \lambda$. Then:
   (a) $(t^\nu \bmod n) \equiv 1 \pmod{n}$ and $L(t^\nu \bmod n^2) \in \mathbb{Z}_n^*$.
   (b) For all $x \in \mathbb{Z}$, it is true that
   $$x \equiv \frac{L(t^{x\nu} \bmod n^2)}{L(t^\nu \bmod n^2)} \pmod{n} \tag{5}$$
   Thus given $(t, t^x \bmod n^2)$, we can efficiently compute $x \bmod n$ if ν is known.

We are now ready to describe the Paillier cryptosystem (see [29] for details).

Key Generation: Generate $p, q \xleftarrow{R} \mathbb{N}$, where p, q are large distinct primes. Set $n \leftarrow pq$ and $\lambda \leftarrow \mathrm{lcm}(p-1, q-1)$. Generate $t \xleftarrow{R} \mathbb{Z}_{n^2}^*$ such that the order of t is a non-zero multiple of n. This can be done by checking that $L(t^\lambda \bmod n^2)$ is invertible in $\mathbb{Z}_n$. The public key is (t, n) and the private key is (λ, n).

Encrypt: To encrypt a message $m \in \mathbb{Z}_n$, generate random $r \xleftarrow{R} \mathbb{Z}_n^*$ and set
$$c \leftarrow \mathbf{E}(m) = t^m r^n \bmod n^2$$
The ciphertext is $c \in \mathbb{Z}_{n^2}^*$.

Decrypt: To decrypt, compute
$$m \leftarrow \mathbf{D}(c) = \frac{L(c^\lambda \bmod n^2)}{L(t^\lambda \bmod n^2)} \bmod n$$

The correctness follows from Equation 5. The semantic security of the above encryption scheme is proved under the *Decision Composite Residuosity Assumption* (DCRA) [29], which states that the following problem is hard unless the factors of n are known.

Decision Composite Residousity Problem [DCRP$_n$] Given $x \xleftarrow{R} \mathbb{Z}_{n^2}^*$, output 1 if $\exists y \in \mathbb{Z}_{n^2}^*$ s.t. $x \equiv y^n \pmod{n^2}$ otherwise output 0.

The DCRA is a stronger assumption than factoring [29]. See [32, 33] for a discussion on the bit-security of the Paillier cryptosystem.

3.2.1 Homomorphic Properties

The Paillier cryptosystem has the following homomorphic properties [29].

1. Plaintext multiplication:
$$\forall m_1, m_2 \in \mathbb{Z}_n \quad \mathbf{D}(\mathbf{E}(m_1)^{m_2} \bmod n^2) = \mathbf{D}(\mathbf{E}(m_2)^{m_1} \bmod n^2) = m_1 m_2 \bmod n$$
2. Self Blinding:
$$\forall m \in \mathbb{Z}_n \ \forall r \in \mathbb{N} \quad \mathbf{D}(\mathbf{E}(m) r^n \bmod n^2) = m$$

[6]Although this property is necessary, it is not sufficient; the RSA [30] and Rabin [31] cryptosystems also have this property. However, our construction based on RSA or Rabin is insecure.

4 Our O-SAOWF Construction

Our construction will describe the four algorithms Setup, Sample, $\mathsf{Compute}$ and $\mathsf{PV\text{-}Compute}$ defined in section 2.4. First we describe the Setup procedure. Then we elaborate on the structure of the group $(\mathbb{G}, \star)$ defined by params before describing the remaining algorithms.

4.1 Setup

This algorithm generates the system parameters. The input is a single parameter $\tau \in \mathbb{N}$.

1. Use the BDH parameter generator $\mathcal{BDH}$ of section 3.1.2 to output $(\hat{e}, G_1, G_2, p, q) \leftarrow \mathcal{BDH}(\tau)$, where p, q are large distinct primes of $\approx \tau$ bits each, G_1, G_2 are descriptions of two groups both of order pq and $\hat{e} : G_1 \times G_1 \mapsto G_2$ is a bilinear map (of section 3.1). Then pick a generator $g \xleftarrow{R} G_1$.

2. Set $n \leftarrow pq$ and $\lambda \leftarrow \mathrm{lcm}(p-1, q-1)$. Then generate an element $t \xleftarrow{R} \mathbb{Z}^*_{n^2}$ such that the order of t is a non-zero multiple of n. The pair (t, n) is the public key for the Paillier cryptosystem. The corresponding private key is (λ, n). We will denote the corresponding encryption and decryption algorithms by $\mathbf{E}$ and $\mathbf{D}$ respectively.

3. Generate $\alpha, r \xleftarrow{R} \mathbb{Z}^*_n$. Then set $h \leftarrow g^\alpha \in G_1$ and $\beta \leftarrow \mathbf{E}(\alpha) = t^\alpha r^n \in \mathbb{Z}^*_{n^2}$.

4. Output $\mathsf{params} \leftarrow (\hat{e}, G_1, G_2, g, t, n, h, \beta)$ and $\mathsf{master\text{-}key} \leftarrow \lambda$.

Recall that the $\mathsf{Compute}$ algorithm requires as input the parameter $\mathsf{master\text{-}key}$ and is accessible only as a black-box routine via oracle $\mathcal{O}$ that implements this algorithm. The value $\mathsf{master\text{-}key}$ is sent to $\mathcal{O}$ via a secure channel and the value params is made public.

4.2 Description Of The Underlying Group

From params, the tuple $(\hat{e}, G_1, G_2, g, t, n)$ defines the structure of the group $(\mathbb{G}, \star)$ and the pair (h, β) represents a random element of this group. We now describe the structure of this group.

1. Consider the set $\mathbb{S} \subsetneq G_1$ defined as

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

Clearly, $|\mathbb{S}| = \phi(n) = |\mathbb{Z}^*_n|$ and $\mathbb{S}$ is exactly the set of elements of G_1 having order n.

2. Define the set $\mathbb{G} \subsetneq \mathbb{S} \times \mathbb{Z}^*_{n^2}$ as

$$\mathbb{G} = \{(x, y) | x = g^{\mathbf{D}(y)}\} \tag{6}$$

and define a binary operation $\star$ on $\mathbb{G}$ using the multi-valued mapping

$$\begin{aligned} f : \mathbb{G} \times \mathbb{G} &\mapsto \mathbb{G} \\ (A, B) &\mapsto A \star B \end{aligned}$$

as follows. Let $A = (x_A, y_A)$ and $B = (x_B, y_B)$. Then $A \star B = (x_C, y_C)$, where

$$x_C \leftarrow {x_A}^{\mathbf{D}(y_B)} = g^{\mathbf{D}(y_A)\mathbf{D}(y_B)} = {x_B}^{\mathbf{D}(y_A)} \in G_1 \tag{7}$$

$$y_C \leftarrow \mathbf{E}(\mathbf{D}(y_A)\mathbf{D}(y_B) \bmod n) \in \mathbb{Z}^*_{n^2} \tag{8}$$

Thus, $x_C = g^{\mathbf{D}(y_C)}$ and therefore $(x_C, y_C) \in \mathbb{G}$.

3. Finally, define an equivalence relation $\sim$ on $\mathbb{G}$ as follows. For any $A, B \in \mathbb{G}$, where $A = (x_A, y_A)$ and $B = (x_B, y_B)$, we say that $A \sim B$ if and only if $x_A = x_B$. This relation is symmetric, reflexive and transitive. Thus, it indeed forms an equivalence relation.

We state without proof the following lemmas (which can be easily verified):

Lemma 4.1. *For any $A, B \in \mathbb{G}$, it is true that $A \star B \sim B \star A$. That is, the relation $\sim$ transforms $\star$ into an commutative operation over $\mathbb{G}$.*

Lemma 4.2. *For any $A, B, C \in \mathbb{G}$, it is true that $(A \star B) \star C \sim A \star (B \star C)$. That is, the relation $\sim$ transforms $\star$ into an associative operation over $\mathbb{G}$.*

For any $A \in \mathbb{G}$, denote by $[A] \subsetneq \mathbb{G}$ the *equivalence class* of A with respect to the relation $\sim$. Therefore we can define an equivalence class $[I] \subsetneq \mathbb{G}$ as follows:

$$[I] = \{X | X \sim (g, t) \sim (g, \mathbf{E}(1))\}$$

Lemma 4.3. *For any $[A] \subsetneq \mathbb{G}$, there exists a unique $[B] \subsetneq \mathbb{G}$ such that $[A] \star [B] = [I]$. Additionally, $[A] \star [I] = [A]$.*

It is clear from the above lemmas that the relation $\sim$ transforms the equivalence classes of $\mathbb{G}$ into an abelian group with respect to the binary operation $\star$. The order of this group ($\phi(n)$) is effectively hidden from anyone who does not know the factors of n.

For any $[A] \subsetneq \mathbb{G}$, 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.

$$\left.\begin{array}{rcl} [A]^i \star [A]^j & = & [A]^{i+j} \\ ([A]^i)^j & = & [A]^{ij} \\ [A] \star [A]^{-1} & = & [A]^0 = [I] \\ ([A]^i \star [B]^j)^k & = & [A]^{ik} \star [B]^{jk} \end{array}\right\} \begin{array}{c} \forall\ [A], [B] \subsetneq \mathbb{G} \\ \forall\ i, j, k \in \mathbb{Z} \end{array}$$

We will slightly abuse notation and denote the equivalence class $[A]$ by A. We will use $=$ instead of $\sim$ to indicate that we are working with equivalence classes. For any j given elements $A_1, A_2, \ldots A_j \in \mathbb{G}$, we denote $A_1 \star A_2 \star \ldots A_j$ by

$$\prod_{i=1}^{j} A_i$$

4.3 Properties Of The Underlying Group

We now enumerate some important properties of the group $(\mathbb{G}, \star)$.

1. **Samplability:** $\mathbb{G}$ is efficiently samplable. To sample from $\mathbb{G}$, first generate random $\sigma \xleftarrow{R} \mathbb{Z}_n^*$. Then set $x \leftarrow g^\sigma \in G_1$ and $y \leftarrow \mathbf{E}(\sigma) \in \mathbb{Z}_{n^2}^*$. We see that $(x, y) \in \mathbb{G}$. In this case we call σ, the *sampling information* of (x, y). When we say that $A \in \mathbb{G}$ has been sampled by us, we imply that the sampling information of A is known. The sampling information acts like a trapdoor in our construction.

2. **Trapdoor Computability:** Let $A, B \in \mathbb{G}$ be given. Anyone who has sampled either one of A or B can compute $A \star B$ efficiently as follows:

   Let $A = (x_A, y_A)$ and $B = (x_B, y_B)$ be given. Additionally, we are given $\sigma_A \in \mathbb{Z}_n^*$, the sampling information of A. That is, $x_A = g^{\sigma_A} \in G_1$ and $y_A = \mathbf{E}(\sigma_A) \in \mathbb{Z}_{n^2}^*$. To compute $A \star B$, first generate random $r \xleftarrow{R} \mathbb{Z}_n^*$. Then set $x \leftarrow {x_B}^{\sigma_A} \in G_1$ and $y \leftarrow {y_B}^{\sigma_A} \cdot r^n \in \mathbb{Z}_{n^2}^*$.

   Therefore, $x = {x_B}^{\mathbf{D}(y_A)}$ and due to the homomorphic properties of the Paillier cryptosystem, we find that $y = \mathbf{E}(\sigma_A \mathbf{D}(y_B) \bmod n) = \mathbf{E}(\mathbf{D}(y_A)\mathbf{D}(y_B) \bmod n)$. Thus, $(x, y) = A \star B$.

3. **Trapdoor Strong Invertibility and Exponentiation:** Let $A, B \in \mathbb{G}$ be given. Anyone who has sampled $A \in \mathbb{G}$ can also compute $A^{-1} \star B$ because if $\sigma_A \in \mathbb{Z}_n^*$ is the sampling information for A then $\sigma_A^{-1} \in \mathbb{Z}_n^*$ is the sampling information for A^{-1}. Also, for any $i \in \mathbb{Z}$, the sampling information for $A^i \in \mathbb{G}$ is $(\sigma_A)^i \in \mathbb{Z}_n^*$.

4. **Non-computability:** Let $A, B \in \mathbb{G}$ be given. Anyone who has *not sampled* at least one of $\{A, B, A^{-1}, B^{-1}\}$ cannot compute $A \star B$ without knowledge of λ.
5. **Strong Non-invertibility:** Let $A, B \in \mathbb{G}$ be given. Anyone who has *not sampled* at least one of $\{A, A^{-1}\}$ cannot compute $A^{-1} \star B$ without knowledge of λ.
6. **Indistinguishability:** Let $(x, y) \in G_1 \times \mathbb{Z}^*_{n^2}$ be given. It is infeasible to decide if $(x, y) \overset{?}{\in} \mathbb{G}$ without knowledge of λ.
7. **Black-Box Computability:** Let $A, B \in \mathbb{G}$ be given. Anyone knowing λ has the ability to compute $A \star B$ using equations 8 and 7.
8. **Black-Box Distinguishability:** Let $(x, y) \in G_1 \times \mathbb{Z}^*_{n^2}$ be given. Anyone knowing λ, also has the ability to decide if $(x, y) \overset{?}{\in} \mathbb{G}$ by virtue of equation 6.

