J. Math. Crypt. **3** (2009), 89–132
DOI 10.1515 / JMC.2009.005

A cryptographic primitive based on hidden-order groups

Amitabh Saxena and Ben Soh

Communicated by Igor Shparlinski

Abstract. Let G_1 be a cyclic multiplicative group of order n. It is known that the computational Diffie–Hellman (CDH) problem is random self-reducible in G_1 if $\phi(n)$ is known. That is, given $g, g^x \in G_1$ for some generator g and oracle access to a "Diffie-Hellman Problem solver" for g, it is possible to compute $g^{1/x} \in G_1$ in polynomial time (with which we can then solve the CDH problem w.r.t. any other generator). On the other hand, it is not clear if such a reduction exists when $\phi(n)$ is unknown. We exploit this "gap" to construct a novel cryptographic primitive, which we call an *Oracle-based Group with Infeasible Inversion* (O-GII). O-GIIs have applications in multiparty protocols. We demonstrate this by presenting a novel multi-party key agreement protocol that does not require interaction between the parties. Instead, the protocol requires each party to query a remote stateless device. Our method relies on the observation that it is considerably more expensive to interact with every party connected via an unreliable network, than it is to query one of several identical stateless devices, some of which may be located in a more reliable sub-network.

Keywords. Groups with infeasible inversion, non-interactive key agreement, multiparty computation, broadcast encryption.

AMS classification. 94A60.

1 Introduction

The problem of efficient key agreement in ad-hoc groups is a challenging one, 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 an initial key distribution phase. Two notable examples of one-round key agreement protocols are the classic two-party Diffie–Hellman key exchange [15] and the Joux tripartite key exchange using bilinear maps [26]. However, to date, the construction of a generalized one-round n-party key agreement protocol has remained a challenging and open problem.

OUR CONTRIBUTION. In this paper, we present a practical 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 stateless third-party computing device is needed. The advantages of our approach compared to conventional hardware-based approaches are:

This research was supported by funding from the Australian Government under the APA and ICT awards.

(1) *Location independence.* The device may be located remotely.

(2) *Transparency.* We do not require any registration process for new members.

(3) *Replicability.* The device may be replicated arbitrarily many times and may even be locally implemented in a tamper-proof manner. In the latter case, we have zero communication cost.

(4) *Statelessness.* The device needs a small amount of computing power and no memory. In other words, the device is *stateless* – it does not maintain information about the users or its other copies.

   a) This precludes the possibility of secure channels existing between the device and its users, a common assumption in many third-party based approaches. Our device is completely oblivious to the identity and number of users.

   b) Due to the statelessness and replicability of the device, our approach provides fault tolerance. In a large group of geographically disparate users, several independent copies of this device could be operated such that members are not tied to any particular one. Although members located geographically close would share a copy located in their neighborhood, they could switch to a different copy in case of network failures.

(5) *Overhead.* For a set of n users, each user sends one $O(n)$ sized message to the device. The response size is $O(1)$. However, the results of earlier computations can be cached and stored at insecure locations (for join/leave/merge operations), reducing the sent message size to $O(1)$. Furthermore, the size and number of transmitted messages can be greatly reduced by arranging members in data structures such as trees. All information transmitted/stored is considered public.

Our approach can be summarized using Figure 1.

MOTIVATION. We refer the reader to [41, 57] 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. The main problems with distributed and centralized approaches is that (1) they require group controllers that need to main a large amount of internal state information (i.e., the session and pairwise long-term keys) for the users it is managing; and (2) the group controller forms a central point of failure for its managed members. On the other hand, contributory key agreement approaches (which do not rely on group controllers) require several rounds of interaction between the members. Our motivation was to design a protocol that does not suffer from any of these weaknesses.

RELATED WORK. Our basic idea arises due to the paper of Rabi and Sherman [39], 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. SAOWFs with a group structure are called *Groups with Infeasible Inversion (GIIs)*. 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* [9]. However, as of now, no practical construction of either primitive is known. In

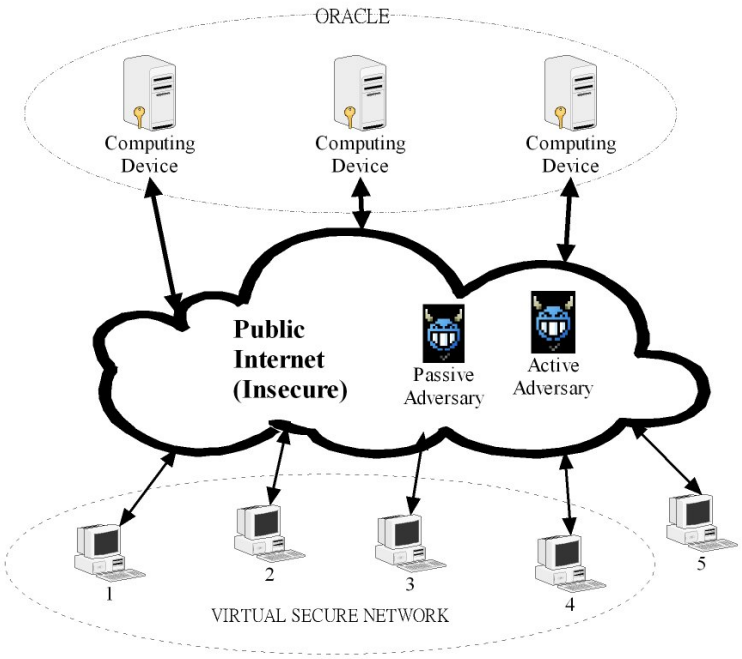

Figure 1. In our model, secure group communication is facilitated by the Oracle. Assuming that public keys are known in advance, users can use this Oracle to compute a shared secret key independently of the other users such that no (active or passive) adversary has the ability to compute this key. Essentially the oracle is used as a "verifiable computing device" and the adversary as the communication medium.

this paper, we extend the work of Rabi and Sherman and give a practical construction of a GII under a restricted model of computation, namely *black-box computation*. Our construction also provides an example of a *Trapdoor Group with Infeasible Inversion* (TGII) discussed in [24].

ORGANIZATION OF THIS PAPER. The rest of the paper is organized as follows. In Section 2, we give some background and notation. We define GIIs in Section 2.2 and extend this definition to include black-box computation in Section 3.1. Our construction is presented in Section 5 and some applications are given in Section 6. Finally, we discuss implementation issues in Section 7.

2 Preliminaries

Around 1984, Rivest and Sherman suggested a one-round fully contributory key agreement protocol for ad-hoc groups using a class of cryptographic primitives that they called *Strong Associative One-Way Functions (SAOWFs)* [51, 28]. Later in 1993, Rabi

and Sherman suggested the use of SAOWFs in digital signatures [38]. We give the definitions from [38] with some simplifications.

2.1 Strong associative one-way functions

Definition 2.1. A binary function $f : \Sigma^* \times \Sigma^* \mapsto \Sigma^*$ is a *Strong Associative One-Way Function* (SAOWF) if the following properties are satisfied.[1]

(1) *Associativity*: $\forall X, Y, Z \in \Sigma^* : f(f(X,Y),Z) = f(X,f(Y,Z))$.

(2) *Computability*: $\forall X, Y \in \Sigma^* : f(X,Y)$ is efficiently computable.

(3) *Strong Non-Invertibility*: It is infeasible to invert f w.r.t. any input. That is,

$$\forall \text{PPT} \mathcal{A}\ \exists \ell' \in \mathbb{N}\ \forall \ell > \ell' : \Pr\left[X, Y \xleftarrow{R} \Sigma^{\ell} : f(X, \mathcal{A}(X, f(X,Y)) = f(X,Y)\right] \leq \mathit{negl}(\ell).$$

SAOWFs have many interesting applications such as one-round multiparty key agreement and homomorphic signatures [39, 49].

MULTIPARTY KEY AGREEMENT USING SAOWFS. As early as 1984, Rivest and Sherman noted that SAOWFs can be used for non-interactive multiparty key agreement (however, the first published reference we found is from 1993 [38, 39]). Their protocol, described below, requires f to commute (i.e., $\forall X, Y \in \Sigma^* : f(X,Y) = f(Y,X)$).

Initial Key Distribution. All users agree on a common parameter $P \xleftarrow{R} \Sigma^*$ and each user i $(1 \leq i \leq n)$ generates random $X_i \xleftarrow{R} \Sigma^*$ and computes $Y_i = f(X_i, P)$. The values X_i and Y_i are the private and public keys respectively.

