Group Decryption*

Bo Qin[1,2], Qianhong Wu[2], Willy Susilo[2], Yi Mu[2],

[1] National Key Laboratory of Integrated Service Networks
Xidian University, Xi'an City 710048
Shaanxi Province, P. R. China
qinboo@xaut.edu.cn
[2] Center for Information Security Research
School of Information Technology and Computer Science
University of Wollongong
Wollongong NSW 2522, Australia
{qhw,wsusilo,ymu}@uow.edu.au

Abstract. Anonymity is one of the main concerns in group-orient cryptography. However, most efforts, for instance, group signatures and ring signatures, are only made to provide anonymity on the sender side. There is few work done to ensure anonymity in a cryptographic sense on the recipient side in group-orient communications. In this paper, we formalize the notion of anonymous group decryptions. It is an analog of group signatures in the context of public key encryptions. In this notion, a sender can encrypt a committed message to any intended member of a group managed by a group manager while *the recipient of the ciphertext keeps anonymous*. The sender can convince a verifier this fact without leaking the plaintext or the identity of the recipient. If necessary, the group manger can verifiably open the identity of the recipient. We propose the first anonymous group decryption scheme that is proven secure in the random oracle model. The overhead in both computation and communication is independent of the group size and the scheme is practical.

1 Introduction

Anonymity is a main concern in group-orient cryptography. It has attracted a lot of attentions in the context of digital signatures and extensively researched in the notions of such as group signatures, ring signatures and etc. However, such notions of anonymous signatures only provide anonymity on the sender side in the communication. There is few work done to ensure anonymity on the recipient side in a cryptographic sense. This paper concentrates on the identity privacy of recipients in group-oriented public key encryptions and proposes practical solutions.

Let us consider the following scenario. Alice wants to send a secret message to Bob who is a department manager of a company. For some security reasons and the purpose to protect the managers from dealing with junk messages, the company gateway system does not allow the message in unless it is for a department manager. However, Bob may not want to let the gateway know he is the actual recipient. By knowing only the public information of managers of the company, the gateway system has to determine whether allowing the encrypted message in without being able to identify the actual recipient. Furthermore, in case of dispute, we may hope a trusted third party to trace the actual recipient.

There may exist other applications where the recipients' anonymity is required. In a fair exchange of secrets in the trusted third party model, the two parties who want to exchange their secrets may not want to let the third party know their identities. When one downloads priced

* We noted that Aggelos Kiayias and Yiannis Tsiounis and Moti Yung recently presented an independent paper Group Encryption which achieved the same goals with an different implementation at: http://eprint.iacr.org/2007/015.pdf. Their work was submitted to eprint on 12 Jan 2007 by the record of the database and we accessed it on 21 Jan 2007

data from a server, the user may not want to leak its identity to the database server. In an identity escrow, the user may want to protect its privacy. It can diced its identity into pieces and send them to multiple trusted authorities and any t ones of the authorities can recover its identity. But no one knows which ones of the many potential authorities have the identity shares. Furthermore, in the last scenario, the recipients' anonymity also improves the privacy of the sender.

There are some related notions on the anonymity of users in the context of signatures. Group signatures, introduced by Chaum and van Heyst [9,5,8], provide signers' anonymity. Any group member can sign messages on behalf of the group, but the resulting signatures keep the identity of signer secret. In the standard definition, there is a third party that can open the signature, or undo its anonymity if required. A ring signature introduction in [17] is an alternative mean to achieve anonymity for ad hoc groups *without* any trusted manager. It is used to convince any third party that *at least* one member in an ad hoc group has indeed issued the signature on behalf of the group. Since there is no any trusted group manager, the anonymity in ring signatures cannot be revoked.

In the context of public key based encryptions, recently, Bellare *et al.* [3] presented a notion of key privacy in public-key encryption schemes. However, the setting, goal and model in this notion are different from ours. They studied the setting of asymmetric encryptions to capture a security property for public-key-based encryption schemes that an attacker cannot determine the public keys that were used to generate the ciphertexts that it sees. Notice that the attacker cannot verify whether the ciphertexts are valid or really for some of the potential recipients, and no trusted party can trace the actual recipient. They use the classic chosen plaintext attack (CPA) and chosen ciphertext attack (CCA) to model the adversary in their notion. Their goal is to find public key encryption schemes with a special property referred to as *recipient anonymity.* They showed that the existing well-known schemes such as the ElGamal encryption [11], the Cramer-Shoup encryption [10] and the RSA-OAEP [7,18] provide such a recipient anonymity with or without some trivial modifications.