4.4 A Concrete O-SAOWF Construction

We now describe a concrete construction of an O-SAOWF under definition 2.4. In addition to the four main algorithms Setup, Sample, Compute, PV-Compute and the three algorithms Verify, Blind and Unblind used as subroutines in PV-Compute, our construction has four 'auxiliary' algorithms Verify-In-Group, Verify-Not-In-Group, TD-Exponentiate and V-Compute. Thus, our construction has a total of eleven algorithms. The Setup algorithm is described in section 4.1 while the Sample algorithm is described in section 4.3, item 1.

A-1.

	Setup
Input:	$\tau \in \mathbb{N}$
Step-1.	Generate $\{\hat{e}, G_1, G_2, g, t, n, h, \beta, \lambda\}$ as described in section 4.1.
Step-2.	Set $\mathsf{params} \leftarrow (\hat{e}, G_1, G_2, g, t, n, h, \beta)$ and $\mathsf{master\text{-}key} \leftarrow \lambda$.
Output:	(params, master-key)

A-2.

	Sample
Input:	params
Step-1.	Generate $\sigma_A, r \overset{R}{\leftarrow} \mathbb{Z}^*_n$
Step-2.	Set $x_A \leftarrow g^{\sigma_A} \in G_1$; $y_A \leftarrow t^{\sigma_A} r^n \bmod n^2 = \mathbf{E}(\sigma_A) \in \mathbb{Z}^*_{n^2}$
Step-3.	Set $A \leftarrow (x_A, y_A) \in \mathbb{G}$
Output:	$(A, \sigma_A) \in \mathbb{G} \times \mathbb{Z}^*_n$ [σ_A is the sampling information of A]

Remark 4.4. From the value params, the pair $(h, \beta) \in \mathbb{G}$ such that its sampling information $\alpha \in \mathbb{Z}^*_n$ is unknown (see section 4.1).

A high level description of the Compute algorithm is given below.

A-3.

	Compute
Input:	(master-key, params, A, B), where $A, B \in G_1 \times \mathbb{Z}^*_{n^2}$
Step-1.	Use $\mathsf{master\text{-}key} = \lambda$ to decide if $(A, B) \overset{?}{\in} \mathbb{G}^2$ [See section 4.3, item 8]
Step-2.	If $(A, B) \notin \mathbb{G}^2$, set $C \leftarrow I \in \mathbb{G}$; otherwise, compute $A \star B$ using λ and set $C \leftarrow A \star B$ [See section 4.3, item 7]
Output:	$C \in \mathbb{G}$

Functionality Of Oracle $\mathcal{O}$: Access to $\mathsf{Compute}$ is provided in a black-box manner via the oracle $\mathcal{O}$ that knows $\mathsf{master\text{-}key}$ and params. The oracle works as follows.

Oracle $\mathcal{O}$

Input: $A, B \in G_1 \times \mathbb{Z}^*_{n^2}$

Step-1. Set $C \leftarrow \mathsf{Compute}(\mathsf{master\text{-}key}, \mathsf{params}, A, B)$

Output: $C \in \mathbb{G}$ [We say $C = \mathcal{O}(A, B)$]

Remark 4.5. A query to oracle $\mathcal{O}$ on inputs $(A, B) \notin \mathbb{G}^2$ requires at most two exponentiations in G_1 and $\mathbb{Z}^*_{n^2}$. On the other hand, if $(A, B) \in \mathbb{G}^2$, the query always involves three exponentiations in G_1 and $\mathbb{Z}^*_{n^2}$. Also, $\mathcal{O}(A, B) = A \star B$ whenever $(A, B) \in \mathbb{G}^2$.

Remark 4.6. Assuming that access to oracle $\mathcal{O}$ is authentic, we can use $\mathcal{O}$ to decide if any given pair $(x, y) \overset{?}{\in} \mathbb{G}$. Additionally we can use $\mathcal{O}$ to compute A^i for any $A \in \mathbb{G}$, $i \in \mathbb{N}$ using the "repeated squaring and multiply" method [25, p.71].

Since access to oracle $\mathcal{O}$ is over an insecure public channel, we cannot assume that oracle replies are authentic. Denote by $\mathcal{O}^*$ the unauthenticated oracle (which could be an active adversary) supposedly claiming to be oracle $\mathcal{O}$.

The following algorithm $\mathsf{Verify\text{-}In\text{-}Group}$ uses oracle $\mathcal{O}^*$ to decide that any given pair $(x, y) \in G_1 \times \mathbb{Z}^*_{n^2}$ is indeed an element of $\mathbb{G}$. If $(x, y) \notin \mathbb{G}$ the algorithm outputs 0 with a high probability.

A-4.

Verify-In-Group

Input: (params, x, y) such that $(x, y) \in G_1 \times \mathbb{Z}^*_{n^2}$

Step-1. Generate $u_1, u_2, v_1, v_2, \xleftarrow{R} \mathbb{Z}_n$ and $w_1, w_2 \xleftarrow{R} \mathbb{Z}^*_n$

Step-2. Set $x_1 \leftarrow x^{u_1} g^{v_1} \in G_1$; $x_2 \leftarrow x^{u_2} g^{v_2} \in G_1$

Step-3. Set $y_1 \leftarrow y^{u_1} t^{v_1} {w_1}^n \bmod n^2$; $y_2 \leftarrow y^{u_2} t^{v_2} {w_2}^n \bmod n^2$; $\mathsf{result} \leftarrow 0$

Step-4. Set $(x', y') \leftarrow \mathcal{O}^*((x_1, y_1), (x_2, y_2))$

Step-5. If $\hat{e}(x', g) = \hat{e}(x_1, x_2)$, set $\mathsf{result} \leftarrow 1$

Output: $\mathsf{result} \in \{0, 1\}$

We prove in appendix A that the above algorithm is sound (under a non-standard assumption). That is, if $(x, y) \notin \mathbb{G}$ then the algorithm outputs 0 with a high probability. However, the converse is not true. Hence, the above algorithm cannot be used to conclude that $(x, y) \notin \mathbb{G}$ if the output is 0.

In some cases, we may need to decide with certainty that a given pair (x, y) is indeed not an element of $\mathbb{G}$. The next algorithm $\mathsf{Verify\text{-}Not\text{-}In\text{-}Group}$ enables us to do this using oracle $\mathcal{O}^*$. If $(x, y) \in \mathbb{G}$ the algorithm outputs 0 with a high probability.[7]

[7] The reader should note that the Verify-Not-In-Group algorithm is never used in any of the protocols discussed in this paper. It is only provided for completeness of our O-SAOWF construction.

A-5.

Verify-Not-In-Group

Input: (params, x, y) such that $(x, y) \in G_1 \times \mathbb{Z}^*_{n^2}$

Step-1. Set a security parameter j and generate a j-bit string $a \xleftarrow{R} \{0,1\}^j$. Set result $\leftarrow 0$. Initialize another j-bit string $b \in \{0,1\}^j$.

Step-2. Repeat for i from 1 to j (denote by a_i and b_i, the i^{th} bits of a and b respectively).

i. If $a_i = 1$, set $(x', y') \xleftarrow{R}$ Sample(params); otherwise, set $(x', y') \leftarrow (x, y)$

ii. Set $b_i \leftarrow$ Verify-In-Group(params, x', y')

Step-3. If $(a = b)$, set result $\leftarrow 1$

Output: result $\in \{0,1\}$

The following lemma shows that the Verify-Not-In-Group algorithm is sound if the Verify-In-Group algorithm is sound.

Lemma 4.1. *If the* Verify-In-Group *algorithm is sound then the* Verify-Not-In-Group *algorithm is also sound.*

Proof. We must show that if the Verify-Not-In-Group algorithm outputs 1 then $(x, y) \notin \mathbb{G}$.

If $(x, y) \in \mathbb{G}$, then (x', y') in step 2 of Verify-Not-In-Group is always an element of $\mathbb{G}$. Now assume that the Verify-In-Group algorithm is sound. Thus, the probability that $a_i = b_i$ is $\frac{1}{2}$ for any i. Also, each bit a_i is independent of other bits. Thus, for a total of j bits, $\Pr[(a_i = b_i)\forall 1 \le i \le j] = \frac{1}{2^j}$. In other words, if $(x, y) \in \mathbb{G}$ the probability that the Verify-Not-In-Group algorithm outputs 1 is $\frac{1}{2^j}$, which can be made arbitrarily small. □

The next algorithm Verify takes as input a 3-tuple (A, B, C), where $A, B \in \mathbb{G}$ and $C \in G_1 \times \mathbb{Z}^*_{n^2}$. It outputs 1 only if $C = A \star B$

A-6.

Verify

Input: (params, A, B, C) such that $A, B \in \mathbb{G}$ and $C \in G_1 \times \mathbb{Z}^*_{n^2}$.
Assume that the input is correct.

Step-1. Set $(x_A, y_A) \leftarrow A$; $(x_B, y_B) \leftarrow B$; $(x_C, y_C) \leftarrow C$; result $\leftarrow 0$

Step-2. If $\hat{e}(x_C, g) = \hat{e}(x_A, x_B)$, set result $\leftarrow$ Verify-In-Group(params, x_C, y_C)

Output: result $\in \{0,1\}$

Clearly, the Verify algorithm is sound if the Verify-In-Group algorithm is sound. We observe that we can remove the function call Verify-In-Group(params, x_C, y_C) in step 2 of the above algorithm (and simply set result $\leftarrow 1$ instead) without introducing any weakness in the construction. However, including this call enables us to reduce the soundness of other related algorithms to the soundness of the Verify-In-Group algorithm.

Algorithm V-Compute takes as input two elements $A, B \in \mathbb{G}$. It uses the Verify-In-Group algorithm as a subroutine and computes $A \star B$ verifiably by querying $\mathcal{O}^*$.

A-7.

V-Compute

Input: (params, A, B) such that $A, B \in \mathbb{G}$. Assume that the input is correct.

Step-1. Set $C \leftarrow \mathcal{O}^*(A, B) \in G_1 \times \mathbb{Z}^*_{n^2}$

Step-2. If Verify$(A, B, C) = 0$, set $C \leftarrow I \in \mathbb{G}$

Output: $C \in \mathbb{G}$

Clearly, the soundness of the above algorithm reduces to the soundness of the Verify algorithm. As a consequence, we state the following theorem which says that if the Verify algorithm is sound then having indirect access to the oracle $\mathcal{O}$ via some active adversary $\mathcal{O}^*$ is the same has having authentic and public access to $\mathcal{O}$.

Theorem 4.2. *If the* Verify *algorithm is sound then $\mathcal{O}$ is a V-Oracle.*

The next algorithm, TD-Exponentiate ("trapdoor-exponentiate") takes as input (i) the sampling information $\sigma_A \in \mathbb{Z}_n^*$ of an element $A \in \mathbb{G}$, (ii) an arbitrary index $i \in \mathbb{Z}$, and (iii) an element $B \in \mathbb{G}$. It outputs $A^i \star B \in \mathbb{G}$. TD-Exponentiate will be primarily used as a subroutine in the Blind and Unblind algorithms.

A-8.

TD-Exponentiate

Input: (params, σ_A, i, B), where $\sigma_A \in \mathbb{Z}_n^*$; $i \in \mathbb{Z}$; $B \in \mathbb{G}$.
Here, σ_A is the sampling information of $A \in \mathbb{G}$. Assume that the input is correct.

Step-1. Generate $r \xleftarrow{R} \mathbb{Z}_n^*$

Step-2. Set $\sigma \leftarrow {\sigma_A}^i \in \mathbb{Z}_n^*$; $(x_B, y_B) \leftarrow B \in G_1 \times \mathbb{Z}_{n^2}^*$

Step-3. Set $x \leftarrow {x_B}^\sigma \in G_1$; $y \leftarrow (y_B)^\sigma r^n = \mathbf{E}(\sigma\mathbf{D}(y_B) \bmod n) \in \mathbb{Z}_{n^2}^*$

Output: $(x, y) \in \mathbb{G}$

The next two algorithms Blind and Unblind work as follows.

Blind takes as input a value $A \in \mathbb{G}$. It generates $B \xleftarrow{R} \mathbb{G}$ and outputs $(A \star B) \in \mathbb{G}$, along with $\sigma_B \in \mathbb{Z}_n^*$, the sampling information of B. Unblind is the inverse of Blind. It takes as input an pair $(A, \sigma_B) \in \mathbb{G} \times \mathbb{Z}_n^*$ and outputs $A \star B^{-1} \in \mathbb{G}$ such that σ_B is the sampling information of $B \in \mathbb{G}$.