Key Agreement. To agree on a key, each user i $(1 \leq i \leq n)$ computes independently

$$K_i = f(X_i, f(Y_1, f(Y_2, \ldots, f(Y_{i-1}, f(Y_{i+1}, f(Y_{i+2}, \ldots, f(Y_{n-1}, Y_n)\ldots)))\ldots))).$$

Due to the associativity and commutativity of f, each user will compute the same value K_i, which will be used as the secret key. The security lies in the apparent difficulty of inverting f with respect to P. A security analysis of this protocol in Yao's model of information theory [56, 55] is given in [38]. Although the above protocol fails if the commutativity requirement is relaxed, a minor modification allows us to use it with non-commutative SAOWFs [48].

SUBSEQUENT WORK. Subsequently, Rabi and Sherman [39] gave an existence proof of complexity-theoretic[2] SAOWFs under the $P \neq NP$ hypothesis. Other authors studied complexity-theoretic SAOWFs with respect to different properties such as low ambiguity, strong invertibility, totality and commutativity [25, 4, 22]. Finally, in [23], Hemaspaandra, Rothe and Saxena gave a complete characterization of complexity-theoretic SAOWFs.

[1] Since we later restrict the algebraic structure of SAOWFs to finite abelian groups, we avoid mentioning properties such as *honesty*, *non-commutativity* and *totality* used in [22, 23] for describing SAOWFs.

[2] In complexity-theoretic SAOWFs, the strong non-invertibility property is satisfied in the worst case but not necessarily in the average case.

2.2 Groups with infeasible inversion

In this work, we restrict the structure of SAOWFs to finite abelian groups. The resulting object is called a *Group with Infeasible Inversion (GII)* [24].

Definition 2.2. We say that a finite abelian group $(\mathbb{G}, \star, I)$ with identity element I is a *Group with Infeasible Inversion (GII)* if the following four properties are satisfied:

(1) *Efficient and unique representation:* Elements of $\mathbb{G}$ are efficiently and uniquely representable (i.e., in $O(\log |\mathbb{G}|)$ bits).

(2) *Samplability:* It is easy to sample pairs $(P, P^{-1}) \stackrel{R}{\leftarrow} \mathbb{G}^2$ such that $I = P \star P^{-1}$.

(3) *Efficient Composability:* Given any pair $(X, Y) \in \mathbb{G}^2$, it is possible to compute $X \star Y$ efficiently.

(4) *Strong Non-invertibility:* Given $P \in_R \mathbb{G}$, it is infeasible to compute P^{-1} efficiently. Formally, for all PPT algorithms $\mathcal{A}$, and for all sufficiently large $|\mathbb{G}|$,

$$\Pr[P \stackrel{R}{\leftarrow} \mathbb{G} : \mathcal{A}(P) \star P = I] \leq negl(|\mathbb{G}|).$$

The following result is well known.

GIIs MUST HAVE HIDDEN ORDER. This is stated as Lemma 2.3.

Lemma 2.3 ([50, 45, 24]). *If $(\mathbb{G}, \star, I)$ is a GII then $|\mathbb{G}|$ must be unknown.*

Proof. Let $|\mathbb{G}| = m$. We will show that Properties 3 and 4 of Def. 2.2 cannot simultaneously hold if m is known. Let $P \in \mathbb{G}$. To compute P^{-1}, it suffices to compute P^{m-1}, which can be done efficiently using the *Repeat and Square algorithm* [30] for generic groups if Property 3 holds. □

2.2.1 Gap-CDH groups

A possible construction of GIIs is based on the hardness of the discrete logarithm problem in certain groups. Our idea is based on this approach.

Let G_1 be a cyclic multiplicative group of order n and let g generator of G_1. Define the following three problems:

(1) **Computational Diffie–Hellman ($\mathbf{CDH_g}$) Problem:** *Input:* $(g^x, g^y) \in {G_1}^2$ (for some $x, y \in \mathbb{Z}_n$). *Output:* $g^{xy} \in G_1$.

(2) **Inverse Diffie–Hellman ($\mathbf{IDH_g}$) Problem:** *Input:* $g^x \in G_1$ (for some $x \in \mathbb{Z}_n^*$). *Output:* $g^{1/x} \in G_1$.

(3) **Discrete Logarithm ($\mathbf{DL_g}$) Problem:** *Input:* $g^x \in G_1$ (for some $x \in \mathbb{Z}_n$). *Output:* x.

We say that G_1 is a Gap-CDH group if there is a "gap" between IDH_g and the CDH_g problems; that is, if the CDH_g problem is *easy* but the IDH_g problem is *hard*. We define this formally below.

Let $\mathcal{G}$ be an algorithm taking as input a securty parameter $\tau \in \{1\}^*$ and outputting a tuple (G_1, g, n) such that n is a $|\tau|$ bit integer, G_1 is (the description of) a cyclic group of order n and g is a generator of G_1. We say $\mathcal{G}$ is a Gap-CDH generator if there exists a negligible function $\nu : \mathbb{N} \mapsto [0, 1]$ such that the following two conditions hold:

Condition 1. The CDH_g problem is easy. That is, $\exists$PPT $\mathcal{A}$ $\forall \tau \in \mathbb{N}$:

$$\Pr\left[\mathcal{A}(G_1, g, n, h, g^x) = h^x \;\middle|\; \begin{array}{c} (G_1, g, n) \xleftarrow{R} \mathcal{G}(\tau), \\ h \xleftarrow{R} G_1, x \xleftarrow{R} \mathbb{Z}_n \end{array} \right] = 1. \tag{2.1}$$

Condition 2. The IDH_g problem is hard. That is, $\forall$PPT $\mathcal{A}$ $\exists \tau' \in \mathbb{N}$ $\forall \tau > \tau'$:

$$\Pr\left[\mathcal{A}(G_1, g, n, g^x) = g^{\frac{1}{x}} \;\middle|\; \begin{array}{c} (G_1, g, n) \xleftarrow{R} \mathcal{G}(\tau), \\ x \xleftarrow{R} \mathbb{Z}_n^* \end{array} \right] \leq \nu(\tau). \tag{2.2}$$

Condition 2 is equivalent to the following two:

Condition 2a. The DL_g problem is hard. That is, $\forall$PPT $\mathcal{A}$ $\exists \tau' \in \mathbb{N}$ $\forall \tau > \tau'$:

$$\Pr\left[\mathcal{A}(G_1, g, n, g^x) = x \;\middle|\; \begin{array}{c} (G_1, g, n) \xleftarrow{R} \mathcal{G}(\tau), \\ x \xleftarrow{R} \mathbb{Z}_n \end{array} \right] \leq \nu(\tau). \tag{2.3}$$

Condition 2b. The CDH problem is not random-self reducible in G_1. In particular, the CDH_h problem for a random generator $h \xleftarrow{R} G_1$ (with $h \neq g$) is hard and not reducible to the CDH_g problem. That is, $\forall$PPT $\mathcal{A}$ $\exists \tau' \in \mathbb{N}$ $\forall \tau > \tau'$:

$$\Pr\left[\mathcal{A}(G_1, g, n, h, h^x, i) = i^x \;\middle|\; \begin{array}{c} (G_1, g, n) \xleftarrow{R} \mathcal{G}(\tau), \\ (h, i) \xleftarrow{R} {G_1}^2, x \xleftarrow{R} \mathbb{Z}_n \end{array} \right] \leq \nu(\tau). \tag{2.4}$$

Remark 2.4. The restriction of Condition 2b, Eq. 2.4 (i.e., the CDH_g problem is easy but CDH_h problem is hard for some randomly selected element $h \in G_1$) is necessary to ensure that the IDH_g problem is hard, since otherwise, $\text{IDH}_g(h) = \text{CDH}_h(g, g)$.

GIIs FROM GAP-CDH GROUPS. Let $\mathcal{G}$ be a Gap-CDH generator algorithm. Let $(G_1, g, n) \xleftarrow{R} \mathcal{G}(\tau)$. Define the set $\mathbb{G} \subsetneq G_1$ as $\mathbb{G} = \{h | \exists x \in \mathbb{Z}_n^* : h = g^x\}$. Thus, $|\mathbb{G}| = \phi(n)$. Define an operation $\star$ in $\mathbb{G}$ equivalent to solving the CDH_g problem in $\mathbb{G}$. That is, for $g^x, g^y \in \mathbb{G}$, define $g^x \star g^y = g^{xy} \in \mathbb{G}$. Clearly, $(\mathbb{G}, \star)$ has the structure of an abelian group. Assuming that the CDH_g problem is easy, it is always possible to compute the group operation in $(\mathbb{G}, \star)$. To turn $(\mathbb{G}, \star)$ into a GII we simply keep the value of $\phi(n)$ hidden by keeping the factorization of n hidden.

Thus, a possible construction of a GII arises from Gap-CDH groups of composite order with hidden factorization. We hypothesize the existence of such groups and as a starting point present a practical black-box construction of a Gap-CDH group.