In [16,15], a similar notion of custodian-hiding verifiable encryption schemes was presented. In their notion, a sender can verifiably encrypt a message under a chosen public key from a public key list but the actual recipient is anonymous. Since there is no group manager to administrate the potential recipients, the notion is only applicable to ad hoc applications and hence each ciphertext has to contain the public key list of potential recipients. Their instantiations suffers from a linear cost in both communication and computation in addition to the public key list in each ciphertext. Furthermore, in case of dispute introduced by a ciphertext in their notion, no group manager can revoke the anonymity of the receiver.

In this paper, we formalize the notion of group decryptions. It can be viewed as an analog of group signatures in the context of public key based encryptions. In this notion, a sender can encrypt a committed message to any intended member of a group managed by a group manager while *the recipient of the ciphertext keeps anonymous.* The sender can convince a verifier this fact without leaking the plaintext or the identity of the recipient. If necessary, *the group manger can verifiably open the identity of the recipient.* We propose the first anonymous group decryption scheme from pairing groups secure in the random oracle model[6]. The overhead in both computation and communication is independent of the group size and the scheme is practical. We also present several efficient sub-protocols such as commitment schemes to commit a group element in pairing groups and the corresponding zero-knowledge proof protocols. These sub-protocols are of independent interest and may find other applications.

The rest of the paper is organized as follows. The next section formalize the notion of group decryptions. In section 3, we review the underlying computational assumptions. Section 4 presents the building blocks. We propose our group decryption scheme in Section 5, followed several applications in Section 6. Section 7 concludes the paper.

2 Modeling Group Decryption

In this section, we formalize the notion of group decryption. It allows a sender to verifiably encrypt a committed message to any group member while the actual recipient is anonymous. In case of dispute, the anonymity can be revoked by the group manager.

2.1 Group decryption algorithms

An group decryption scheme consists of the following procedures.

- ParaGen: It is a polynomial time algorithm which on input a security λ, outputs the system parameters π.
- GKeyGen: It is a polynomial time algorithm which on input the system parameters π, outputs the group public and secret key pair (gpk, gsk).
- UKeyGen: It is a polynomial time algorithm which on input the system parameters π, outputs a user's public and secret key pair (upk, usk).
- Join: It is a polynomial time interactive algorithm between a user $\mathcal{U}$ who wants to join a group and the group manager $\mathcal{GM}$. The user has input usk while the group manager has input gsk. The common input is the (π, gpk, upk). The user has output (mpk, msk) which is the public and secret key pair of $\mathcal{U}$ as a legitimate group member. The group manager has output an updated local database which includes a tracing trapdoor $T_{\mathcal{U}}$ corresponding to group member $\mathcal{U}$. The tracing trapdoors forms a tracing list L_T secretly maintained by the group manager. All the legitimate group members' public keys mpk comprise of a public key list L_{pk}.
- Encrypt: It is a polynomial time algorithm which on input a secret message m in the message space, one of the group members's public key mpk in the public key list and the system parameters π, outputs a ciphertext c in the ciphertext space.
- EnVerify: It is a polynomial time algorithm which on input a ciphertext c, the system parameters π, the group public key gpk and the public key list of the group members, outputs a bit 1 or 0 to represent that the ciphertext is valid or not.
- Decryption: It is a polynomial time algorithm which on input a valid ciphertext c, the system parameters π, the group member $\mathcal{U}$'s public key mpk and secret key msk, outputs a message m in the message space.
- Trace: It is a polynomial time algorithm which on input a valid ciphertext c, the system parameters π, the group key pair (gpk, gsk), the public key list of the group members and its local tracing list, outputs an *error* message or the public key mpk of the recipient which represents the recipient's identity.
- TrProof: It is a polynomial time interactive algorithm between the group manager $\mathcal{GM}$ and a verifier. The group manager has input $(\pi, gpk, gsk, T_{\mathcal{U}}, L_{pk}, L_T)$ while the verifier has input (π, gpk, L_{pk}). After the interactive algorithm is run, the verifier outputs a bit 1 or 0 to represent that the Trace procedure has been correctly run or not while the group manager has no output.

A group decryption scheme is said to be correct if all the parties follow the scheme honestly, the EnVerify algorithm outputs 1, the Decryption algorithm outputs the correct message and the verifier in the TrProof procedure outputs 1.

2.2 Adversary model in group decryption

We model the adversaries in group decryption schemes with the following oracles to which the adversaries can query. These oracles are maintained by a challenger.