A-9.

Blind

Input: (params, A) such that $A \in \mathbb{G}$. Assume that the input is correct.

Step-1. Set $(B, \sigma_B) \xleftarrow{R}$ Sample(params) $\in \mathbb{G} \times \mathbb{Z}_n^*$ [B will be ignored]

Step-2. Set $(x, y) \leftarrow$ TD-Exponentiate(params, $\sigma_B, 1, A) \in \mathbb{G}$

Output: $(x, y, \sigma_B) \in \mathbb{G} \times \mathbb{Z}_n^*$

A-10.

Unblind

Input: (params, A, σ_B), where $A \in \mathbb{G}$ and $\sigma_B \in \mathbb{Z}_n^*$.
Here, σ_B is the sampling information of $B \in \mathbb{G}$. Assume that the input is correct.

Step-1. Set $(x, y) \leftarrow$ TD-Exponentiate(params, $\sigma_B, -1, A) \in \mathbb{G}$

Output: $(x, y) \in \mathbb{G}$

Lemma 4.3. *The* Blind/Unblind *algorithms provide information theoretic secrecy.*

Proof. Clearly, the Blind and Unblind algorithms are inverses of each other. Now, if the output of the Sample algorithm is uniformly distributed over $\mathbb{G}$ then the output of the Blind algorithm is also uniformly distributed over $\mathbb{G}$, independent of the input. $\square$

Algorithm PV-Compute takes as inputs $A, B \in \mathbb{G}$. It uses the Blind, Unblind and and V-Compute algorithms as subroutines to compute $A \star B$ privately and verifiably.

A-11.

	PV-Compute
Input:	(params, A, B) such that $A, B \in \mathbb{G}$. Assume that the input is correct.
Step-1.	Set $(A', \sigma_{A'}) \xleftarrow{R}$ Blind(params, A) $\in \mathbb{G} \times \mathbb{Z}_n^*$
Step-2.	Set $(B', \sigma_{B'}) \xleftarrow{R}$ Blind(params, B) $\in \mathbb{G} \times \mathbb{Z}_n^*$
Step-3.	Set $C' \leftarrow$ V-Compute$(A', B') \in G_1 \times \mathbb{Z}_{n^2}^*$
Step-4.	Set $C \leftarrow$ Unblind(params, Unblind(params, $C', \sigma_{A'}), \sigma_{B'}) \in \mathbb{G}$
Output:	$C \in \mathbb{G}$

Since the Blind/Unblind algorithms provide information theoretic secrecy (lemma 4.3), the soundness of the above algorithm also reduces to the soundness of the Verify algorithm. As a consequence, we state the following theorem which says that if the Verify algorithm is sound then having indirect access to the oracle $\mathcal{O}$ via some active adversary $\mathcal{O}^*$ is the same has having private and authentic access to $\mathcal{O}$.

Theorem 4.4. *If the* Verify *algorithm is sound then* $\mathcal{O}$ *is a PV-Oracle.*

This completes our O-SAOWF construction. Figure 5 gives the dependencies between the eleven algorithms. We can essentially use PV-Compute(params, A, B) to denote $f(A, B)$, where f is a real SAOWF defined using $(\mathbb{G}, \star)$ in Section 2.1. When considering the security, we will assume that $\mathcal{O}$ takes one time unit to respond to each query and that the sum of the number of queries to $\mathcal{O}$ and the running time of an adversary attacking the O-SAOWF is bounded by a polynomial in τ.

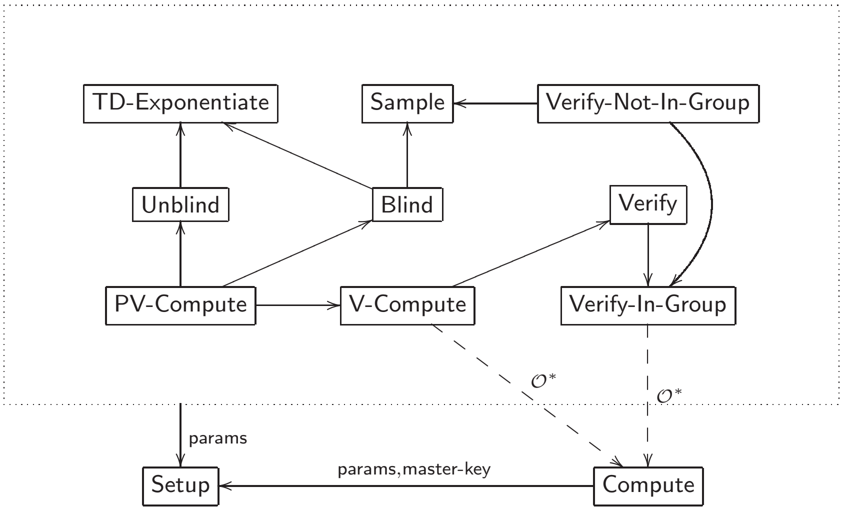


Figure 5: Dependencies between the algorithms.

4.5 Notation

For convenience we will adopt the following shorthand notation.

1 We will denote TD-Exponentiate(params, σ_A, i, B) by $\mathcal{T}(\sigma_A, i, B)$.

2 Since invoking V-Compute is equivalent to making a public query to oracle $\mathcal{O}$ (Theorem 4.2), we will denote V-Compute(params, A, B) simply by $\mathcal{O}(A, B)$.

3 Invoking PV-Compute is equivalent to making a private query to oracle $\mathcal{O}$ (Theorem 4.4). We will denote PV-Compute(params, A, B) by $\widehat{\mathcal{O}}(A, B)$.

4 For any set of k elements $\{A_1, A_2, \ldots A_k\} \subset \mathbb{G}$, we denote by $\left\langle \mathcal{O} \right\rangle_{i=1}^{k} (A_i)$ the value

$$\mathcal{O}(\mathcal{O}(\ldots \mathcal{O}(A_1, A_2), \ldots), A_k) = \prod_{i=1}^{k} A_i$$

Similarly, we denote by $\left\langle \widehat{\mathcal{O}} \right\rangle_{i=1}^{k} (A_i)$ the value

$$\widehat{\mathcal{O}}(\widehat{\mathcal{O}}(\ldots \widehat{\mathcal{O}}(A_1, A_2), \ldots), A_k) = \prod_{i=1}^{k} A_i$$

5 We will denote by $\mathcal{E}(A, i)$ an algorithm to compute A^i for any $A \in \mathbb{G}$ with the repeated squaring method using V-Compute as a subroutine. This algorithm does not provide privacy of inputs. However, the outputs are verifiable.

6 We will denote by $\widehat{\mathcal{E}}(A, i)$ an algorithm to compute A^i for any $A \in \mathbb{G}$ with the repeated squaring method using PV-Compute as a subroutine. This algorithm provides information theoretic privacy of inputs and verifiability of outputs.

Remark 4.7. Computing A^i using algorithms $\mathcal{E}$ and $\widehat{\mathcal{E}}$ will amount to $\approx c \cdot \log i$ queries to oracle $\mathcal{O}$ (for constant c) with the repeated squaring method [25, p.71].

4.6 Security Of The Construction

The oracle is primarily used as a "computing device" in the proofs. We assume that the oracle always functions correctly and keeps the trapdoor information λ secret. Recall that out of params, the pair $(h, \beta) \in \mathbb{G}$. Denote this value by P. The security of our O-SAOWF relies on the difficulty of inverting $\star$ with respect to P. One way to do this would be to extract λ from the oracle. However, this is equivalent to factoring n so we should look at indirect methods for inverting $\star$ (with respect to P) using the oracle. The security of all our constructions reduces to the difficulty of 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 $\beta = \mathbf{E}(\alpha) \in \mathbb{Z}_{n^2}^*$. Given P, compute $P^{-1} = (h', \beta') \in \mathbb{G}$, where $h' = g^{1/\alpha} \in G_1$ and $\beta' = \mathbf{E}(1/\alpha) \in \mathbb{Z}_{n^2}^*$, possibly by using the oracle $\mathcal{O}$.

Computing h' becomes an instance of the inverse Diffie-Hellman problem $\text{IDHP}_{(g,G_1)}$ defined in Section 3.1, which is believed to be hard even if the Diffie-Hellman problem is easy. We hypothesize that any method of reducing $\text{IDHP}_{(g,G_1)}$ to $\text{CDHP}_{(g,G_1)}$ will yield a method of reducing $\text{GIP}_{\mathbb{G}}$ to the oracle $\mathcal{O}$. We define the advantage of an algorithm for solving the group inversion problem as follows.

Definition 4.8. *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:*

$$\textit{GIP-Adv}_{\mathcal{A}}(\tau) = \Pr \left[\begin{array}{l} \mathcal{A}^{\mathcal{O}(\lambda)}(\hat{e}, G_1, G_2, n, g, t, h, \beta) = (g^{1/\alpha}, \mathbf{E}(1/\alpha)) : \\ \\ \hfill (\hat{e}, G_1, G_2, p, q) \xleftarrow{R} \mathcal{BDH}(\tau) \text{ s.t. } |G_1| = |G_2| = pq, \\ \hfill n = pq, \quad \alpha \xleftarrow{R} \mathbb{Z}_n^*, \quad g \xleftarrow{R} G_1 \text{ s.t. } \langle g \rangle = G_1, \\ \hfill t \xleftarrow{R} \mathbb{Z}_{n^2}^* \text{ s.t. } |\langle t \rangle| = n\lambda, \quad h = g^\alpha, \quad \beta = \mathbf{E}(\alpha) \end{array} \right] \qquad (9)$$

Here $\mathcal{BDH}$ is the BDH parameter generator algorithm (Section 3.1.2); $\mathbf{E}$ *denotes the Paillier encryption algorithm with public key (t, n) (Section 3.2), and $\mathcal{O}$ is an oracle implementing the* Compute *algorithm (Section 4.4).*

For any algorithm $\mathcal{A}$, let $\delta_{\mathcal{A}}$ denote the upper-bound on the running time of $\mathcal{A}$, and let $k_{(\mathcal{O},\mathcal{A})}$ denote the upper-bound on the number of queries to oracle $\mathcal{O}$ by $\mathcal{A}$. Our security is based on the following conjuncture.

Conjuncture 4.9. *For any algorithm $\mathcal{A}$ such that $k_{(\mathcal{O},\mathcal{A})}, \delta_{\mathcal{A}} \in Poly(\tau)$, GIP-Adv$_{\mathcal{A}}(\tau)$ is a negligible function in τ. In other words, for all $k_{\mathcal{O}}, \delta, 1/\epsilon \in Poly(\tau)$, the O-SAOWF is $(k_{\mathcal{O}}, \delta, \epsilon)$-secure under an adaptive attack using definition 2.5.*

5 Applications Of O-SAOWFs

In this section we describe three applications of O-SAOWFs: (a) Multiparty-Key Agreement, (b) Signatures and (c) Broadcast encryption (another application, Identity Based Encryption (IBE) is described in appendix C).

5.1 Key Generation (Setup PKI)

To participate in the protocols, each user i must have a certified public key and the corresponding private key. Recall that out of params, the pair $(h, \beta) = P \in \mathbb{G}$. This will serve as a common starting value.

1 User i generates $(X_i, \sigma_{X_i}) \overset{R}{\leftarrow} \mathsf{Sample}(\mathsf{params}) \in \mathbb{G} \times \mathbb{Z}_n^*$. The private key is σ_{X_i}.

2 User i computes the public key $Y_i \leftarrow \mathcal{T}(\sigma_{X_i}, 1, P) = X_i \star P$. The public key is made available in an authentic way (for instance via a certificate).

We give in appendix B a zero-knowledge proof of knowledge that can be used by a Certification Authority (CA) to ascertain that user i indeed knows the private key σ_{X_i} corresponding to the public key Y_i before issuing a certificate.

5.2 Multiparty Key Agreement

In this section, we describe the multiparty key agreement protocol of Rivest, Rabi and Sherman [6] using O-SAOWFs. At a high level, the objective of a multiparty key agreement protocol is to enable a set of users to compute a shared secret key (the *group private key*) such that no one outside the set can compute this key. In our model each group private key also has a corresponding *group public key*, which can be used for join/merge operations and for verifying (group) signatures created using the group private key. We also define a *partial public key* that is used in the intermediate steps for group private key computation.