KNOWN RESULTS ABOUT GAP-CDH GROUPS. It is noteworthy that a result of Maurer (1994) states that for any *prime* n, if a smooth elliptic curve of order n is known, then the CDH_g and DL_g problems in all groups of order n are equivalent [32]. Another related result states that in a group of order n, the DL_g problem can be polytime reduced to the CDH_g problem if the factors of n are "small" [34]. However, it is shown in [33, Theorem 7] that the DL_g problem is *not* computationally equivalent to the CDH_g problem in generic groups if the order of the group contains large prime factors. Therefore, the above results leave open the possibility of Gap-CDH groups with composite order n having large prime factors – exactly the ones we desire.

CONNECTION WITH THE PAIRING INVERSION PROBLEM. One method to construct Gap-CDH groups is to exhibit an efficiently computable non-degenerate symmetric bilinear map $\hat{e} : G_1 \times G_1 \mapsto G_2$ [8] such that the discrete log problem in G_1 and G_2 is hard but the *pairing inversion problem* w.r.t. some special generator $g \in G_1$ is easy. The pairing inversion problem is the problem of inverting the bilinear map w.r.t. some given element of G_1 [14]. See [46, 20] for a recent study of this problem.

3 Black-Box GII constructions

Loosely speaking, a black-box GII is defined by extending the definition of "efficient computation" in Property 3 of Def. 2.2 to include *efficient black-box computation*.

In our black-box model although the group $(\mathbb{G}, \star, I)$ is easily samplable, 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. However, for a black-box construction to have any practical significance it must support (a) verifiable and (b) private computation as elaborated next.

PV-ORACLES. To justify the use of a black-box (a.k.a. an oracle) as one-way function in a cryptographic protocol, we must provide the same guarantees that a real function provides. Specifically, a real function is private and verifiable. We define similar properties for oracle-computed functions. We restrict ourselves to an oracle that computes a binary commutative function.

Definition 3.1 (Verifiable Oracle). Let f be a binary commutative function computed by an oracle. We say that the oracle allows verifiable computation if for all $X, Y \in$ *domain*(f) and all $Z \in$ *image*(f), there exists a PPT verification algorithm Verify such that $\mathsf{Verify}(X, Y, Z) = 1$ iff $C = f(A, B)$. An oracle allowing verifiable computation is called a *Verifiable Oracle (V-Oracle)*.

Definition 3.2 (Private and Verifiable Oracle). Let f be a binary commutative function computed by a V-Oracle. We say that the V-Oracle allows *private computation* if there exist two PPT algorithms Blind and $\mathsf{Unblind}$ satisfying correctness and perfect privacy defined below.

Blind is a randomized algorithm and takes as input an element $X \in \mathit{domain}(f)$ and outputs a tuple (X', σ), where $X' \in \mathit{domain}(f)$ and σ is some auxiliary information (called the *Unblinding Value*).

Unblind takes as input a tuple (Z', σ), where $Z' \in \mathit{image}(f)$ and σ is an unblinding value. It outputs a value $Z \in \mathit{image}(f)$.

(1) We say that (Blind, Unblind) satisfy correctness if $\forall X, Y \in \mathit{domain}(f)$:

$$\Pr\left[(X', \sigma) \stackrel{R}{\leftarrow} \mathsf{Blind}(X) : \mathsf{Unblind}(f(X', Y), \sigma) = f(X, Y)\right] = 1.$$

(2) Let α, β be any functions and $\mathcal{A}$ be any algorithm. We say that Blind satisfies *perfect privacy* if $\forall \mathcal{A}\ \forall \alpha\ \forall \beta\ \forall X \in \mathit{domain}(f)$:

$$\Pr\left[\mathcal{A}(|X|, \alpha(X)) = \beta(X)\right] = \Pr\left[(X', \sigma) \stackrel{R}{\leftarrow} \mathsf{Blind}(X) : \mathcal{A}(X', |X|, \alpha(X)) = \beta(X)\right]$$

the probability taken over the coin tosses of $\mathcal{A}$ and Blind.

We call a V-Oracle allowing private computation a *Private V-Oracle (PV-Oracle)*.

3.1 Oracle GIIs (O-GIIs)

We now extend the definition of computation in in Property 3 of Def. 2.2 to include computation by PV-Oracles. We call such a construction an *Oracle-GII (O-GII)*.

Definition 3.3. An O-GII construction has four PPT "algorithms" as described below (One of the algorithms, PV-Compute, is not an algorithm in the usual sense; it involves a call to a PV-Oracle).

Setup (τ): This randomized algorithm takes in as input a security parameter τ and outputs the system parameters params (containing the description of a group $(\mathbb{G}, \star, I)$) and a master key master-key.

Sample (params): This randomized algorithm outputs a uniformly selected element $X \stackrel{R}{\leftarrow} \mathbb{G}$ along with some auxiliary information σ_X, which we will call the *sampling information* in our construction.

Compute (params, master-key, X, Y): If $(X, Y) \notin \mathbb{G}^2$ this algorithm outputs I (recall that I is the identity element), otherwise it outputs $X \star Y$.

Define a PV-Oracle $\mathcal{O}$ such that $\mathcal{O}(X, Y) = \mathsf{Compute}(\mathsf{params}, \mathsf{master\text{-}key}, X, Y)$.

PV-Compute (params, X, Y): This algorithm uses the Verify, Blind and Unblind algorithms of Def. 3.2 to compute $Z \leftarrow \mathcal{O}(X, Y)$ privately and verifiably. It outputs $Z \in \mathbb{G}$.

3.1.1 Security of O-GIIs

We say that a PPT algorithm $\mathcal{A}$ breaks the O-GII if it is able to compute inverses in $\mathbb{G}$ having access to Compute via a PV-Oracle. We call this the *(black-box) Group Inversion Problem (GIP$_{\mathbb{G}}$)*. Formally, the advantage of $\mathcal{A}$ in solving GIP$_{\mathbb{G}}$ is defined as

$$\mathrm{Adv}^{\mathrm{GIP}}_{\mathcal{A}}(\tau) = \Pr\left[\ \mathcal{A}^{\mathcal{O}}(P, \mathsf{params}) = P^{-1} \ \middle|\ \begin{array}{c}(\mathsf{params}, \mathsf{master\text{-}key}) \xleftarrow{R} \mathsf{Setup}(\tau),\\ (P, \sigma_P) \xleftarrow{R} \mathsf{Sample}(\mathsf{params})\end{array}\ \right],$$

where the oracle $\mathcal{O}$ is defined as:

$$\mathcal{O}(\cdot,\cdot) = \mathsf{Compute}(\mathsf{params}, \mathsf{master\text{-}key}, \cdot, \cdot).$$

Definition 3.4. We say that algorithm $\mathcal{A}$ $(k_{\mathcal{O}}, \delta, \epsilon)$-breaks the O-GII f if $\mathcal{A}$ runs at most time δ; $\mathcal{A}$ makes at most $k_{\mathcal{O}}$ adaptive queries to $\mathcal{O}$; and $\mathrm{Adv}^{\mathrm{GIP}}_{\mathcal{A}}(\tau)$ is at least ϵ. We say that the O-GII is $(k_{\mathcal{O}}, \delta, \epsilon)$-secure if no such $\mathcal{A}$ exists.

Remark 3.5. It should be noted an O-GII is different from a *generic black-box group*, a notion introduced by Babai and Szemerédi [1] (see also, [43]), where access to the entire group $(\mathbb{G}, \star, I)$ is provided through black-box routines and the representation of group elements is opaque. In an O-GII, the representation of group elements is not opaque and sampling can be done outside of the black-box.

4 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 encryption scheme (although a variation of El-Gamal can also be used – see [47, §5.2.1]).

4.1 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.

Additionally, we assume that it is easy to sample elements from G_1. In a practical implementation, G_1 is the (additive) group 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 [7, 5] or the Tate pairing [2].

4.1.1 Problems in bilinear maps

Fix some generator g of G_1 and define the following problems.

Computational Diffie–Hellman Problem [$\mathbf{CDH_{(g,G_1)}}$] *Input:* $g, g^x, g^y \in G_1$ (for some $x, y \in \mathbb{Z}$). *Output:* g^{xy}.

Decision Diffie–Hellman Problem [$\mathbf{DDH_{(g,G_1)}}$] *Input:* $g, g^x, g^y, g^z \in G_1$ (for some $x, y, z \in \mathbb{Z}$). *Output:* 1 if $z \equiv xy \pmod{n}$, otherwise 0.

Inverse Diffie–Hellman Problem [$\mathbf{IDH_{(g,G_1)}}$] *Input:* $g, g^x \in G_1$ (for some $x \in \mathbb{Z}_n^*$). *Output:* $g^{1/x}$.

The following result was noted by Joux and Nguyen [27].

Lemma 4.1. *$DDH_{(g,G_1)}$ (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 $\text{DDH}_{(g,G_1)}$ is equivalent to computing the mapping $\hat{e}$ twice. □

The following result is also well known.