- UKeyGen Oracle. For the i-th ($i > 0$) query to this oracle, the adversary queries this oracle with an integer i. The challenger responds with the i-th user's public key upk_i but keeps the corresponding secret key usk_i. The challenger mains a counter n to records the query times and updates $n = i$.
- Join Oracle. The adversary queries this oracle with upk_i which is an output of the UKeyGen Oracle. The challenger runs the Join procedure for (upk_i, usk_i). The transcript of this procedure and the corresponding group member public key mpk_i are sent to the adversary. The challenger updates the corresponding tracing list as the real scheme.
- Corruption Oracle. The adversary queries with mpk_i and obtains the corresponding secret key msk_i if mpk_i is in the group member public key list.
- Encryption Oracle. The adversary queries this oracle with (m, mpk_i), where m is a message in the message space and mpk_i is in the group member public key list. The challenger responds the corresponding ciphertext c.
- Decryption Oracle. The adversary queries this oracle with a valid ciphertext. The challenger responds with the corresponding message.
- Trace Oracle. The adversary queries this oracle with a valid ciphertext. The challenger responds with a public key which represents the identity of the true recipient.
- TrProof Oralce. The adversary queries this oracle with a valid ciphertext and the identity of the traced recipient. The challenger responds with a proof to show that the ciphertext is sent the traced recipient.

2.3 Security definitions of group decryptions

The security of group decryption schemes includes three aspects, i.e., the semantic security against chosen-ciphertext attacks, the anonymity and traceability.

Let us first consider semantic security against chosen-chiphertext attacks. It is defined by the following game between a challenger $\mathcal{CH}$ and an adversary $\mathcal{A}$.

Setup: $\mathcal{CH}$ runs ParaGen and GkeyGen algorithms to generate the system parameters π and the group public and secret key pair (gpk, gsk). (π, gpk) are sent to the attacker $\mathcal{A}$. $\mathcal{CH}$ also maintains a counter and three lists L_U, L_M, L_T to recorder the users, the group members, and the tracing trapdoors.

Phase 1: $\mathcal{A}$ can adaptively make all the oracles defined above.

Challenge: $\mathcal{A}$ chooses a tuple (m_0, m_1, mpk_i), where m_0, m_1 are in the message space and $mpk_i \in L_M$ was not queried to the Corruption oracle. $\mathcal{CH}$ randomly selects a bit $b \in \{0, 1\}$. outputs the challenge ciphertext c^* =Encrypt(π, mpk_i, m_b). $\mathcal{CH}$ sends c^* to $\mathcal{A}$.

Phase 2: $\mathcal{A}$ may make another sequence of queries as in Phase 1 except that the Corruption oracle cannot be queried on mpk_i and c^* cannot be queried to Decrypt oracle.

Output: Finally $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{A}$ wins if $b' = b$.

Definition 1. *We say that a group decryption scheme is semantically secure against chosen ciphertext attacks if no polynomially bounded adversary has advantage that is non-negligibly greater than 1/2 of winning in the above game.*

Anonymity is defined by the following game between a challenger $\mathcal{CH}$ and an adversary $\mathcal{A}$.

Setup: It is the same as the semantic security game.
Phase 1: $\mathcal{A}$ can adaptively make all the oracles defined above.
Challenge: $\mathcal{A}$ chooses a tuple $(m, mpk_{i_0}, mpk_{i_1})$, where $mpk_{i_0}, mpk_{i_1} \in L_M$ were not queried to the Corruption oracle and m is in the message space. $\mathcal{CH}$ randomly selects a bit $b \in \{0,1\}$. outputs the challenge ciphertext $c^* = \mathsf{Encrypt}(\pi, mpk_{i_b}, m)$. $\mathcal{CH}$ sends c^* to $\mathcal{A}$.
Phase 2: $\mathcal{A}$ may make another sequence of queries as in Phase 1 except that the Corruption oracle cannot be queried on mpk_{i_0}, mpk_{i_1} and c^* cannot be queried to the Decrypt oracle.
Output: Finally $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. $\mathcal{A}$ wins if $b' = b$.

Definition 2. *We say that a group decryption scheme is anonymous if no polynomially bounded adversary has advantage that is non-negligibly greater than 1/2 of winning in the above game.*

A group decryption scheme should allow to revoke the identity of the recipient's identity in case of dispute. The traceability of a group decryption scheme is defined by the following game between a challenger $\mathcal{CH}$ and an adversary $\mathcal{A}$.