5.2.1 Key Agreement Protocol

[k **users**] *A set $a = \{1, 2, 3 \ldots k\}$ of k users compute a shared group key.*

1 *Partial public key:* Each user $j \in a$ first computes the partial public key

$$Y_{a\backslash\{j\}} \leftarrow \Big\langle \mathcal{O} \Big\rangle_{i=1;i\neq j}^{k} (Y_i) = \prod_{i=1;i\neq j}^{k} Y_i = P^{k-1} \star \prod_{i=1;i\neq j}^{k} X_i$$

2 *Group Private Key:* Each user $j \in a$ then computes the group private key

$$K_a \leftarrow \mathcal{T}(\sigma_{X_j}, 1, Y_{a\backslash\{j\}}) = X_j \star Y_{a\backslash\{j\}} = P^{k-1} \star \prod_{i=1}^{k} X_i$$

3 *Group Public Key:* The group public key for a is computed by anyone as

$$Y_a \leftarrow \Big\langle \mathcal{O} \Big\rangle_{i=1}^{k} (Y_i) = \prod_{i=1}^{k} Y_i = P^k \star \prod_{i=1}^{k} X_i$$

Thus, the partial public key of user j in set a is the group public key of the set $a\backslash\{j\}$.

5.2.2 Overview Of The Key Agreement Protocol

1 *Complexity*: For a group of k users, $k-2$ oracle queries are required for each user to compute the shared key. Thus, total $k(k-2)$ queries are required for all the k users. However, no specific ordering is required between the users (users can compute the shared key *after* receiving a ciphertext). Additionally, oracle queries can be batched.

2 *Universal Escrow*: Given a public key $Y_i = X_i \star P$, the oracle $\mathcal{O}$ can compute the corresponding private key σ_{X_i}. Therefore, $\mathcal{O}$ has universal escrow capability.

3 *Restricted Private Keys*: Observe that any computation with the private key σ_{X_i} (except inversion w.r.t. X_i) is efficiently possible just from X_i using the $\mathsf{PV\text{-}Compute}$ algorithm.[8] Thus X_i can be considered as a 'restricted' private key corresponding to the 'un-restricted' private key σ_{X_i}, the restriction being the inability to invert $\star$ w.r.t. X_i.

4 *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.

5 *Multiple copies of the Oracle*: An arbitrary number of "copies" of the oracle can be run without any compromise in security.

5.2.3 Join And Merge Operations

Clearly, members can join any group and many groups can merge arbitrarily. For simplicity we only demonstrate the merge operation between two disjoint sets a and b of users.

Example [Merge] *A set a of users merges with another set b of users such that $a \cap b = \emptyset$. Further assume that a has the private key K_a and the public key Y_a. Similarly, b has the private key K_b and the public key Y_b*

1 *Group private key:* Each member $i \in a$ computes $K_{a\cup b} \leftarrow \widehat{\mathcal{O}}(K_a, Y_b)$, while each member $j \in b$ computes $K_{a\cup b} \leftarrow \widehat{\mathcal{O}}(K_b, Y_a)$.

2 *Group public key:* The group public key corresponding to the group private key $K_{a\cup b}$ can be computed as $Y_{a\cup b} \leftarrow \mathcal{O}(Y_a, Y_b) = Y_a \star Y_b$.

In the above merge procedure, we assumed that a and b are disjoint (i.e. they have no common members). In case the sets are not disjoint, we could still use the above merge procedure without any serious drawback as long as this instantiation of O-SAOWF is only used for key agreement (and not for signatures, which are discussed below in Section 5.3). In case the same instantiation of O-SAOWF is also used for signatures, we would require the merge procedure to eliminate duplicate users in the merged set (this can be efficiently done if the intermediate values in the partial public key computation are cached).

Forward Secrecy: Due to the above mentioned merge procedure, the compromise of the group private key of a set a of users compromises the group private key of any other set c of users whenever $c \supsetneq a$. To overcome this weakness, if the private key of group a is compromised, at least one member of a must compute a new public-private key pair. Compromise of a group private key of a set a of users, however, does not compromise the group private key of any set c of users whenever $c \subsetneq a$.

5.2.4 Security Of The Key Agreement Protocol

From the key agreement procedure, it is clear that if the adversary knows the private key of user $i \in a$ then the adversary knows the group private key of the set a of users. Additionally, if the adversary knows the group private key of the set a then the adversary also knows the group private key of any set that properly includes a. Thus, we restrict the adversary to output the private key of any set a of users such

[8] With only a polynomial amount of increase in the number of oracle queries.

that the adversary knows neither the group private key of any proper subset of a nor the private keys of any users in the set a. We show that any algorithm that breaks the key agreement protocol with this restriction can be used to compute P^{-1}. First observe that the secret key K_a for the set $a = \{1, 2, \ldots k\}$ of users is related to the public keys $\{Y_1, Y_2, \ldots Y_k\}$ as:

$$K_a = P^{k-1} \prod_{i=1}^{k} X_i = P^{-1} \star \prod_{i=1}^{k} Y_i \tag{10}$$

We use the security model of security of multiparty key agreement similar to the one used in [7], namely security under a *one-time key attack*. The difference here is that we allow the attacker to choose the set of public keys to attack. Formally, we define a one-time key attack on a multiparty key agreement using game 1.

Game 1

Initialize. To initialize the game, the challenger $\mathcal{C}$ gives a security parameter τ to the adversary. The adversary $\mathcal{A}$ responds with a value $\mu_1 \in \mathbb{N}$

Challenge. The challenger performs the key generation phase and gives a set $\{Y_1, Y_2, \ldots Y_{\mu_1}\}$ of μ_1 public keys to $\mathcal{A}$.

Output. Eventually $\mathcal{A}$ outputs a pair $\langle a, K_a \rangle$.

Result: $\mathcal{A}$ wins the game if $a \subseteq \{1, 2, \ldots \mu_1\}$ and K_a is the group private key of a.

Definition 5.1. *We say that adversary $\mathcal{A}$ $(\mu_1, \delta_1, \epsilon_1)$-breaks the key agreement protocol in the* one-time key attack *if for a total of μ_1 public keys output in the setup phase $\mathcal{A}$ runs at most time δ_1 and the probability of $\mathcal{A}$ winning game 1 is at least ϵ_1. Alternatively, we say that the key agreement protocol is $(\mu_1, \delta_1, \epsilon_1)$-secure under a one-time key attack if no such adversary exists.*

The next theorem shows that the key agreement protocol is secure under a one-time key attack if the group inversion problem is hard.

Theorem 5.1. *Let the O-SAOWF be $(\cdot, \delta, \epsilon)$-secure under an adaptive attack. Then the multiparty key agreement protocol is $(\mu_1, \delta_1, \epsilon_1)$-secure in a one-time key attack, where $\delta \leq \delta_1 + \Theta(c_1\mu_1)$ and $\epsilon = \epsilon_1$. Here, c_1 is the time for a multiplication in $\mathbb{Z}_n^*$.*

Proof. Let the O-SAOWF be $(\cdot, \delta, \epsilon)$-secure under an adaptive attack and let $\mathcal{A}$ be an algorithm that $(\mu_1, \delta_1, \epsilon_1)$-breaks the key agreement protocol in a one-time key attack. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to solve $\text{GIP}_{\mathbb{G}}$ in at most δ time with probability at least ϵ, thus arriving at a contradiction. The input to $\mathcal{B}$ is $P \in \mathbb{G}$ and its goal is to output P^{-1}. $\mathcal{B}$ simulates the challenger of game 1 and runs algorithm $\mathcal{A}$ as follows.

Initialize. $\mathcal{B}$ gives the security parameter τ to $\mathcal{A}$ who replies with μ_1.

Challenge. $\mathcal{B}$ generates $(Y_1, \sigma_{Y_1}), (Y_2, \sigma_{Y_2}), \ldots (Y_{\mu_1}, \sigma_{Y_{\mu_1}}) \xleftarrow{R} \mathsf{Sample}(\mathsf{params}) \in \mathbb{G} \times \mathbb{Z}_n^*$ and gives the $(\mu_1 + 1)$-tuple $(Y_1, Y_2, \ldots Y_{\mu_1}, P)$ to $\mathcal{A}$.

Output. Eventually $\mathcal{A}$ outputs a pair $\langle a, K_a \rangle$.

Result: If $\langle a, K_a \rangle$ is a winning configuration, then $a \subseteq \{1, 2, \ldots \mu_1\}$ and $K_a = P^{-1} \star \prod_{i \in a} Y_i$ by virtue of equation 10. Clearly, the simulation provided by $\mathcal{B}$ is perfect. Algorithm $\mathcal{B}$ then proceeds as follows:

i. If $\langle a, K_a \rangle$ is not a winning configuration, $\mathcal{B}$ reports failure and terminates.

ii. We know that $\langle a, K_a \rangle$ is a winning configuration. Algorithm $\mathcal{B}$ sets $\sigma_Y \leftarrow \prod_{i \in a} \sigma_{Y_i} \bmod n$. Thus, σ_Y is the sampling information of $\prod_{i \in a} Y_i$ (see Section 4.3, item 3).

iii. $\mathcal{B}$ sets $\mathsf{result} \leftarrow \mathcal{T}(\sigma_Y, -1, K_a)$ and outputs result.

Algorithm $\mathcal{B}$ is correct because

$$\mathcal{T}(\sigma_Y, -1, K_a) = (\prod_{i\in a} Y_i)^{-1} \star K_a = (\prod_{i\in a} Y_i)^{-1} \star P^{-1} \star \prod_{i\in a} Y_i = P^{-1}$$

The running time of $\mathcal{B}$ is the running time of $\mathcal{A}$ plus the time required for generating the μ_1 public keys; the time required for computing $\mathcal{T}$; and the time required for at most μ_1 multiplications in $\mathbb{Z}_n^*$. The probability of $\mathcal{B}$'s success is the same as the probability of $\mathcal{A}$'s success. This gives the bounds. $\square$

5.3 Signatures

As noted in [11], SAOWFs give rise to signature schemes. Here, we describe two signature schemes using O-SAOWFs: ordinary signatures and multi-user signatures. A signature scheme consists of three algorithms KeyGen, Sign and VerifySig, where the algorithms have their usual constraints [22]. Our message space is $\mathbb{N}$.

5.3.1 Single-User Signatures

This is a variation of the scheme for single-user signatures described in [6].

KeyGen. This algorithm is described in Section 5.1. The private key of user i is $\sigma_{X_i} \in \mathbb{Z}_n^*$. The public key is $Y_i = X_i \star P \in \mathbb{G}$.

Sign. Let $m \in \mathbb{N}$ be the message. To sign m, user i computes the signature $S_{(i,m)}$ as:

$$S_{(i,m)} \leftarrow \mathcal{T}(\sigma_{X_i}, m, P) = {X_i}^m \star P$$

VerifySig. To verify a signature $S_{(i,m)}$ of user i on message m, we check if the following holds:

$$\mathcal{E}(Y_i, m) \stackrel{?}{=} \mathcal{O}(S_{(i,m)}, \mathcal{E}(P, m-1))$$

In other words, we check if ${Y_i}^m \stackrel{?}{=} S_{(i.m)} \star P^{m-1}$

5.3.2 Multi-User And Ring Signatures

The above construction of single-user signatures can be trivially extended to multi-user signatures. To sign messages, members of a group must share a secret group key.

KeyGen. This algorithm is described in Section 5.2. Without loss of generality, assume that any of the set $a = \{1, 2, \ldots k\}$ of users want to independently sign messages using the group private key $K_a = P^{k-1} \star \prod_{i=1}^k X_i$ such that the signatures can be verified using the group public key $Y_a = \prod_{i=1}^k Y_i$.

Sign. Let $m \in \mathbb{N}$ be the message. To sign m, any member $i \in a$ computes the signature $S_{(a,m)}$ as:

$$S_{(a,m)} \leftarrow \widehat{\mathcal{O}}(\widehat{\mathcal{E}}(K_a, m), P) = {K_a}^m \star P$$

VerifySig. To verify a signature $S_{(a,m)}$ of user $i \in a$ on message m, we check if the following holds:

$$\mathcal{E}(Y_a, m) \stackrel{?}{=} \mathcal{O}(S_{(a,m)}, \mathcal{E}(P, m-1))$$

In other words, we check if ${Y_a}^m \stackrel{?}{=} S_{(a,m)} \star P^{m-1}$

Given a signature of some set a, 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 is an example of ring signatures [34].

5.3.3 Security Of The Signature Schemes

The strongest model for security of signatures is security against *existential forgery* under an *adaptive chosen message attack* [22], where the attacker is required to output a successful forgery under the challenge public key after having access to the signing oracle. However, we only prove the security of our schemes in a weaker model that we call security against *existential forgery* under a *non-adaptive chosen message attack*. In a non-adaptive attack, the attacker is not allowed to make any signature queries. We define this using the following game between the challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. For technical convenience we bound the value of the message in the forgery to be $\leq n$, although this bound could be any polynomial function of n

Game 2

Initialize. To initialize the game, the challenger gives a security parameter τ to the adversary. The adversary $\mathcal{A}$ outputs $\mu_2 \in \mathbb{N}$.