Lemma 4.2. *$IDH_{(g,G_1)} \Rightarrow CDH_{(g,G_1)}$ if $\phi(n)$ is known.*

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

(1) We know that $\forall u \in \mathbb{Z}_n^* : u^{\phi(n)-1} \equiv 1/u \mod n$ (Euler's theorem [35, p.69]).

(2) For every $(g^u, g^v) \in {G_1}^2$ and $u, v \in \mathbb{N}$ we can compute $g^{uv} \in G_1$ using the $\text{CDH}_{(g,G_1)}$ oracle.

(3) For every $g^u \in G_1$ and $i \in \mathbb{N}$, we can compute $g^{u^{2^i}}$ using the $\text{CDH}_{(g,G_1)}$ oracle.

Therefore, from g^x we can efficiently compute $h = g^{x^{\phi(n)-1}} \in G_1$ using the $\text{CDH}_{(g,G_1)}$ oracle and the "repeated squaring and multiply" method of [35, p. 71] (via facts 2 and 3). From fact 1, $h = g^{1/x}$. □

Although Lemma 4.2 states that $\text{IDH}_{(g,G_1)} \Rightarrow \text{CDH}_{(g,G_1)}$ 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:

Conjecture 4.3. $\text{IDH}_{(g,G_1)} \not\Rightarrow \text{CDH}_{(g,G_1)}$ if $\phi(n)$ is unknown.

4.1.2 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 [6] for details on generating composite order bilinear maps for any given n that is square free.

Using the idea of [5], 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 §4.1.

4.1.3 Hardness assumptions

For any PPT algorithm $\mathcal{A}$, define the advantage of $\mathcal{A}$ in solving $\text{CDH}_{(g,G_1)}$ for some security parameter τ as

$$\text{Adv}_{\mathcal{A}}^{\text{CDH}}(\tau) = \Pr\left[\begin{array}{l|c} \mathcal{A}(\hat{e}, n, G_1, G_2, g, g^x, h) = h^x & \begin{array}{c}(\hat{e}, G_1, G_2, p, q) \xleftarrow{R} \mathcal{BDH}(\tau) \\ \text{s.t. } |G_1| = |G_2| = pq, \\ g, h \xleftarrow{R} G_1 \text{ s.t. } \langle g \rangle = G_1, \\ n \leftarrow pq, x \xleftarrow{R} \mathbb{Z}_n\end{array}\end{array}\right]. \quad (4.1)$$

Similarly, define the advantage of $\mathcal{A}$ in solving $\text{IDH}_{(g,G_1)}$ as

$$\text{Adv}_{\mathcal{A}}^{\text{IDH}}(\tau) = \Pr\left[\begin{array}{l|c} \mathcal{A}(\hat{e}, n, G_1, G_2, g, g^x) = g^{\frac{1}{x}} & \begin{array}{c}(\hat{e}, G_1, G_2, p, q) \xleftarrow{R} \mathcal{BDH}(\tau) \\ \text{s.t. } |G_1| = |G_2| = pq, \\ g \xleftarrow{R} G_1 \text{ s.t. } \langle g \rangle = G_1, \\ n \leftarrow pq, x \xleftarrow{R} \mathbb{Z}_n^*\end{array}\end{array}\right]. \quad (4.2)$$

Our assumptions: There exists a negligible function $\nu : \mathbb{N} \mapsto [0, 1]$ such that:

CDH Assumption: $\forall \text{PPT} \mathcal{A}\ \exists \tau' \in \mathbb{N}\ \forall \tau > \tau' : \text{Adv}_{\mathcal{A}}^{\text{CDH}}(\tau) \leq \nu(\tau)$.

IDH Assumption: $\forall \text{PPT} \mathcal{A}\ \exists \tau' \in \mathbb{N}\ \forall \tau > \tau' : \text{Adv}_{\mathcal{A}}^{\text{IDH}}(\tau) \leq \nu(\tau)$.

4.2 The Paillier cryptosystem

Our idea of constructing the O-GII is to take an "ordinary" group G_1 of order n and convert it into a Gap-CDH group using an oracle as a 'Diffie–Hellman problem solver'. 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 **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 [37] has this property.[3]

Let $n = pq$, where p, q are distinct odd primes. Let $\lambda = \text{lcm}(p-1, q-1)$ and $\phi(n) = (p-1)(q-1)$. For any integer $x \equiv 1 \pmod{n}$ define $L(x) = (x-1)/n$. The following facts are trivial to verify.

(1) $|\mathbb{Z}_{n^2}^*| = n \cdot \phi(n)$.

(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) 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

$$x \equiv \frac{L(t^{x\nu} \bmod n^2)}{L(t^\nu \bmod n^2)} \pmod{n}. \tag{4.3}$$

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 [37] for details).

Key Generation: Generate $p, q \xleftarrow{R} \mathbb{N}$, where p, q are distinct primes. Set $n = pq$ and $\lambda = \text{lcm}(p-1, q-1)$. 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 Eq. 4.3. The semantic security of the above encryption scheme is proved under the *Decision Composite Residuosity Assumption* (DCRA) [37], which states that the following problem is hard unless the factors of n are known.

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

The DCRA is a stronger assumption than factoring [37]. See [12, 11] for a discussion on the bit-security of the Paillier cryptosystem.

[3] Although this property is necessary, it is not sufficient; the RSA [42] and Rabin [40] cryptosystems also have this property. However, our construction based on RSA or Rabin is insecure. For details, see [47, Appendix B.2].

4.2.1 Homomorphic properties

The Paillier cryptosystem has the following homomorphic properties [37].

(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:

$$\forall m \in \mathbb{Z}_n \; \forall r \in \mathbb{N} \quad \mathbf{D}(\mathbf{E}(m) r^n \bmod n^2) = m.$$

5 Our O-GII construction

Our construction will describe the four algorithms Setup, Sample, Compute, PV-Compute defined in §3.1. 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.

5.1 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 §4.1.2 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 §4.1). Then pick a generator g of G_1.

(2) Set $n \leftarrow pq$ and $\lambda \leftarrow \text{lcm}(p-1, q-1)$. Then generate an element $t \stackrel{R}{\leftarrow} \mathbb{Z}^*_{n^2}$ 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 **E** and **D** respectively.

(3) Generate $\alpha, r \stackrel{R}{\leftarrow} \mathbb{Z}^*_n$. 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 params $\leftarrow (\hat{e}, G_1, G_2, g, t, n, h, \beta)$ and master-key $\leftarrow \lambda$.

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

5.2 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, $|\mathbb{S}| = \phi(n)$ 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

$$\mathbb{G} = \{(x, y) | x = g^{\mathbf{D}(y)}\} \tag{5.1}$$

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

$$x_C \leftarrow {x_A}^{\mathbf{D}(y_B)} = g^{\mathbf{D}(y_A)\mathbf{D}(y_B)} = {x_B}^{\mathbf{D}(y_A)} \in G_1 \tag{5.2}$$

$$y_C \leftarrow \mathbf{E}(\mathbf{D}(y_A)\mathbf{D}(y_B) \bmod n) \in \mathbb{Z}_{n^2}^*. \tag{5.3}$$

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

(3) Finally, define a congruence 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 a congruence relation.

We state without proof the following lemmas:

Lemma 5.1. *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 5.2. *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 *congruence class* of A with respect to the relation $\sim$. Therefore we can define an congruence class $[I] \subsetneq \mathbb{G}$ as follows:

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

Lemma 5.3. *For any $[A] \subsetneq \mathbb{G}$, there exists a unique $[B] \subsetneq \mathbb{G}$ such that $[A] \star [B] = [I]$. Additionally, $[A] \star [I] = [A]$.*

From the above, it is clear that the relation $\sim$ transforms the congruence 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 $[A] \star [A] \star \ldots \star [A]$ (i times). The inverse of $[A]$ is denoted by $[A]^{-1}$.

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

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

5.3 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 at least one of $\{A, B\}$ 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) **Black-Box Computability:** Let $A, B \in \mathbb{G}$ be given. Anyone knowing λ has the ability to compute $A \star B$ using equations 5.3 and 5.2.

(7) **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 $(x, y) \stackrel{?}{\in} \mathbb{G}$ by virtue of Eq. 5.1.

5.4 A concrete O-GII construction

We now describe a concrete construction of an O-GII under Def. 3.3. 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. Setup is described in §5.1 while Sample is described in §5.3, 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 §5.1.

Step-2. Set params $\leftarrow (\hat{e}, G_1, G_2, g, t, n, h, \beta)$ and master-key $\leftarrow \lambda$.

Output: (params, master-key)

A-2.

Sample

Input: params

Step-1. Generate $\sigma_A, r \stackrel{R}{\leftarrow} \mathbb{Z}_n^*$

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 5.4. From the value params, the pair $(h, \beta) \in \mathbb{G}$ such that its sampling information $\alpha \in \mathbb{Z}_n^*$ is unknown (see §5.1).