Setup: It is the same as the semantic security game.
Probe Phase: $\mathcal{A}$ can adaptively make queries to all the oracles defined above.
Output: $\mathcal{A}$ outputs a valid ciphertext c^*. $\mathcal{A}$ wins if the Trace algorithm outputs an error message or a string which is not the identity of the true recipient.

Definition 3. *We say that a group decryption scheme is traceable if no polynomially bounded adversary has negligible probability to win the above game.*

3 Mathematical Aspects

3.1 Bilinear pairings

We review some general concepts of pairing groups [4,13,19]. Let `PairingGen` be an algorithm that, on input a security parameter 1^λ, outputs a tuple $\Upsilon = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, g_1, g_2, e)$, where $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ have the same prime order p. $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ is an efficient bilinear map if the following conditions hold:

1. **Non-degeneration:** $e(g_1, g_2) \neq 1$;
2. **Bilinearity:** For all $h_1 \in \mathbb{G}_1, h_2 \in \mathbb{G}_2$ and $u, v \in \mathbb{Z}$, $e(h_1^u, h_2^v) = e(h_1, h_2)^{uv}$

From [12], there are three types of pairing groups:

1. $\mathbb{G}_2 = \mathbb{G}_1$ and accordingly denote $\Upsilon = (p, \mathbb{G}, \mathbb{G}_3, g, e) \leftarrow \texttt{PairingGen}(1^\lambda)$ for simplicity;

2. $\mathbb{G}_2 \neq \mathbb{G}_1$ in which there is an *efficient distortion map* $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ but there is no *efficient* distortion map $\varphi : \mathbb{G}_1 \to \mathbb{G}_2$, where the distortion map satisfies $\psi(g_2^u) = \psi(g_2)^u \in \mathbb{G}_1$ for any $u \in Z_p$;
3. $\mathbb{G}_2 \neq \mathbb{G}_1$ but there is no *efficient* distortion map $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ or $\varphi : \mathbb{G}_1 \to \mathbb{G}_2$.

If $\mathbb{G}_2 \neq \mathbb{G}_1$ and there are efficient homomorphisms $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ and $\varphi : \mathbb{G}_1 \to \mathbb{G}_2$, it can be re-interpreted as Type 1. The Type 1 case is implemented using supersingular curves. The curves of Type 2 are ordinary and the homomorphism from $\mathbb{G}_2 \to \mathbb{G}_1$ is the trace map. The curves of Type 3 are ordinary and $\mathbb{G}_2$ is typically taken to be the kernel of the trace map.

3.2 Computational assumptions

Suppose that $\Upsilon = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, g_1, g_2, e) \leftarrow \texttt{PairingGen}(1^\lambda)$ are SXDH pairing groups, where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_3$ are public. Our proposals are based on the following assumptions about pairing groups. We recall these assumptions used by previous works.

Assumption 1 (Inverse of Bilinear Pairing (IBP) Assumption) *Let* $\Upsilon = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_2, g_1, g_2, e) \leftarrow \texttt{PairingGen}(1^\lambda)$. *Given random values* $Y, h_2 \in \mathbb{G}_2$, *for any probabilistic polynomial time (PPT) adversary* $\mathcal{A}$, *the probability to compute* $X \in \mathbb{G}_1$ *satisfying* $e(X, g_2) = e(Y, h_2)$ *is negligible in* λ.

The IBP assumption is a weak assumption. It is indeed weaker than the co-CDH assumption [4]. An adversary $\mathcal{A}$ breaking the IBP assumption can be efficiently converted into an adversary $\mathcal{B}$ to break the co-CDH assumption. The transformation is trivial: Given a co-CDH challenge (g_1, g_2, g_1^u, g_2^v), $\mathcal{B}$ computes $A = e(g_1^u, g_2^v) = e(g_1, g_2)^{uv}$ and queries $\mathcal{A}$ with (A, g_1, g_2). $\mathcal{B}$ straightforward uses $\mathcal{A}$'s reply $X = g_1^{uv}$ to answer the co-CDH challenge. Similarly, if $\mathbb{G}_1 = \mathbb{G}_2$, the IBP assumption is implied by the classic CDH assumption in $\mathbb{G} = \mathbb{G}_1 = \mathbb{G}_2$.

The IBP assumption is an analog of the RSA assumption in the pairing group settings. That is, given a image of a one-way function, it is hard to find the pre-image. We will use a strong version of the IBP assumption which can be viewed as an analog of the strong RSA assumption in the pairing group settings. This assumption holds only in the SXDH pairing groups.