Challenge. The challenger $\mathcal{C}$ performs the key generation phase and gives a set $\{Y_1, Y_2, \dots Y_{\mu_2}\}$ of μ_2 public keys to $\mathcal{A}$.

Output. Eventually $\mathcal{A}$ outputs a tuple $\langle a, S_{(a,m)}, m \rangle$.

Result: $\mathcal{A}$ wins the game if $a \subseteq \{1, 2, \dots \mu_2\}$ and $S_{(a,m)}$ is a valid signature by a on the message m and $m \leq n$.

Definition 5.2. *We say that adversary $\mathcal{A}$ $(\mu_2, \delta_2, \epsilon_2)$-breaks the signature scheme in a* non-adaptive chosen message attack *if for a total of μ_2 public keys output in the setup phase $\mathcal{A}$ runs at most time δ_2 and the probability of $\mathcal{A}$ winning game 2 is at least ϵ_2. Alternatively, we say that the signature scheme is $(\mu_2, \delta_2, \epsilon_2)$-secure under a non-adaptive chosen message attack if no such adversary exists.*

The next theorem shows that any algorithm that is successful in existential forgery of signatures under a non-adaptive chosen message attack can be used to solve $\mathrm{GIP}_{\mathbb{G}}$. First observe that $S_{(a,m)}$ can be rewritten as

$$S_{(a,m)} = {K_a}^m \star P = P^{1-m} \star (\prod_{i=1}^{k} Y_i)^m \tag{11}$$

Also note that game 2 considers both single and multi-user signatures.

Theorem 5.2. *If there exists an algorithm $\mathcal{A}$ that $(\mu_2, \delta_2, \epsilon_2)$-breaks the signature scheme under a non-adaptive chosen message attack then there exists an algorithm $\mathcal{B}$ that $(k_{\mathcal{O}}, \delta, \epsilon)$-breaks the O-SAOWF under an adaptive attack, where $k_{\mathcal{O}} < \Theta(3 \log n)$; $\delta \leq \delta_2 + \Theta(c_2 \mu_2)$; and $\epsilon = \epsilon_2$. Here, c_2 is a constant that depends on $\mathbb{G}$.*

Proof. Let the O-SAOWF be $(k_{\mathcal{O}}, \delta, \epsilon)$-secure under an adaptive attack and let $\mathcal{A}$ be an algorithm that $(\mu_2, \delta_2, \epsilon_2)$-breaks the signature scheme in a non-adaptive chosen message attack. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to solve $\mathrm{GIP}_{\mathbb{G}}$ in at most δ time with probability at least ϵ, thus arriving at a contradiction. The input to $\mathcal{B}$ is $P \in \mathbb{G}$ and its goal is to output P^{-1}. $\mathcal{B}$ simulates the challenger of game 2 and runs algorithm $\mathcal{A}$.

Initialize. $\mathcal{B}$ gives the parameter τ to $\mathcal{A}$, who outputs $\mu_2 \in \mathbb{N}$.

Challenge. $\mathcal{B}$ generates $(Y_1, \sigma_{Y_1}), (Y_2, \sigma_{Y_2}), \dots (Y_j, \sigma_{Y_{\mu_2}}) \xleftarrow{R} \mathsf{Sample}(\mathsf{params}) \in \mathbb{G} \times \mathbb{Z}_n^*$ and gives the $(\mu_2 + 1)$-tuple $(Y_1, Y_2, \dots Y_{\mu_2}, P)$ as the input to $\mathcal{A}$.

Output. Finally $\mathcal{A}$ outputs a tuple $\langle a, S_{(a,m)}, m \rangle$.

Result: If the tuple $\langle a, S_{(a,m)}, m \rangle$ represents a winning configuration, then $a \subseteq \{1, 2, \dots \mu_2\}$, $S_{(a,m)} = P^{1-m} \star (\prod_{i \in a} Y_i)^m$ (by virtue of equation 11), and $m \leq n$. Algorithm $\mathcal{B}$ then proceeds as follows:

i. If $\langle a, S_{(a,m)}, m \rangle$ not a winning configuration, algorithm $\mathcal{B}$ reports failure and terminates.

ii. We know that $a \subseteq \{1, 2, \ldots \mu_2\}$ and $S_{(a,m)} = P^{1-m} \star (\prod_{i\in a} Y_i)^m$. Algorithm $\mathcal{B}$ sets $C \leftarrow \mathcal{E}(P, m-2) = P^{m-2}$ and $\sigma_Y \leftarrow \prod_{i\in a} \sigma_{Y_i} \bmod n$. Thus, σ_Y is the sampling information of $\prod_{i\in a} Y_i$ (see Section 4.3, item 3).

iii. Finally, $\mathcal{B}$ sets $\mathsf{result} \leftarrow \mathcal{T}(\sigma_Y, -m, \mathcal{O}(S_{(a,m)}, C))$ and outputs result.

Algorithm $\mathcal{B}$ is correct because

$$\mathcal{T}(\sigma_Y, -m, \mathcal{O}(S_{(a,m)}, C)) = (\prod_{i\in a} Y_i)^{-m} \star S_{(a,m)} \star C$$

$$= (\prod_{i\in a} Y_i)^{-m} \star (P^{1-m} \star \prod_{i\in a} Y_i) \star (P^{m-2}) = P^{-1}$$

The running time of $\mathcal{B}$ is the running time of $\mathcal{A}$ plus the time required for generating the μ_2 public keys; the time required for computing $\mathcal{T}$ once; and the time required for at most μ_2 multiplications in $\mathbb{Z}_n^*$. Therefore $\delta \leq \delta_2 + \Theta(c_2\mu_2)$, where c_2 is the time for generating one public key plus the time for one multiplication in $\mathbb{Z}_n^*$.

Clearly, the simulation provided to $\mathcal{A}$ is perfect. Hence the probability of $\mathcal{B}$'s success is the same as the probability of $\mathcal{A}$'s success. Finally, $\mathcal{B}$ queries the oracle for computing $\mathcal{O}(S_{(a,m)}, C)$ and $\mathcal{E}(P, m-2)$. This amounts to a maximum of $\Theta(c_1 \log m)$ queries for some constant $c_1 < 3$. Considering that $m \leq n$, we have the required bounds. □

5.4 Broadcast Encryption

In a broadcast encryption scheme [35], anyone can encrypt a message addressed to a closed set of users using their public keys such that only those users have the ability to decrypt the message (we do not consider schemes that allow *traitor tracing* [36]). Using our method, the size of ciphertexts and public/private keys is $O(1)$ and for a set of k users, a total of $O(k)$ calls to the oracle $\mathcal{O}$ are required for encryption and decryption. A broadcast encryption scheme consists of four algorithms BC-Setup, BC-KeyGen, BC-Encrypt and BC-Decrypt, where the algorithms have their usual meanings and constraints [35]. (We use the prefix 'BC' to indicate 'broadcast').

BC-Setup This algorithm is used to set up the initial system parameters and is described in Section 5.1, where the initial public key infrastructure is created for individual users. We additionally require a cryptographic hash function $\mathcal{H} : G_1 \mapsto \{0,1\}^l$, that will be treated as a random oracle in the proofs.[9] The message space is $\{0,1\}^l$, where $l \leq \log_2 n$.

BC-KeyGen. Without loss of generality, assume that messages will be encrypted to any arbitrary set $a = \{1, 2, \ldots k\}$ of k users with public keys $\{Y_1, Y_2, \ldots Y_k\}$.

Encryption Key: The sender of the message generates the group public key $Y_a = \prod_{i=1}^{k} Y_i$ by making $k-1$ oracle queries (as described in Section 5.2).

Decryption Key: Any receiver $j \in a$ must independently compute the group private key $K_a = P^{k-1} \star \prod_{i=1}^{k} X_i = P^{-1} \star \prod_{i=1}^{k} Y_i$ by making $k-2$ oracle queries (as described in Section 5.2).

BC-Encrypt. To encrypt $m \in \{0,1\}^l$ to the set $a = \{1, 2, \ldots k\}$ of k users with group public key Y_a, generate $(R, \sigma_R) \xleftarrow{R} \mathsf{Sample}(\mathsf{params}) \in \mathbb{G} \times \mathbb{Z}_n^*$ and compute

$$c_1 \leftarrow m \oplus \mathcal{H}(\mathcal{T}(\sigma_R, 1, Y_a)) = m \oplus \mathcal{H}(R \star Y_a)$$

$$C_2 \leftarrow \mathcal{T}(\sigma_R, 1, P) = R \star P$$

Here $\oplus$ denotes the XOR operator. The ciphertext is $C = (c_1, C_2) \in \{0,1\}^l \times \mathbb{G}$.

[9] To construct this hash function, let $A = (x, y) \in \mathbb{G} \in G_1 \times \mathbb{Z}_n^*$ be some input and let $\mathcal{H}_1 : G_1 \mapsto \{0,1\}^l$ be a hash function. Then $\mathcal{H}(A) = \mathcal{H}_1(x)$.

BC-Decrypt. To decrypt ciphertext (c_1, C_2) using group private key K_a, compute

$$m \leftarrow c_1 \oplus \mathcal{H}(\widehat{\mathcal{O}}(K_a, C_2)) = c_1 \oplus \mathcal{H}(K_a \star C_2)$$

The decryption is correct, because for a legitimate ciphertext we have

$$(K_a \star C_2) = (P^{k-1} \star \prod_{i=1}^{k} X_i) \star (R \star P) = R \star P^k \star \prod_{i=1}^{k} X_i = R \star Y_a$$

5.4.1 Security Of Broadcast Encryption

We use a restricted model for security called security under an adaptive chosen plaintext attack (IND-CPA). In this model, we fix some arbitrary set $a = \{1, 2, \ldots k\}$ of k users and require the adversary to attack the semantic security of the scheme without access to a decryption oracle. However, we allow the adversary to choose the subset of keys it is attacking. Since full security in the sense of adaptive chosen ciphertext attacks (IND-CCA) in the random oracle model can be achieved using the Fujisaki-Okamoto transformation [37], we only prove security in the IND-CPA model. IND-CPA security of a broadcast encryption scheme is defined using the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Game 3

Initialize. The challenger $\mathcal{C}$ gives a security parameter τ to the adversary $\mathcal{A}$, who outputs a tuple μ_3. The challenger performs the key generation phase and gives a set $\{Y_1, Y_2, \ldots Y_{\mu_3}\}$ of μ_3 public keys to $\mathcal{A}$.

Challenge. $\mathcal{A}$ generates two messages m_0, m_1 along with a set $a \subseteq \{1, 2, \ldots \mu_3\}$ of users. The challenger chooses a bit $b \overset{R}{\leftarrow} \{0, 1\}$ and outputs the encryption of m_b under the group public key Y_a of a.

Guess. Eventually $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$

Result: $\mathcal{A}$ wins the game if $b = b'$.

We refer to such an adversary $\mathcal{A}$ as an IND-CPA adversary. We define $\mathcal{A}$'s advantage in attacking the broadcast encryption scheme $\text{Adv-cpa}_{\mathcal{A}}(\tau)$ as:

$$\text{Adv-cpa}_{\mathcal{A}}(\tau) = \left| \Pr[b = b'] - \frac{1}{2} \right|,$$

where the probability is taken over the random coin tosses of $\mathcal{C}$ and $\mathcal{A}$.

Definition 5.3. *Let $\mathcal{H}$ be a random oracle. We say that an* IND-CPA *adversary $\mathcal{A}$ $(\mu_3, \delta_3, k_3, \epsilon_3)$-breaks the broadcast encryption scheme in an* adaptive chosen plaintext attack *if for a total of μ_3 public keys output in the setup phase $\mathcal{A}$ runs at most time δ_3; $\mathcal{A}$ makes at most k_3 queries to the oracle for $\mathcal{H}$; and Adv-cpa$_{\mathcal{A}}(\tau)$ at least ϵ_3. Alternatively, we say that the broadcast encryption scheme is $(\mu_3, \delta_3, k_3, \epsilon_3)$-secure under an adaptive chosen plaintext attack if no such adversary $\mathcal{A}$ exists.*

The next theorem shows that any IND-CPA adversary $\mathcal{A}$ with non-negligible advantage $\text{Adv-cpa}_{\mathcal{A}}(\tau)$ in the random oracle model can be used to solve the group inversion problem with non-negligible advantage. The proof is similar to the proof of [23, lemma 4.3]

Theorem 5.3. *Let $\mathcal{H}$ be a random oracle and let the O-SAOWF be $(k_{\mathcal{O}}, \delta, \epsilon)$-secure under an adaptive attack. Then the broadcast encryption scheme is $(\mu_3, \delta_3, k_3, \epsilon_3)$-secure under an adaptive chosen plaintext attack, where $k_{\mathcal{O}} \leq k_3$; $\delta \leq \delta_3 + \Theta(c_1\mu_3) + \Theta(c_2k_3)$; and $\epsilon \geq 2\epsilon_3$. Here, c_1 is the time for one multiplication in $\mathbb{Z}_n^*$, and c_2 is a constant that depends on the oracle $\mathcal{O}$.*

Proof. Let the O-SAOWF be $(k_{\mathcal{O}}, \delta, \epsilon)$-secure under an adaptive attack and let $\mathcal{A}$ be an algorithm that $(\mu_3, \delta_3, k_3, \epsilon_3)$-breaks the key agreement protocol in an adaptive chosen plaintext attack. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to solve $\mathrm{GIP}_{\mathbb{G}}$ in at most δ time with probability at least ϵ and making at most $k_{\mathcal{O}}$ oracle queries, thus arriving at a contradiction. The input to $\mathcal{B}$ is $P \in \mathbb{G}$ and its goal is to output P^{-1}. $\mathcal{B}$ simulates the challenger of game 3 and runs $\mathcal{A}$.