A high level description of Compute 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 master-key $= \lambda$ to decide if $(A, B) \stackrel{?}{\in} \mathbb{G}^2$ [see §5.3, Item 7]

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 §5.3, Item 6]

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

Functionality of Oracle $\mathcal{O}$: Access to Compute is provided in a black-box manner via the oracle $\mathcal{O}$ that knows master-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 $C \leftarrow$ Compute(master-key, params, A, B)

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

Remark 5.5. 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 5.6. Assuming that access to oracle $\mathcal{O}$ is authentic, we can use $\mathcal{O}$ to decide if any given pair $(A, B) \stackrel{?}{\in} \mathbb{G}^2$. Additionally we can use $\mathcal{O}$ to compute A^i for any $A \in \mathbb{G}$ and $i \in \mathbb{N}$ using the "repeated squaring and multiply" method [35, 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, Verify-In-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 $u_1, u_2, v_1, v_2, \stackrel{R}{\leftarrow} \mathbb{Z}_n$ and $w_1, w_2 \stackrel{R}{\leftarrow} \mathbb{Z}^*_n$

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 $y_1 \leftarrow y^{u_1} t^{v_1} {w_1}^n \bmod n^2$; $y_2 \leftarrow y^{u_2} t^{v_2} {w_2}^n \bmod n^2$; 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 result $\leftarrow 1$

Output: 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, Verify-Not-In-Group enables us to do this using oracle $\mathcal{O}^*$. If $(x, y) \in \mathbb{G}$ the algorithm outputs 0 with a high probability.[4]

[4]The reader should note that Verify-Not-In-Group is never used in any of the protocols discussed in this paper. It is provided only for completeness of our O-GII 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 $\in \{0,1\}$

Lemma 5.7 shows that Verify-Not-In-Group is sound if Verify-In-Group is sound.

Lemma 5.7. *If* Verify-In-Group *is sound, then* Verify-Not-In-Group *is also sound.*

Proof. We must show that if Verify-Not-In-Group 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 Verify-In-Group 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, $\Pr[(a_i = b_i) \forall 1 \leq i \leq j] = \frac{1}{2^j}$. In other words, if $(x, y) \in \mathbb{G}$ the probability that Verify-Not-In-Group 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, Verify is sound if Verify-In-Group 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, the inclusion of this call enables us to reduce the soundness of other related algorithms to the soundness of Verify-In-Group.

Algorithm V-Compute takes as input two elements $A, B \in \mathbb{G}$. It uses Verify-In-Group 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 $\mathsf{Verify}(A, B, C) = 0$, set $C \leftarrow I \in \mathbb{G}$

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

Clearly, the soundness of the above algorithm reduces to that of the Verify algorithm. As a consequence, if Verify is sound then having indirect access to the oracle $\mathcal{O}$ via some active adversary $\mathcal{O}^*$ is the same as having authentic and public access to $\mathcal{O}$. For completeness, we state this in Lemma 5.8.

Lemma 5.8. *If* Verify *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 used primarily 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 $r \stackrel{R}{\leftarrow} \mathbb{Z}_n^*$

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 $B \stackrel{R}{\leftarrow} \mathbb{G}$ 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 a 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 $(B, \sigma_B) \stackrel{R}{\leftarrow} \mathsf{Sample}(\mathsf{params}) \in \mathbb{G} \times \mathbb{Z}_n^*$ [B will be ignored]

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

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 5.9. *The* Blind/Unblind *algorithms provide perfect privacy.*

Proof. The Blind and Unblind algorithms are inverses of each other. Now, if the output of Sample is uniformly distributed over $\mathbb{G}$, then the output of Blind is also uniformly distributed over $\mathbb{G}$, independent of the input. This is sufficient to prove the lemma. □

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 $(A', \sigma_{A'}) \stackrel{R}{\leftarrow}$ Blind(params, $A) \in \mathbb{G} \times \mathbb{Z}_n^*$

Step-2. Set $(B', \sigma_{B'}) \stackrel{R}{\leftarrow}$ 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 perfect privacy (Lemma 5.9), the soundness of the above algorithm also reduces to that of the Verify algorithm. We summarize this in Lemma 5.10, which says that if Verify is sound, then having indirect access to the oracle $\mathcal{O}$ via some active adversary $\mathcal{O}^*$ is the same as having private and authentic access to $\mathcal{O}$.

Lemma 5.10. *If* Verify *is sound then $\mathcal{O}$ is a PV-Oracle.*

This completes the construction. Figure 2 gives the dependencies between the eleven algorithms. 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 the adversary is bounded by a polynomial in τ.

5.5 Notation

For convenience, we will adopt the following shorthand notation.

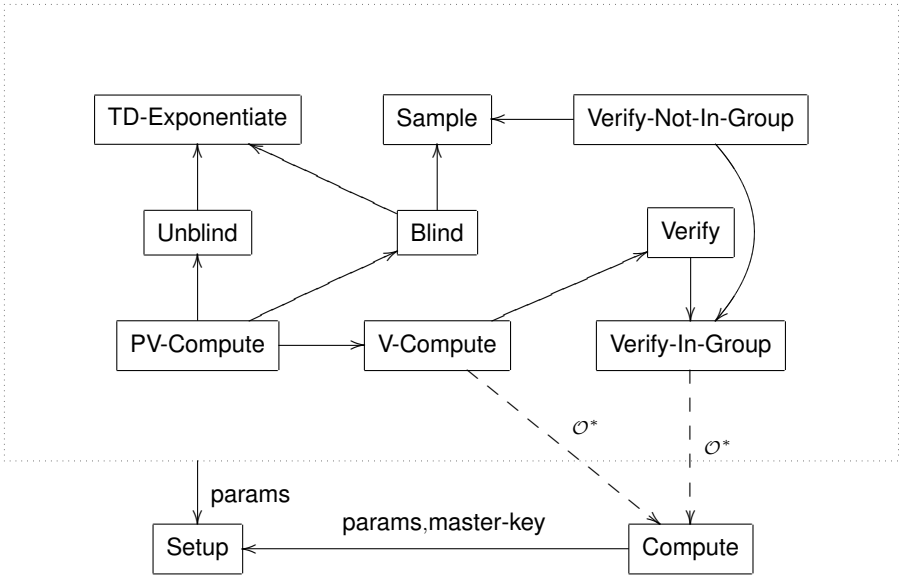

Figure 2. Dependencies between the algorithms.

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 $\mathcal{O}$ (Lemma 5.8), 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 $\mathcal{O}$ (Lemma 5.10). We will denote PV-Compute(params, A, B) by $\widehat{\mathcal{O}}(A, B)$.

4 For any 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 perfect privacy to inputs and verifiability of outputs.

Remark 5.11. 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 [35, p. 71].

5.6 Security of the construction

Recall that out of params, the pair $(h, \beta) \in \mathbb{G}$. Denote this value by P. The security of our O-GII 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$_\mathbb{G}$] *Input:* $P \in \mathbb{G}$. *Output:* $P^{-1} \in \mathbb{G}$, possibly by using the oracle $\mathcal{O}$.

We hypothesize that any method of reducing IDH$_{(g,G_1)}$ to CDH$_{(g,G_1)}$ will yield a method of reducing GIP$_\mathbb{G}$ to the oracle $\mathcal{O}$. We define the advantage of an algorithm for solving the group inversion problem as follows.

Definition 5.12. For any algorithm $\mathcal{A}$, the advantage of $\mathcal{A}$ in solving the group inversion problem $\mathrm{Adv}^{\mathrm{GIP}}_{\mathcal{A}}(\tau)$ for some security parameter τ is defined as: $\mathrm{Adv}^{\mathrm{GIP}}_{\mathcal{A}}(\tau) =$

$$\Pr\left[\mathcal{A}^{\mathcal{O}}(\hat{e}, G_1, G_2, n, g, t, g^\alpha, \mathbf{E}(\alpha)) = (g^{\frac{1}{\alpha}}, \mathbf{E}(\tfrac{1}{\alpha})) \;\middle|\; \begin{array}{c} (\hat{e}, G_1, G_2, p, q) \overset{R}{\leftarrow} \mathcal{BDH}(\tau) \\ \textit{s.t. } |G_1| = |G_2| = pq, \\ g \overset{R}{\leftarrow} G_1 \textit{ s.t. } \langle g \rangle = G_1, \\ n \leftarrow pq, \alpha \overset{R}{\leftarrow} \mathbb{Z}^*_n, \\ t \overset{R}{\leftarrow} \mathbb{Z}^*_{n^2} \textit{ s.t. } |\langle t \rangle| = n\lambda, \\ h \leftarrow g^\alpha, \beta \leftarrow \mathbf{E}(\alpha) \end{array}\right],$$

where $\mathcal{BDH}$ is the BDH parameter generator (§4.1.2); $\mathbf{E}$ is Paillier encryption with public key (t, n) (§4.2), and $\mathcal{O}$ is an oracle implementing *Compute* (§5.4).