Initialize. $\mathcal{B}$ gives the security parameter τ to $\mathcal{A}$ who replies with μ_3. $\mathcal{B}$ generates

$$(Y_1, \sigma_{Y_1}), (Y_2, \sigma_{Y_2}), \ldots (Y_{\mu_3}, \sigma_{Y_{\mu_3}}) \xleftarrow{R} \mathsf{Sample}(\mathsf{params}) \in \mathbb{G} \times \mathbb{Z}_n^*,$$

and gives the $(\mu_3 + 1)$-tuple $(Y_1, Y_2, \ldots Y_{\mu_3}, P)$ to $\mathcal{A}$.

$\mathcal{H}$-queries. At any time, $\mathcal{A}$ may query the random oracle $\mathcal{H}$. To respond to these queries, $\mathcal{B}$ maintains a list of tuples called the $\mathcal{H}^{list}$. Each entry in this list is a tuple of the form $\langle Z_j, \mathcal{H}_j \rangle$. Initially this list is empty. To respond to a $\mathcal{H}$ query on Z_i, algorithm $\mathcal{B}$ does the following:

i. If the query Z_i already appears on the $\mathcal{H}^{list}$ in a tuple $\langle Z_i, \mathcal{H}_i \rangle$, then $\mathcal{B}$ responds with $\mathcal{H}(Z_i) = \mathcal{H}_i$.

ii. Otherwise, $\mathcal{B}$ just picks a random string $\mathcal{H}_i \in \{0,1\}^l$ and adds the tuple $\langle Z_i, \mathcal{H}_i \rangle$ to the $\mathcal{H}^{list}$. It responds with $\mathcal{H}(Z_i) = \mathcal{H}_i$.

Challenge. $\mathcal{A}$ generates two messages m_0, m_1 along with a set $a \subseteq \{1, 2, \ldots \mu_3\}$ and sends the tuple (m_0, m_1, a) to $\mathcal{B}$. Algorithm $\mathcal{B}$ picks random $c_1 \in \{0,1\}^l$; generates $(C_2, \sigma_{C_2}) \xleftarrow{R} \mathsf{Sample}$; defines the ciphertext $C = (c_1, C_2)$; and gives C as the challenge ciphertext to $\mathcal{A}$. Observe that by definition the decryption of C is $c_1 \oplus \mathcal{H}(P^{-1} \star C_2 \star \prod_{i \in a} Y_i)$.

Algorithm $\mathcal{B}$ also computes (and keeps secret) $\sigma_W \leftarrow \sigma_{C_2} \cdot \prod_{i \in a} \sigma_{Y_i} \bmod n$. Clearly, σ_W is the sampling information of $W = C_2 \star \prod_{i \in a} Y_i$ (see Section 4.3, item 3).

Guess. Eventually $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$. At this point, $\mathcal{B}$ searches the $\mathcal{H}^{list}$ to find a tuple $\langle Z_j, \mathcal{H}_j \rangle$ such that

$$\mathcal{O}(Z_j, P) = W \tag{12}$$

If such a tuple does not exist in the $\mathcal{H}^{list}$, algorithm $\mathcal{B}$ reports failure and terminates. Otherwise, $\mathcal{B}$ sets $\mathsf{result} \leftarrow \mathcal{T}(\sigma_W, -1, Z_j) = W^{-1} \star Z_j$. Algorithm $\mathcal{B}$ outputs result as the solution to the $\mathrm{GIP}_{\mathbb{G}}$ instance.

Clearly, the simulation provided by algorithm $\mathcal{B}$ is perfect. Therefore, from claims 1 and 2 in the proof of [23, lemma 4.2], we can conclude that

$$\Pr\left[\text{a tuple } \langle Z_j, \mathcal{H}_j \rangle \text{ appears in the } \mathcal{H}^{list} \text{ such that equation 12 is satisfied}\right] \geq 2\epsilon_3$$

Thus $\epsilon \geq 2\epsilon_3$. Also, algorithm $\mathcal{B}$ makes at most k_3 queries to $\mathcal{O}$. The running time of $\mathcal{B}$ is the running time of $\mathcal{A}$ plus the time required for generating the μ_3 public keys; the time required for computing $\mathcal{T}$; the time required for searching up to k_3 entries in the $\mathcal{H}^{list}$; and the time required for at most μ_3 multiplications in $\mathbb{Z}_n^*$. Therefore $\delta \leq \delta_3 + \Theta(c_1\mu_3) + \Theta(c_2 k_3)$, where c_1 is the time for one multiplication in $\mathbb{Z}_n^*$, and c_2 is the time for checking one entry of the $\mathcal{H}^{list}$. Checking each entry in this list involves a query to $\mathcal{O}$. Combining the above results, we have the required bounds $\square$

6 Implementation And Efficiency

In this section, we will briefly touch upon issues relating to implementation and efficiency of our primitive. Although our construction of O-SAOWF has other applications as demonstrated, we feel that its primary use 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 between the users.

6.1 Key Size

Factoring n enables an attacker to solve $\text{GIP}_{\mathbb{G}}$. Based on the current state-of-the-art factoring algorithms, we suggest using the modulus n of about 313 decimal digits (≈ 1024 bits) for moderate security applications.[10] This also makes computing discrete logarithms in G_1 intractable using Pollard's rho method [25, p.128]. Using these parameters elements of $\mathbb{G}$ can be represented with at most ≈ 384 bytes. The public keys Y_i of Section 5.1, which are elements of $\mathbb{G}$ will be 384 bytes each. The private keys σ_{X_i} on the other hand, which are elements of $\mathbb{Z}_n^*$ will be 128 bytes.

6.2 Query Overhead

In all the above protocols, we have been working in the equivalence classes of $\mathbb{G}$ rather than the individual elements themselves. For any $A = (x, y) \in \mathbb{G}$, the equivalence class $[A]$ is completely characterized by the first element x. The second element y is used only as an 'auxiliary' input for the oracle, and is useless to anyone who does not know the factorization of n. Thus, verification of the second element cannot provide additional security. With this consideration in mind, we slightly modify the Verify algorithm of Section 4.4 and remove the call to the Verify-In-Group subroutine, since computing the bilinear pairing allows verification of the first element x. The computation overhead is given in table 1.

Algorithm	Exp G_1	Exp $\mathbb{Z}_{n^2}^*$	Multi $\mathbb{Z}_{n^2}^*$	Multi $\mathbb{Z}_n^*$	Pairing
Compute	3	4	1	2	-
V-Compute	3	4	1	2	2
PV-Compute	5	5	1	2	2

Table 1: Computation involved in a query

6.3 Fast Paillier Decryption

Since each computation of $\star$ requires two decryptions, it is desirable to obtain a faster decryption procedure. In [29, Section 6], a fast variant of the Paillier cryptosystem is presented where decryption does not require the factors of n and runs with almost quadratic complexity. In this variant, $\lambda = (p-1)(q-1)$ has a large prime factor ν. The public key is (t, n) such that the order of $t \in \mathbb{Z}_{n^2}^*$ is νn. The private key is ν. Encryption and decryption is described below.

Encrypt Plaintext is $m \in \mathbb{Z}_n$. Generate $r \xleftarrow{R} \mathbb{Z}_n^*$ and compute $c = t^{m+nr} \bmod n^2$. The ciphertext is c.

Decrypt Ciphertext is $c \in \mathbb{Z}_{n^2}^*$. Compute $m = \frac{L(c^\nu \bmod n^2)}{L(t^\nu \bmod n^2)} \bmod n$.

Semantic security of this variant does not depend on the DCRA (Section 3.2) but instead relies on the weaker *Decisional Partial Discrete Logarithm Assumption* (DPDLA) [29, Theorem 20], which states that the following problem is hard.

Decisional Partial Discrete Logarithm Problem ($\mathbf{DPDLP}_{(t,n)}$) Fix any $t \in \mathbb{Z}_{n^2}^*$ such that the order of t is νn for unknown ν. Given $w \in \langle t \rangle$ and $x \in \mathbb{Z}_n$, output 1 if $\exists y \in \mathbb{Z}_n^*$ s.t $w \equiv t^x y^n \pmod{n^2}$ otherwise output 0.

6.4 Verifiability Of The Oracle

In this section we discuss two simplification of the construction. The above construction used bilinear maps to *implicitly* verify the output of the black-box computation. However, since we already trust

[10] See 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 1024 bit keys will be secure till the year 2010 while 2048 bit keys will be secure till the year 2030.

the black-box to keep the trapdoor information (λ) secret, we can also trust the black-box to correctly evaluate the SAOWF.[11]

1 Our first simplification is to authenticate (i.e. verify) the output of the oracle via an existentially unforgeable signature scheme. In this construction, instead of the bilinear group G_1, we use a finite field having a multiplicative subgroup of order n. The set $\mathbb{S}$ defined in Section 4.2 is then the $\phi(n)$ elements of this field of order n. Note that in addition to the trapdoor information λ, the oracle will also contain a private key for creating signatures. For all inputs $(X, Y) \in \mathbb{G}^2$, the oracle responds with $Z = X \star Y \in \mathbb{G}$ and a signature $Sig_{X,Y,Z}$ on the tuple (X, Y, Z).

2 Our second simplification is to do away with the finite field and the signature scheme altogether by using a deterministic variant of Paillier encryption defined in Section 3.2 and setting $r = 1$. In this case, the set $\mathbb{S} = \{y | y = \mathbf{E}(x) \wedge x \in \mathbb{Z}_n^*\}$.[12]

6.5 Batch Queries

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 $O(m)$ bits to be sent to the oracle. Batch queries make sense in the first simplification discussed above.

6.6 Decentralizing The Oracle

Distributing the oracle is desirable, since each oracle call involves 3 exponentiations in G_1 (irrespective of the decryption algorithm). It is possible to share the Paillier 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 2.

7 Conclusion

In this paper, we presented a practical implementation of a new cryptographic primitive known as an Oracle Strong Associative One-Way Function (O-SAOWF). As some practical applications of this primitive, we presented a one-round key agreement scheme for dynamic ad-hoc groups based on the protocol due to Rabi and Sherman [6]. The scheme can be extended to group signatures as demonstrated in Section 5.3. In reality, we also demonstrate 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. We envisage several interesting applications of this 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. The curious property of our O-SAOWF 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 (To see this, sample $B \xleftarrow{R} \mathbb{G}$. Then $B' = B^{-1} \star A$). We conclude this section with some open questions.

1 Prove/disprove Conjuncture 4.9.

2 Prove/disprove that $(\text{IDHP}_{(g,G_1)} \Rightarrow \text{CDHP}_{(g,G_1)})$ implies that $(\text{GIP}_{\mathbb{G}} \Rightarrow \text{Oracle } \mathcal{O})$.

3 Remove the oracle from our construction. In other words, exhibit a practical SAOWF construction.

[11]However, the fact that a separation between the two trust models exists may have complexity theoretic significance. To see the separation, consider a game where the oracle always keeps λ secret but may cheat in its computation, its goal being to convince us to accept an invalid group element as valid.

[12]We note that the deterministic variant of the Paillier cryptosystem is not semantically secure and may lead to other subtleties. For instance in this variant we cannot select $t = n + 1$.

Membership size is m	**O-SAOWF**	**GDH** basic [5]	**AGKE** [38]	**GKE**[39]
Number of rounds	1	$m-1$ sequential	2 sequential	2 sequential
Synchronization / ordering needed?	No	Yes	Yes	Yes
Controller needed?	No	No	Yes (initial key distribution)	Yes (group key distribution)
Interaction needed?	No	Yes	Yes	Yes (for synchronization)
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), otherwise $(m-2)k_1$	$(m-1)k_2$	mk_3	k_4
Merge with m_1 users	1 round (total $2(m+m_1)$ messages)	$O(m+m_1)$ rounds (fresh key)	2 rounds (total $m+m_1$ broadcasts)	2 rounds (total $2m_1+m$ messages)
Part with m_1 users	no oracle calls needed if partial keys are cached	$O(m-m_1)$ rounds (fresh key)	2 rounds ($m-m_1$ to $m+m_1$ broadcasts)	1 round (total $m-m_1$ 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 a man-in-the-middle attack	Authentication after 2nd round	Insecure under an active attack [40]
Protection under passive attack	Group Inversion Problem	Diffie-Hellman Problem	Diffie-Hellman Problem	Diffie-Hellman Problem

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

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

\# If public keys are known in advance, the verifiability of the oracle ensures *implicit group key authentication*.

Table 2: Comparison of our group key agreement scheme

Note: We have made a prototype of the construction for cryptanalysis. The implementation is available online at `http://homepage.cs.latrobe.edu.au/asaxena/oracle.htm`. In the implementation, we used the fast variant of the Paillier cryptosystem [29] and a finite field as discussed in Section 6.4.

Acknowledgment

We would like to thank Ronald Rivest, Leonid Levin, Virendra Sule, Pascal Paillier and Chunbo Ma for useful feedback during the preparation of this manuscript.

References

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

[2] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. In *ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory*, pages 385–394, London, UK, 2000. Springer-Verlag.

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

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

[5] 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.

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

[7] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics, American Mathematical Society*, 324:71–90, 2003.

[8] Susan Hohenberger. The cryptographic impact of groups with infeasible inversion. Master's thesis, Massachusetts Institute of Technology, 2003. Supervised by: Ronald L. Rivest.

[9] Alan T. Sherman. *Cryptology and VLSI (a two-part dissertation). I, II, Detecting and exploiting algebraic weaknesses in cryptosystems. Algorithms for placing modules on a custom VLSI chip.* Thesis (Ph.D.), Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, USA, October 1986. Supervised by Ronald Linn Rivest.

[10] Burton S. Kaliksi, Jr., Ronald L. Rivest, and Alan T. Sherman. Is the Data Encryption Standard a group? *Journal of Cryptology*, 1(1):3–36, 1988.

[11] M. Rabi and A. Sherman. Associative one-way functions: A new paradigm for secret-key agreement and digital signatures. Technical Report CS-TR-3183/UMIACS-TR-93-124, Department of Computer Science, University of Maryland Baltimore County, Baltimore, MD, 1993, 1993.

[12] C. M. Homan. Low Ambiguity in Strong, Total, Associative, One-Way Functions. *ArXiv Computer Science e-prints*, October 2000.

[13] Alina Beygelzimer, Lance A. Hemaspaandra, Christopher M. Homan, and Jörg Rothe. One-way functions in worst-case cryptography: algebraic and security properties are on the house. *SIGACT News*, 30(4):25–40, 1999.

[14] Lane A. Hemaspaandra and Jörg Rothe. Creating strong, total, commutative, associative one-way functions from any one-way function in complexity theory. *J. Comput. Syst. Sci.*, 58(3):648–659, 1999.

[15] 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 Mario Coppo, Elena Lodi, and G. Michele Pinna, editors, *ICTCS*, volume 3701 of *Lecture Notes in Computer Science*, pages 265–279. Springer, 2005.

[16] Amitabh Saxena and Ben Soh. A novel method for authenticating mobile agents with one-way signature chaining. In *Proceedings of The 7th International Symposium on Autonomous Decentralized Systems (ISADS 05)*, pages 187–193, China, 2005. IEEE Computer Press.

[17] 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.

[18] Ueli Maurer. Information-theoretic cryptography. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 47–64. Springer-Verlag, August 1999.

[19] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82: Advances in Cryptology*, pages 199–203, New York, 1982. Plenum Press.

[20] László Babai and Endre Szemerédi. On the complexity of matrix group problems I. In *FOCS'1984*, pages 229–240, 1984.

[21] Ronald L. Rivest. On the notion of pseudo-free groups. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 505–521. Springer, 2004.

[22] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532, London, UK, 2001. Springer-Verlag.

[23] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[24] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 354–368, London, UK, 2002. Springer-Verlag.

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

[26] Antoine Joux and Kim Nguyen. Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. *Journal of Cryptology*, 16(4):239–247, 2003.

[27] Neal Koblitz. *A course in number theory and cryptography.* Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[28] 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.

[29] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[30] 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.

[31] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1979.

[32] Dario Catalano, Rosario Gennaro, and Nick Howgrave-Graham. The bit security of Paillier's encryption scheme and its applications. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 229–243, London, UK, 2001. Springer-Verlag.

[33] D. Catalano, R. Gennaro, and N. H. Graham. Paillier's trapdoor function hides up to O(n) bits. *Journal of Cryptology*, 15(4):251–269, 2002.

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

[35] Amos Fiat and Moni Naor. Broadcast encryption. In *CRYPTO '93: Proceedings of the 13th annual international cryptology conference on Advances in cryptology*, pages 480–491, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[36] Hervé Chabanne, Duong Hieu Phan, and David Pointcheval. Public traceability in traitor tracing schemes. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 542–558. Springer, 2005.

[37] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Lecture Notes in Computer Science*, 1666:537–554, 1999.

[38] 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.

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

[40] Junghyun Nam, Seungjoo Kim, 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.

[41] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, 1991.

[42] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO' 86: Proceedings on Advances in cryptology*, pages 186–194, London, UK, 1987. Springer-Verlag.

[43] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[44] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. *Lecture Notes in Computer Science*, 740:390–420, 1993.

APPENDIX

A Soundness Of Verify-In-Group Algorithm

The reader is referred to Section 4.4, algorithm A-4 for the notation used here. First we define the following problem.

Decision Exponent Class Problem [DECP$_{(t,n,g,G_1)}$]: Given $\{t, n, g, G_1\} \subset \mathsf{params}$ and a pair $(x, y) \in G_1 \times \mathbb{Z}^*_{n^2}$, where $x = g^a$ and $y = t^b r^n \bmod n^2$ for unknowns $(a, b, r) \in \mathbb{Z}_n \times \mathbb{Z}_n \times \mathbb{Z}^*_n$, output 1 if $[a \equiv b \pmod p \veebar a \equiv b \pmod q]$, otherwise output 0.

The following theorem shows that the Verify-In-Group algorithm is sound if the problems DECP$_{(t,n,g,G_1)}$ and CDHP$_{(g,G_1)}$ are intractable.

Theorem A.1. *If the decision exponent class problem and the computational Diffie-Hellman problem are hard then the* Verify-In-Group *algorithm is sound.*

Proof. The input to the Verify-In-Group algorithm is $(x, y) \in G_1 \times \mathbb{Z}^*_{n^2}$. We must show that if the algorithm outputs 1 then $(x, y) \in \mathbb{G}$. Let $x = g^a$ and $y = t^b r^n \bmod n^2$ for unknowns $(a, b, r) \in \mathbb{Z}_n \times \mathbb{Z}_n \times \mathbb{Z}^*_n$. The transformation of (x, y) to (x_1, y_1) and (x_2, y_2) in step 2 of the algorithm can be denoted by the mapping

$$\begin{array}{rcl} f_1 : \mathbb{Z}_n \times \mathbb{Z}_n \times \mathbb{Z}^*_n & \mapsto & G_1 \times \mathbb{Z}^*_{n^2} \\ (u, v, w) & \mapsto & (g^{au+v}, t^{bu+v} r^{un} w^n \bmod n^2) \end{array}$$

Consider the cases when the algorithm outputs 1.

Case 1. $[a \equiv b \pmod p \wedge a \equiv b \pmod q]$: In this case $a = b$ and so $(x, y) \in \mathbb{G}$. Therefore, $f_1(u, v, w) \in \mathbb{G}\ \forall\ (u, v, w) \in domain(f_1)$. In this case, the output of Verify-In-Group algorithm is consistent with its requirements.

Case 2. $[a \not\equiv b \pmod p \wedge a \not\equiv b \pmod q]$: It is not hard to prove that the mapping f_1 is a bijection in this case. Since both sides of f_1 have the same number of elements $n^2\phi(n)$, it is enough to prove that f_1 is invertible with respect to every element in $G_1 \times \mathbb{Z}^*_{n^2}$. Let $(g^{a_1}, t^{b_1}{r_1}^n \bmod n^2) \in G_1 \times \mathbb{Z}^*_{n^2}$ be an element of the right side of f_1. If a preimage (u_1, v_1, w_1) of f_1 exists for this element, then we must have

$$\left.\begin{array}{rcll} a_1 & \equiv & au_1 + v_1 & \pmod n \\ b_1 & \equiv & bu_1 + v_1 & \pmod n \\ r_1 & \equiv & r^{u_1}w_1 & \pmod n \end{array}\right\} \tag{13}$$

Clearly equation 13 has a unique solution in (u_1, v_1, w_1) for all (a_1, b_1, r_1) if and only if $(a-b) \in \mathbb{Z}^*_n$. In other words, if and only if $\gcd(a-b, n) = 1$. Note that $\gcd(a-b, n) = 1$ is another way of saying that $[a \not\equiv b \pmod p \wedge a \not\equiv b \pmod q]$.

Since f_1 is a bijection, the distributions $\{(x_1, y_1)\}$ and $\{(x_2, y_2)\}$ are identical to a random distribution in $G_1 \times \mathbb{Z}^*_{n^2}$. If the oracle $\mathcal{O}^*$ can make the algorithm output 1 then we can use $\mathcal{O}^*$ to solve $\text{CDHP}_{(g,G_1)}$ (see Section 3.1) as follows:

1 Input is $g, g^{\sigma_1}, g^{\sigma_2}$ and our goal is to output $g^{\sigma_1\sigma_2}$.

2 Generate $y_1, y_2 \xleftarrow{R} \mathbb{Z}^*_{n^2}$

3 Set $x_1 \leftarrow g^{\sigma_1}$ and $x_2 \leftarrow g^{\sigma_2}$

4 Give $(x_1, y_1), (x_2, y_2)$ as input to oracle $\mathcal{O}^*$ in step 3 of the algorithm instead of the real values.

Since the forged and real distributions of $\{(x_1, y_1)\}$ and $\{(x_2, y_2)\}$ are identical, the oracle $\mathcal{O}^*$ cannot distinguish between the forged and real inputs. Accordingly it will reply with (x', y') such that the algorithm outputs 1 in step 4. In this case x' is the required solution to the $\text{CDHP}_{(g,G_1)}$ instance.

Case 3. $[a \equiv b \pmod p \wedge a \not\equiv b \pmod q]$: (or $\gcd(a - b, n) > 1$ and $a \neq b$)

The probability of a randomly picked pair $(x, y) \in G_1 \times \mathbb{Z}^*_{n^2}$ such that $\gcd(a - b, n) > 1$ and $a \neq b$ is $\frac{p+q-2}{pq}$ which can be neglected for large p, q. On the other hand, if the adversary $(\mathcal{O}^*)$ knows in advance that $\gcd(a - b, n) > 1$ but does not know both of $\{a, b\}$, then the adversary knows that the distribution of the image

$$f_1(u_1, v_1, w_1) = (g^{a_1},\ t^{b_1}{r_1}^n \bmod n^2)$$

always satisfies $a_1 \equiv b_1 \pmod p$. In this case, our security relies on the adversary's inability to distinguish elements of this distribution from randomly chosen elements of $G_1 \times \mathbb{Z}^*_{n^2}$ assuming the hardness of $\text{DECP}_{(t,n,g,G_1)}$. Under this assumption, we can use the adversary $\mathcal{O}^*$ to solve $\text{CDHP}_{(g,G_1)}$ as in the previous case. The case of $[a \not\equiv b \pmod p \wedge a \equiv b \pmod q]$ is handled similarly.

Thus, we have proved that the algorithm is sound under the assumption that the problems $\text{DECP}_{(t,n,g,G_1)}$ and $\text{CDHP}_{(g,G_1)}$ are intractable. $\square$

B Proof Of Knowledge Of Private Keys Of Section 5.1

Recall from Section 5.1 that the private key of user i is σ_{X_i}, the sampling information for some $X_i \in \mathbb{G}$, while the public key is $Y_i = X_i \star P$. In this section we give two constructions of zero-knowledge proofs for this setting. The first construction enables user i to claim knowledge of the value $X_i \in \mathbb{G}$ such that $Y_i = X_i \star P$ (without saying anything about σ_{X_i}). The second construction enables user i to claim knowledge of not only X_i but also σ_{X_i} such that $Y_i = X_i \star P$ and σ_{X_i} is the sampling information of X_i. The first construction is useful when the sampling information of some group element is not known, yet a proof of knowledge of that group element must be given (see item 3 in Section 5.2.2).

For conveineence we will drop the subscript i in what follows and consider the public key as $Y = X \star P$ with σ_X being the sampling information of X. The zero-knowledge proofs given below are similar to the proofs of graph isomorphism [41] and quadratic residuosity [42, 43].