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 conjecture.

Conjecture 5.13. For any algorithm $\mathcal{A}$ such that $k_{(\mathcal{O},\mathcal{A})}, \delta_\mathcal{A} \in \mathit{Poly}(\tau)$, $\mathrm{Adv}^{\mathrm{GIP}}_{\mathcal{A}}(\tau)$ is a negligible function in τ. In other words, for all $k_\mathcal{O}, \delta, 1/\epsilon \in \mathit{Poly}(\tau)$, the O-GII is $(k_\mathcal{O}, \delta, \epsilon)$-secure under an adaptive attack using Def. 3.4.

Remark 5.14. In the generic group model [52], it is known that if $\mathrm{Adv}^{\mathrm{GIP}}_{\mathcal{A}}(\tau) \approx 1$, then $k_\mathcal{O} \approx \sqrt{|\mathbb{G}|}$.

6 Applications of O-GIIs

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

6.1 Key generation

To participate in the protocols below, each user i needs 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 $(X_i, \sigma_{X_i}) \stackrel{R}{\leftarrow} \mathsf{Sample}(\mathsf{params}) \in \mathbb{G} \times \mathbb{Z}_n^*$. 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).

See Appendix B for 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.

6.2 Multiparty key agreement

In this section, we describe the multiparty key agreement protocol of Rabi and Sherman [39] using O-GIIs. 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.

6.2.1 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

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

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

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

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

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

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

6.2.2 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 PV-Compute.[5] Thus, X_i can be considered as a 'restricted' private key corresponding to the 'unrestricted' 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 to security.

6.2.3 Join and merge operations

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

Example 6.1 (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-GII

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

is only used for key agreement (and not for signatures, which are discussed below in §6.3). In case the same instantiation of O-GII 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 abovementioned 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 $c \subsetneq a$. (See also, Footnote 6.)

6.2.4 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 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:

$$K_a = P^{k-1} \prod_{i=1}^{k} X_i = P^{-1} \star \prod_{i=1}^{k} Y_i. \tag{6.1}$$

Security Model: Boneh and Silverberg defined the security of one-round multiparty key agreement in [9], where they require the adversary to compute the group key of a given set of users. Their definition captures security under a *known subset key attack*, because the attacker is not allowed to choose the set of public keys to attack.[6] Here, we use a stronger model – we allow the adversary to choose the subset of keys to attack. We call this security under *chosen subset key attack*. This is defined by Game 1 below.

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 $\mu_1 \in \mathbb{N}$.

[6] Although several stronger notions of security of key agreement protocols have been proposed [53, 31] for interactive schemes, we use the model of Boneh and Silverberg [9], which is standard for non-interactive (i.e., one-round) schemes. The main difference between the stronger model is that several standard features of interactive schemes must be sacrificed in order to enjoy the benefits of non-interactivity ([47, §2.2.1]). Specifically, we cannot obtain *key freshness* and *perfect forward secrecy*. However, we can use standard techniques to fix this at the expense of one interaction [47, p. 58].

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 if $a \subseteq \{1, 2, \ldots, \mu_1\}$ and K_a is the group private key of a.

Definition 6.2. We say that adversary $\mathcal{A}$ $(\mu_1, \delta_1, \epsilon_1)$-breaks the key agreement protocol in the *chosen subset 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 chosen-subset key attack if no such adversary exists.

Theorem 6.3 shows that the key agreement protocol is secure under a chosen-subset key attack if the group inversion problem is hard.

Theorem 6.3. *Let the O-GII 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 chosen-subset 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-GII 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 chosen-subset key attack. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to solve $\text{GIP}_{\mathbb{G}}$ in time at most δ and 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 Eq. 6.1. 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 §5.3, 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

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

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. □

6.3 Signatures

As noted in [38], SAOWFs (and so GIIs) give rise to signature schemes. Here, we describe two signature schemes using O-GIIs: ordinary signatures and multi-user signatures. A signature scheme consists of three algorithms KeyGen, Sign and VerifySig, where the algorithms have their usual constraints [7]. Our message space is $\mathbb{N}$.

6.3.1 Single-user signatures

This is a variation of the scheme for single-user signatures described in [39].

KeyGen. This algorithm is described in §6.1. 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 ${Y_i}^m \stackrel{?}{=} S_{(i.m)} \star P^{m-1}$.

6.3.2 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 §6.2. 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 [44].

6.3.3 Security of the signature schemes

The strongest model for security of signatures is security against *existential forgery* under an *adaptive chosen message attack* [7], where the attacker is required to output a successful forgery under the challenge public key after having access to the signing oracle. However, we prove the security of our schemes only 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 $\{Y_1, Y_2, \ldots, Y_{\mu_2}\}$ 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 $a \subseteq \{1, 2, \ldots, \mu_2\}$ and $S_{(a,m)}$ is a valid signature by a on the message m and $m \leq n$.

Definition 6.4. 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.

Theorem 6.5 shows that any algorithm that is successful in existential forgery of signatures under a non-adaptive chosen message attack can be used to solve $\text{GIP}_{\mathbb{G}}$. First observe that $S_{(a,m)}$ can be rewritten as

$$S_{(a,m)} = {K_a}^m \star P = P^{1-m} \star (\prod_{i=1}^{k} Y_i)^m. \tag{6.2}$$

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

Theorem 6.5. *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-GII 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-GII 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 $\text{GIP}_{\mathbb{G}}$ 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 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 $(Y_1, \sigma_{Y_1}), (Y_2, \sigma_{Y_2}), \ldots, (Y_j, \sigma_{Y_{\mu_2}}) \xleftarrow{R} \mathsf{Sample}(\mathsf{params}) \in \mathbb{G} \times \mathbb{Z}_n^*$ and gives the $(\mu_2 + 1)$-tuple $(Y_1, Y_2, \ldots, Y_{\mu_2}, P)$ 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 $a \subseteq \{1, 2, \ldots, \mu_2\}$, $S_{(a,m)} = P^{1-m} \star (\prod_{i \in a} Y_i)^m$ (by virtue of Eq. 6.2), 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 §5.3, 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

$$\begin{aligned} \mathcal{T}(\sigma_Y, -m, \mathcal{O}(S_{(a,m)}, C)) &= \Big(\prod_{i \in a} Y_i\Big)^{-m} \star S_{(a,m)} \star C \\ &= \Big(\prod_{i \in a} Y_i\Big)^{-m} \star (P^{1-m} \star \prod_{i \in a} Y_i) \star (P^{m-2}) = P^{-1}. \end{aligned}$$

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. □

6.4 Broadcast encryption

In a broadcast encryption scheme [17], 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* [13]). 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 [17]. (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 §6.1, where the initial public key infrastructure is created for individual users. We additionally require a cryptographic hash function $\mathcal{H} : \mathbb{G} \mapsto \{0,1\}^l$, that will be treated as a random oracle in the proofs.[7] 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 §6.2).

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 §6.2).

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 $(R, \sigma_R) \stackrel{R}{\leftarrow} \mathsf{Sample}(\mathsf{params}) \in \mathbb{G} \times \mathbb{Z}_n^*$ 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 $\oplus$ is the XOR operator. The ciphertext is $C = (c_1, C_2) \in \{0,1\}^l \times \mathbb{G}$.

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

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

6.4.1 Security of broadcast encryption

We use a restricted notion of security, namely 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 [19], we prove security only in the IND-CPA model. The IND-CPA security of a broadcast encryption scheme is defined using the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

[7] 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 $\mathcal{H}(A) = \mathcal{H}_1(x)$.

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. $\mathcal{A}$ outputs two messages m_0, m_1 along with a set $a \subseteq \{1, 2, \ldots, \mu_3\}$ of users. The challenger chooses a bit $b \xleftarrow{R} \{0, 1\}$ 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 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 6.6. 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.

Theorem 6.7 shows that any IND-CPA adversary $\mathcal{A}$ with non-negligible advantage 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 [5, Lemma 4.3].

Theorem 6.7. *Let $\mathcal{H}$ be a random oracle and let the O-GII 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 $k_{\mathcal{O}} \leq k_3$; $\delta \leq \delta_3 + \Theta(c_1\mu_3) + \Theta(c_2 k_3)$; 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-GII 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 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. $\mathcal{A}$ outputs 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 $(C_2, \sigma_{C_2}) \stackrel{R}{\leftarrow} \mathsf{Sample}$; 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 §5.3, 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

$$\mathcal{O}(Z_j, P) = W. \tag{6.3}$$

If such a tuple does not exist in the $\mathcal{H}^{list}$, algorithm $\mathcal{B}$ reports failure and terminates. Otherwise, $\mathcal{B}$ sets $\mathsf{result} \leftarrow \mathcal{T}(\sigma_W, -1, Z_j) = W^{-1} \star Z_j$ and 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 [5, Lemma 4.3], we can conclude that

$$\Pr\left[\text{a tuple } \langle Z_j, \mathcal{H}_j \rangle \text{ appears in the } \mathcal{H}^{list} \text{ such that Eq. 6.3 is satisfied}\right] \geq 2\epsilon_3.$$

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 □

7 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-GII 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.

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.

$\|n\|$ (bits)	512	768	1024	1536	2048	2656	3072	4096
Time (ms)	40	110	300	990	2150	3080	3830	7200

Table 2. Typical query times.

7.1 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 ($\approx$ 1024 bits) for moderate security applications.[8] This also makes computing discrete logarithms in G_1 intractable using Pollard's rho method [35, p.128]. Using these parameters elements of $\mathbb{G}$ can be represented with at most $\approx$ 384 bytes. The public keys Y_i of §6.1, 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.

7.2 Query overhead

In all the above protocols, we have been working with the congruence classes of $\mathbb{G}$ rather than the individual elements themselves. For any $A = (x, y) \in \mathbb{G}$, the congruence 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 §5.4 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 and the query time is given in Table 2. We used a finite field instead of bilinear maps (see § 7.3). The times are measured on a Pentium 4 CPU (2.53 Ghz) server with 1 GB RAM running Red Hat Linux kernel 2.4.21-47.

7.3 Verifiability of the oracle

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

[8] See "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.

However, since we already trust the black-box to keep the trapdoor information (λ) secret, we can also trust the black-box to correctly evaluate the group operation.[9]

1 Our first simplification is to 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 §5.2 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 $\text{Sig}_{X,Y,Z}$ 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 §4.2 and setting $r = 1$. In this case, the set $\mathbb{S} = \{y | y = \mathbf{E}(x) \wedge x \in \mathbb{Z}_n^*\}$.[10]

7.4 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 $A_1 \star A_2 \star \ldots \star A_i$. 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.

7.5 Decentralizing the oracle

Decentralization of the oracle is desirable, since each query 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 compromising even one would compromise the entire system.

Finally, we can also consider a system where the oracle is implemented in a tamper-proof device and is made available to all users (for instance as a smart-card, or inside the CPU itself).

We close this section with a comparison of our scheme with previously proposed group key agreement methods in Table 3.

8 Conclusion

In this paper, we presented a practical implementation of a new cryptographic primitive known as an Oracle-based Group with Infeasible Inversion (O-GII). 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 [39]. The scheme

[9] 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.

[10] 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-GII**	**GDH** basic [54]	**AGKE** [29]	**GKE** [10]
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 man-in-the-middle attack	Authentication after 2nd round	Insecure under an active attack [36]
Protection under passive attack	Group Inversion Problem	Diffie–Hellman Problem	Diffie–Hellman Problem	Diffie–Hellman Problem

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

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

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

Table 3. Comparison of our group key agreement scheme.

can be extended to group signatures as demonstrated in §6.3. 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. We conclude this section with some open questions.

1 Prove/disprove Conjecture 5.13.

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

3 Distribute the oracle using techniques of threshold cryptography.

4 Remove the oracle from our construction. In other words, exhibit a practical GII construction.

Appendix

A Soundness of Verify-in-group algorithm

The reader is referred to §5.4, Algorithm A-4 for the notation used here. First we define the following problem.

Decision Exponent Class Problem [$\text{DECP}_{(t,n,g,G_1)}$]: Given $\{t, n, g, G_1\} \subset$ 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.

Theorem A.1 states that Verify-In-Group is sound if the $\text{DECP}_{(t,n,g,G_1)}$ and $\text{CDH}_{(g,G_1)}$ problems are intractable.

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

Proof. The input to Verify-In-Group 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

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

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, $f_1(u, v, w) \in \mathbb{G} \; \forall \; (u, v, w) \in \textit{domain}(f_1)$. In this case, the output of Verify-In-Group 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

$$\left.\begin{array}{rcrl} a_1 & \equiv & au_1 + v_1 & \pmod n \\ b_1 & \equiv & bu_1 + v_1 & \pmod n \\ r_1 & \equiv & r^{u_1}w_1 & \pmod n \end{array}\right\} \tag{A.1}$$

Clearly, Eq. A.1 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 $\text{CDH}_{(g,G_1)}$ (see §4.1) 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 $\text{CDH}_{(g,G_1)}$ 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 $\text{CDH}_{(g,G_1)}$ 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 $\text{CDH}_{(g,G_1)}$ are intractable. □

B Proof of knowledge of private keys of §6.1

Recall from §6.1, 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 §6.2.2).

For convenience, 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 [21] and quadratic residuosity [18, 16].

B.1 Zero-knowledge proof of knowledge of group element

The following is a proof of knowledge 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).

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 $U_0 \stackrel{R}{\leftarrow} \mathbb{G}$ 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:

$$W_0 \star W_1 \stackrel{?}{=} Y \star P. \tag{B.1}$$

$\mathcal{V}$ terminates the protocol if the above 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.

4 On receiving U_b, $\mathcal{V}$ accepts iff the following equality holds:

$$W_b = U_b \star P. \tag{B.2}$$

The above is an atomic zero-knowledge proof of knowledge of X with an error factor of $1/2$. The protocol is (sequentially) repeated k times to reduce this error to $1/2^k$. Theorem B.1 shows that the above protocol is indeed a zero-knowledge proof of knowledge of X.

Theorem B.1. *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). We must show that the protocol is a proof of knowledge (i.e. has an *extractor* [3]) and is zero-knowledge (i.e. has a *simulator* [21]).

Proof of Knowledge: We must exhibit an extractor for X. Similar to [16], our extractor is constructed 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 $W_0 \star W_1 = Y \star P = X \star P^2$ (Eq. B.1) and $U_b = W_b \star P^{-1}$ (Eq. B.2). Therefore, once we obtain (U_0, U_1) we can compute $X = U_0 \star U_1$.

Zero-Knowledge: The simulator for zero-knowledge is similar to that of [16]. Assume that the simulator interacts with the verifier and knows in advance what the challenge bit b is going to be. It proceeds 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 Eq. B.1. 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 Eq. B.2. Clearly, the transcript of the simulation is distributed identically to that of a real interaction. □

B.2 Zero-knowledge proof of knowledge of sampling information

The following is a zero-knowledge proof of knowledge 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.

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 $(U, \sigma_U) \stackrel{R}{\leftarrow} \mathsf{Sample}(\mathsf{params}) \in \mathbb{G} \times \mathbb{Z}_n^*$ 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 Verify-In-Group algorithm of §5.4 and terminates the protocol if $W \notin \mathbb{G}$. 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 $\sigma_Z = \sigma_U \cdot {\sigma_X}^b \bmod n \in \mathbb{Z}_n^*$.

4 On receiving σ_Z, $\mathcal{V}$ computes $Z \in \mathbb{G}$ such that σ_Z is the sampling information of Z and accepts iff the following equality holds:

$$W = Z \star Y^{1-b} \star P^b. \tag{B.3}$$

Like the previous protocol, the above is an atomic zero-knowledge proof of knowledge of σ_X with an error factor of $1/2$. The protocol is (sequentially) repeated k times to reduce this error to $1/2^k$. Theorem B.2 shows that the above protocol is indeed a zero-knowledge proof of knowledge of σ_X.

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

Proof. First note that the protocol is complete.

Proof of Knowledge: Construction of an 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 a simulator interacting with the verifier knows in advance what the challenge bit b is going to be. The simulator runs as follows: It generates $(Z, \sigma_Z) \stackrel{R}{\leftarrow} \mathsf{Sample}(\mathsf{params}) \in \mathbb{G} \times \mathbb{Z}_n^*$ 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 Eq. B.3 will be satisfied irrespective of the choice of bit b. □

C Identity based encryption using O-GIIs

In this section, we give (without a security proof) an Identity Based Encryption (IBE) scheme as another application of our O-GII. We refer the reader to [5] for the definitions of an IBE scheme and to §5.4 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 Setup-IBE takes as input some security parameter. It outputs the IBE system parameters par and the IBE master key m-key.

2 KeyGen takes as input the value par, m-key and a random string i. It outputs the private key prv-key$_i$ corresponding to the string i.

3 ID-Encrypt takes as input par, a random message m and a random string i. It outputs a ciphertext c.

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

The ID-Encrypt and ID-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 m-key is known only to a trusted authority known as the Key Generating Center (KGC) that is responsible for distributing private keys. 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 Setup-IBE: Set $(X, \sigma_X), (Y, \sigma_Y) \stackrel{R}{\leftarrow} \mathsf{Sample}(\mathsf{params})$ and set $Z \leftarrow \mathcal{T}(\sigma_X, 1, Y) = X \star Y$. Finally set par $\leftarrow (Y, Z) \in \mathbb{G}^2$; m-key $\leftarrow (\sigma_X, \sigma_Y) \in {\mathbb{Z}_n^*}^2$ and output (par, m-key).