B.1 Zero-Knowledge Proof Of Knowledge Of Group Element

In this section we give a zero-knowledge proof of knowledge that can be used by a prover $\mathcal{P}$ to prove to a verifier $\mathcal{V}$, the knowledge of a value X such that $Y = X \star P$ for some given pair (P, Y). The proof of knowledge is as follows.

PROTOCOL $(\mathcal{P}, \mathcal{V})$

Common input: $(Y, P) \in \mathbb{G}^2$.
$\mathcal{P}$'s auxiliary input: $X \in \mathbb{G}$ such that $Y{=}X \star P$.
$\mathcal{V}$'s auxiliary input: none
Claim: $\mathcal{P}$ claims to know X such that $Y{=}X \star P$.

1 $\mathcal{P}$ samples $U_0 \stackrel{R}{\leftarrow} \mathbb{G}$ using the method described earlier and computes $U_1 = X \star {U_0}^{-1} \in \mathbb{G}$. Thus $U_0 \star U_1 = X$ (essentially $\mathcal{P}$ will prove knowledge of (U_0, U_1) such that the $X = U_0 \star U_1$). $\mathcal{P}$ then computes $W_j = U_j \star P$ for $j \in \{0, 1\}$ and sends the pair (W_0, W_1) of commitments to $\mathcal{V}$.

2 $\mathcal{V}$ checks if the following equality holds

$$W_0 \star W_1 \stackrel{?}{=} Y \star P \tag{14}$$

and terminates the protocol if this check fails. Otherwise, $\mathcal{V}$ picks up a random bit $b \stackrel{R}{\leftarrow} \{0, 1\}$ and sends b as its challenge to $\mathcal{P}$.

3 On receiving $b \in \{0, 1\}$, $\mathcal{P}$ replies with U_b.
On receiving U_b, $\mathcal{V}$ checks if the following equality holds:

$$W_b = U_b \star P \tag{15}$$

If the equality does not hold then $\mathcal{P}$ outputs *REJECT*. Otherwise $\mathcal{P}$ outputs *ACCEPT*.

The above protocol $(\mathcal{P}, \mathcal{V})$ is a *atomic* zero-knowledge proof of knowledge of X with *knowledge error* 1/2 using the definitions of [44]. The protocol is (sequentially) repeated k times to reduce this knowledge error to $1/2^k$. The next theorem shows that the above protocol is indeed a zero-knowledge proof of knowledge of X.

Theorem B.1. *The above protocol* $(\mathcal{P}, \mathcal{V})$ *is a zero-knowledge proof of knowledge of* X.

Proof. First note that the protocol is *complete* (i.e. if both parties are honest, the verifier always accepts). To finish the prove, we must show that the protocol is a proof of knowledge (i.e. has an *extractor* [44]) and is zero-knowledge (i.e. has a *simulator* [41]).

Proof Of Knowledge: To prove that protocol $(\mathcal{P}, \mathcal{V})$ is a proof of knowledge, we must exhibit an extractor for X. Our extractor is similar to that of [43]. Construction of this extractor is trivial by rewinding the prover $\mathcal{P}$ and giving it different challenge bits b on the same commitment (W_0, W_1) and obtaining both U_0 and U_1. We already know that $W_0 \star W_1 = Y \star P = X \star P^2$ (equation 14) and $U_b = W_b \star P^{-1}$ (equation 15). Therefore, once we obtain (U_0, U_1) we can compute $X = U_0 \star U_1$.

Zero-Knowledge: We will demonstrate the zero-knowledge property of protocol $(\mathcal{P}, \mathcal{V})$ using an idea similar to the one used in [43]. We assume that the simulator knows in advance what the challenge bit b is going to be. The simulator runs as follows: It samples $U_b \stackrel{R}{\leftarrow} \mathbb{G}$. Then it computes $W_b = U_b \star P$

and $W_{1-b} = {U_b}^{-1} \star Y$. The commitment of step 1 is (W_0, W_1). It is easily checked that $W_0 \star W_1 = W_b \star W_{1-b} = U_b \star P \star {U_b}^{-1} \star Y = Y \star P$ as needed in equation 14. Finally after the challenge bit b is received, the simulator simply reveals U_b. We have $W_b = U_b \star P$ as required in equation 15. Clearly, the transcript of the simulation is distributed identically to the transcript of the real interaction. □

B.2 Zero-Knowledge Proof Of Knowledge Of Sampling Information

In this section we give a zero-knowledge proof of knowledge that can be used by a prover $\mathcal{P}$ to prove to a verifier $\mathcal{V}$, the knowledge of a value σ_X such that $Y = X \star P$ for some given pair (P, Y) and σ_X is the sampling information of X. The proof of knowledge is as follows.

PROTOCOL $(\mathcal{P}, \mathcal{V})$

Common input: $(Y, P) \in \mathbb{G}^2$.
$\mathcal{P}$'s auxiliary input: $\sigma_X \in \mathbb{Z}_n^*$ such that σ_X is the sampling information of $X \in \mathbb{G}$ and $Y = X \star P$.
$\mathcal{V}$'s auxiliary input: none
Claim: $\mathcal{P}$ claims to know σ_X, the sampling information of $X \in \mathbb{G}$ such that $Y = X \star P$.

1 $\mathcal{P}$ generates $(U, \sigma_U) \xleftarrow{R} \mathsf{Sample}(\mathsf{params}) \in \mathbb{G} \times \mathbb{Z}_n^*$ and computes $W = U \star Y \in \mathbb{G}$. The commitment W is sent to $\mathcal{V}$.

2 $\mathcal{V}$ checks that $W \in \mathbb{G}$ using the $\mathsf{Verify\text{-}In\text{-}Group}$ algorithm of Section 4.4 and terminates the protocol if $W \notin \mathbb{G}$. Otherwise, $\mathcal{V}$ picks up a random bit $b \xleftarrow{R} \{0, 1\}$ and sends b as its challenge to $\mathcal{P}$.

3 On receiving $b \in \{0, 1\}$, $\mathcal{P}$ replies with $\sigma_Z = \sigma_U \cdot {\sigma_X}^b \bmod n \in \mathbb{Z}_n^*$.
On receiving σ_Z, $\mathcal{V}$ computes $Z \in \mathbb{G}$ such that σ_Z is the sampling information of Z and checks if the following equality holds:
$$W = Z \star Y^{1-b} \star P^b \tag{16}$$
If the equality does not hold then $\mathcal{P}$ outputs *REJECT*. Otherwise $\mathcal{P}$ outputs *ACCEPT*.

Like the previous protocol the above protocol $(\mathcal{P}, \mathcal{V})$ is a *atomic* zero-knowledge proof of knowledge of X with *knowledge error* $1/2$ using the definitions of [44]. The protocol is (sequentially) repeated k times to reduce this knowledge error to $1/2^k$. The next theorem shows that the above protocol is indeed a zero-knowledge proof of knowledge of σ_X.

Theorem B.2. *The above protocol $(\mathcal{P}, \mathcal{V})$ is a zero-knowledge proof of knowledge of σ_X.*

Proof. First note that the protocol is *complete* (i.e. if both parties are honest, the verifier always accepts). To finish the prove, we must show that the protocol is a proof of knowledge (has an extractor) and is zero-knowledge (has a simulator).

Proof Of Knowledge: To prove that protocol $(\mathcal{P}, \mathcal{V})$ is a proof of knowledge, we must exhibit an extractor for σ_X. Construction of this extractor is trivial by rewinding the prover $\mathcal{P}$ and giving it different challenge bits b on the same commitment W to obtain both $\sigma_Z = \sigma_U$ for $b = 0$ and $\sigma_{Z'} = \sigma_U \cdot \sigma_X$ for $b = 1$. Clearly $\sigma_X = \sigma_{Z'}/\sigma_Z$.

Zero-Knowledge: Assume that the simulator knows in advance what the challenge bit b is going to be. The simulator runs as follows: It generates $(Z, \sigma_Z) \xleftarrow{R} \mathsf{Sample}(\mathsf{params}) \in \mathbb{G} \times \mathbb{Z}_n^*$ and computes $W = Z \star Y^{1-b} \star P^b \in \mathbb{G}$ as the commitment. On receiving the challenge bit b, it replies with σ_Z. It is clear that equation 16 will be satisfied irrespective of the choice of bit b. □

C Identity Based Encryption Using O-SAOWFs

In this section we give (without a security proof) an Identity Based Encryption (IBE) scheme as another application of our O-SAOWFs. We refer the reader to [23] for the definitions of an IBE scheme and to Section 4.4 for the notation used here. In summary, our IBE scheme has four PPT algorithms Setup-IBE, KeyGen, ID-Encrypt and ID-Decrypt. The definition of "PPT" has the usual caveat; oracles are considered as algorithms.

1 The $\mathsf{Setup\text{-}IBE}$ algorithm takes as input some security parameter. It outputs the IBE system parameters par and the IBE master key $\mathsf{m\text{-}key}$.

2 The KeyGen algorithm takes as input the value par, $\mathsf{m\text{-}key}$ and a random string i. It outputs the private key $\mathsf{prv\text{-}key}_i$ corresponding to the string i.

3 The $\mathsf{ID\text{-}Encrypt}$ algorithm takes as input par, a random message m and a random string i. It outputs a ciphertext c.

4 The $\mathsf{ID\text{-}Decrypt}$ algorithm takes as input par, a private key $\mathsf{prv\text{-}key}_i$ (corresponding to some string i) and ciphertext c. It outputs a message m.

The $\mathsf{ID\text{-}Encrypt}$ and $\mathsf{ID\text{-}Decrypt}$ algorithms satisfy the standard consistency constraint:

$$\forall m\ \forall i\ \mathsf{ID\text{-}Decrypt}(\mathsf{par}, \mathsf{ID\text{-}Encrypt}(\mathsf{par}, m, i), \mathsf{KeyGen}(\mathsf{par}, \mathsf{m\text{-}key}, i)) = m$$

In an IBE scheme, the master key $\mathsf{m\text{-}key}$ is known only to a trusted authority known as the Key Generating Center (KGC) that is responsible for distributing private keys. In our construction although the oracle $\mathcal{O}$ is required for computation, it *need not* be the Key Generating Center (KGC). The four algorithms are described below.

1 $\mathsf{Setup\text{-}IBE}$: Set $(X, \sigma_X), (Y, \sigma_Y) \overset{R}{\leftarrow} \mathsf{Sample}(\mathsf{params})$ and set $Z \leftarrow \mathcal{T}(\sigma_X, 1, Y) = X \star Y$. Finally set $\mathsf{par} \leftarrow (Y, Z) \in \mathbb{G}^2$; $\mathsf{m\text{-}key} \leftarrow (\sigma_X, \sigma_Y) \in {\mathbb{Z}_n^*}^2$ and output $(\mathsf{par}, \mathsf{m\text{-}key})$.

2 KeyGen: Let $i \in \mathbb{N}$ be the input string. Set $\mathsf{prv\text{-}key}_i \leftarrow \mathcal{T}(\sigma_X, -i, Y) = X^{-i} \star Y \in \mathbb{G}$ and output $\mathsf{prv\text{-}key}$.

3 $\mathsf{ID\text{-}Encrypt}$: Our message space is $\{0,1\}^l$ where $l < \log_2(n)$ and we require a cryptographic hash function $\mathcal{H} : \mathbb{G} \mapsto \{0,1\}^l$. To encrypt a message $m \in \{0,1\}^l$ using input string $i \in \mathbb{N}$, first generate random $(R, \sigma_R) \overset{R}{\leftarrow} \mathsf{Sample}(\mathsf{params})$. Then compute

$$c_1 = m \oplus \mathcal{H}(\mathcal{T}(\sigma_R, 1, \mathcal{E}(Y, i+1))) = m \oplus \mathcal{H}(Y^{i+1} \star R)$$

$$C_2 = \mathcal{T}(\sigma_R, 1, \mathcal{E}(Z, i)) = Z^i \star R = X^i \star Y^i \star R$$

The ciphertext is (c_1, C_2).

Both c_1 and C_2 can be directly computed if Y^{i+1} and Z^i are precomputed.

4 $\mathsf{ID\text{-}Decrypt}$: To decrypt arbitrary ciphertext (c_1, C_2) compute

$$m = c_1 \oplus \mathcal{H}(\widehat{\mathcal{O}}(C_2, \mathsf{prv\text{-}key}_i)) = c_1 \oplus \mathcal{H}(C_2 \star X^{-i} \star Y)$$

Decryption is correct, because for a legitimate ciphertext:

$$C_2 \star X^{-i} \star Y = (X^i \star Y^i \star R) \star (X^{-i} \star Y) = Y^{i+1} \star R$$

The security of the above construction will be discussed in the full version of this paper.