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

3 ID-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 $(R, \sigma_R) \stackrel{R}{\leftarrow} \mathsf{Sample}(\mathsf{params})$. 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 ID-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:

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

We leave the security of the above construction as an open question.

Acknowledgments. We would like to thank Ronald Rivest, Leonid Levin, Virendra Sule, Pascal Paillier and Chunbo Ma for useful feedback.

References

[1] László Babai and Endre Szemerédi, *On the Complexity of Matrix Group Problems I*. FOCS'1984, pp. 229–240, 1984.

[2] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott, *Efficient Algorithms for Pairing-Based Cryptosystems*. CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology, pp. 354–368. Springer-Verlag, London, UK, 2002.

[3] Mihir Bellare and Oded Goldreich, *On Defining Proofs of Knowledge*, Lecture Notes in Computer Science 740 (1993), pp. 390–420.

[4] 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 (1999), pp. 25–40.

[5] Dan Boneh and Matthew K. Franklin, *Identity-Based Encryption from the Weil Pairing*, SIAM J. Comput. 32 (2003), pp. 586–615.

[6] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim, *Evaluating 2-DNF Formulas on Ciphertexts*. TCC (Joe Kilian, ed.), Lecture Notes in Computer Science 3378, pp. 325–341. Springer, 2005.

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

[8] ———, *Short Signatures from the Weil Pairing*, J. Cryptology 17 (2004), pp. 297–319.

[9] Dan Boneh and Alice Silverberg, *Applications of Multilinear Forms to Cryptography*, Contemporary Mathematics, American Mathematical Society 324 (2003), pp. 71–90.

[10] E. Bresson, O. Chevassut, A. Essiari, and D. Pointcheval, *Mutual Authentication and Group Key Agreement for Low-Power Mobile Devices*, 2003.

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

[12] Dario Catalano, Rosario Gennaro, and Nick Howgrave-Graham, *The Bit Security of Paillier's Encryption Scheme and Its Applications*. EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques, pp. 229–243. Springer-Verlag, London, UK, 2001.

[13] Hervé Chabanne, Duong Hieu Phan, and David Pointcheval, *Public Traceability in Traitor Tracing Schemes*. EUROCRYPT (Ronald Cramer, ed.), Lecture Notes in Computer Science 3494, pp. 542–558. Springer, 2005.

[14] Jung Hee Cheon and Dong Hoon Lee, *Diffie-Hellman Problems and Bilinear Maps*, Cryptology ePrint Archive, Report 2002/117, 2002.

[15] Whitfield Diffie and Martin E. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory IT-22 (1976), pp. 644–654.

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

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

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

[19] Eiichiro Fujisaki and Tatsuaki Okamoto, *Secure Integration of Asymmetric and Symmetric Encryption Schemes*, Lecture Notes in Computer Science 1666 (1999), pp. 537–554.

[20] S. D. Galbraith, F. Hess, and F. Vercauteren, *Aspects of Pairing Inversion*, Cryptology ePrint Archive, Report 2007/256, 2007.

[21] 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 (1991), pp. 690–728.

[22] 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 (1999), pp. 648–659.

[23] Lane A. Hemaspaandra, Jörg Rothe, and Amitabh Saxena, *Enforcing and Defying Associativity, Commutativity, Totality, and Strong Noninvertibility for One-Way Functions in Complexity Theory*. ICTCS (Mario Coppo, Elena Lodi, and G. Michele Pinna, eds.), Lecture Notes in Computer Science 3701, pp. 265–279. Springer, 2005.

[24] Susan Hohenberger, *The Cryptographic Impact of Groups with Infeasible Inversion*, Master's thesis, Massachusetts Institute of Technology, 2003, Supervised by: Ronald L. Rivest.

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

[26] Antoine Joux, *A One Round Protocol for Tripartite Diffie-Hellman*. ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory, pp. 385–394. Springer-Verlag, London, UK, 2000.

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

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

[29] Hyun-Jeong Kim, Su-Mi Lee, and Dong Hoon Lee, *Constant-Round Authenticated Group Key Exchange for Dynamic Groups*. ASIACRYPT (Pil Joong Lee, ed.), Lecture Notes in Computer Science 3329, pp. 245–259. Springer, 2004.

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

[31] Mark Manulis, *Survey on Security Requirements and Models for Group Key Exchange*, Cryptology ePrint Archive, Report 2006/388, 2006, `http://eprint.iacr.org/`.

[32] Ueli M. Maurer, *Towards the Equivalence of Breaking the Diffie-Hellman Protocol and Computing Discrete Algorithms*. CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology, pp. 271–281. Springer-Verlag, London, UK, 1994.

[33] Ueli M. Maurer and Stefan Wolf, *Lower Bounds on Generic Algorithms in Groups*. EUROCRYPT, pp. 72–84, 1998.

[34] ———, *The Relationship Between Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms*, SIAM Journal on Computing 28 (1999), pp. 1689–1721.

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

[36] 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.

[37] Pascal Paillier, *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. EUROCRYPT, pp. 223–238, 1999.

[38] M. Rabi and A. Sherman, *Associative One-Way Functions: A New Paradigm for Secret-Key Agreement and Digital Signatures*, Department of Computer Science, University of Maryland Baltimore County, Baltimore, MD, 1993, Report no. CS-TR-3183/UMIACS-TR-93-124, 1993.

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

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

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

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

[43] Ronald L. Rivest, *On the Notion of Pseudo-Free Groups*. TCC (Moni Naor, ed.), Lecture Notes in Computer Science 2951, pp. 505–521. Springer, 2004.

[44] Ronald L. Rivest, Adi Shamir, and Yael Tauman, *How to Leak a Secret*. ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security, pp. 552–565. Springer-Verlag, London, UK, 2001.

[45] Ahmad-Reza Sadeghi and Michael Steiner, *Assumptions Related to Discrete Logarithms: Why Subtleties Make a Real Difference*, Lecture Notes in Computer Science 2045 (2001), pp. 244–261.

[46] Takakazu Satoh, *On Pairing Inversion Problems*. Pairing (Tsuyoshi Takagi, Tatsuaki Okamoto, Eiji Okamoto, and Takeshi Okamoto, eds.), Lecture Notes in Computer Science 4575, pp. 317–328. Springer, 2007.

[47] Amitabh Saxena, *New Paradigms for Group Oriented Cryptography: Non-Interactive Key Agreement, Chain Signatures and Additive Proofs of Knowledge*, Thesis (Ph.D.), School of Engineering and Mathematical Sciences, La Trobe University, Bundoora, VIC, Australia, January 2007, Supervised by Ben Soh. Available at `http://amitabh.home.googlepages.com/thesis.pdf`.

[48] Amitabh Saxena and Ben Soh, *A new paradigm for group cryptosystems using quick keys*. Proceedings of the The 11th IEEE International Conference on Networks (ICON2003), pp. 385–389, Sydney, Australia, 2003.

[49] ———, *A Novel Method For Authenticating Mobile Agents With One-Way Signature Chaining*. Proceedings of The 7th International Symposium on Autonomous Decentralized Systems (ISADS 05), pp. 187–193. IEEE Computer Press, China, 2005.

[50] ———, *A New Cryptosystem Based On Hidden Order Groups*, Cryptology ePrint Archive, Report 2006/178, 2006, Available at `http://eprint.iacr.org/2006/178.pdf`.

[51] 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, p. 221.

[52] Victor Shoup, *Lower Bounds for Discrete Logarithms and Related Problems*. Advances in Cryptology — EUROCRYPT '97 (Walter Fumy, ed.), Lecture Notes in Computer Science 1233, pp. 256–266. Springer-Verlag, May 1997.

[53] M. Steiner, *Secure Group Key Agreement*, Ph.D. thesis, Universitat des Saarlandes, 2002.

[54] Michael Steiner, Gene Tsudik, and Michael Waidner, *CLIQUES: A New Approach to Group Key Agreement*. Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98), pp. 380–387. IEEE Computer Society Press, Amsterdam, 1998.

[55] A. C. Yao, *Theory and Applications of Trapdoor Functions*. Proceedings of the 23rd IEEE Annual Symposium on the Foundations of Computer Science, pp. 80–91, 1982.

[56] ———, *Computational information theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1988.

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

Received 23 April, 2007; revised 24 September, 2008

Author information

Amitabh Saxena, International University in Germany, 76646 Bruchsal, Germany.
Email: `amitabh.saxena@i-u.de`

Ben Soh, La Trobe University, Bundoora VIC 3086, Australia.
Email: `b.soh@latrobe.edu.au`