Assumption 2 (Strong Inverse of Bilinear Pairing (SIBP) Assumption) *Let* $\Upsilon = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_2, g_1, g_2, e) \leftarrow \texttt{PairingGen}(1^\lambda)$ *be pairing groups of Type 3. Given random values* $h_2 \in \mathbb{G}_3$, $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, *for any PPT adversary* $\mathcal{A}$, *the probability to compute a pair* $(X, Y) \in \mathbb{G}_1$ *satisfying* $e(X, g_2) = e(Y, h_2)$ *is negligible in* λ.

In pairing groups of type 3, the conventional DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$. Hence, such pairing groups are also called SXDH (symmetric external Diffie-Hellman) pairing groups [1]. In [1], Ateniese *et al.* exploited such pairing groups to built their practical group signatures without random oracles.

Assumption 3 (Symmetric External Diffie-Hellman (SXDH) Assumption) *Let* $\Upsilon = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_2, g_1, g_2, e) \leftarrow \texttt{PairingGen}(1^\lambda)$ *be pairing groups of Type 3. The SXDH assumption states that the standard DDH assumption holds in both* $\mathbb{G}_1$ *and* $\mathbb{G}_2$.

The LRSW assumption is a discrete-logarithm assumption originally introduced by Lysyanskaya *et al.* [14] and used in many subsequent works. Recently, a stronger from of the LRSW assumption, called Strong LRSW, was introduced by Ateniese et al. [2]. Strong LRSW only holds in pairing groups of type 3.

Assumption 4 (Strong LRSW Assumption) *For SXDH pairing groups* $\Upsilon = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_2, g_1, g_2, e) \leftarrow \texttt{PairingGen}(1^\lambda)$, *Let* $X, Y \in \mathbb{G}_2$ *be chosen at random, and* $O_{X,Y}(\cdot)$ *be an oracle* *that takes as input a value* $v \in \mathbb{Z}_p^*$, *and outputs an LRSW-tuple* (a, a^x, a^{y+vxy}) *for a random* $a \in \mathbb{G}_1$. *Then for any PPT adversary* $\mathcal{A}^{(\cdot)}$ *and all* $u \in \mathbb{Z}_p^*$,

$$\Pr\left[\begin{array}{l|l} x \leftarrow \mathbb{Z}_p, y \leftarrow \mathbb{Z}_p & (a_1, a_2, a_3, a_4, a_5) \leftarrow \mathcal{A}^{O_{X,Y}(\cdot)}(g_1, g_2, X, Y) \wedge a_1 \in \mathbb{G}_1 \\ X = g_2^x, Y = g_2^y & \wedge a_2 = a_1^u \wedge a_3 = a_1^x \wedge a_4 = a_1^{ux} \wedge a_5 = a_1^{y+uxy} \wedge u \notin \mathbb{Q} \end{array}\right] \leq \frac{1}{poly(\lambda)}$$

where $\mathbb{Q}$ *is the set of queries* $\mathcal{A}$ *makes to* $O_{X,Y}(\cdot)$.

4 Building Blocks

In this section, as building blocks of our anonymous group decryptions, we present new commitment schemes to commit group element in pairings. The commitment schemes works similarly to the well-known discrete and Peterson commitments. Then we propose zero-knowledge proof protocols of knowledge of the committed values.

4.1 Commitment

A commitment scheme consists of four efficient algorithms: $\mathcal{C} = (\texttt{ParaGen}, \texttt{Com}, \texttt{Open}, \texttt{Ver})$. The generation algorithm $\texttt{ParaGen}(1^k)$, where k is the security parameter, outputs a public commitment key pk (possibly empty, but usually consisting of public parameters for the commitment scheme). Given a message m from the associated message space $\mathcal{M}$ (e.g., $\{0,1\}^k$, although we will mainly concentrate on bit commitments), $\texttt{Com}_{pk}(m; r)$ (computed using the public key pk and additional randomness r) produces a commitment string c for the message m. We will sometimes omit r and write $c \leftarrow \texttt{Com}_{pk}(m)$. Similarly, the opening algorithm $\texttt{Open}_{pk}(m; r)$ (which is supposed to be run using the same value r as the commitment algorithm) produces a decommitment value d for c. Finally, the verification algorithm $Ver_{pk}(m, c, d)$ accepts (i.e., outputs 1) if it thinks the pair (c, d) is a valid commitment/decommitment pair for m. We require that for all $m \in \mathcal{M}$, $\texttt{Ver}_{pk}(m, \texttt{Com}_{pk}(m; r), \texttt{Open}_{pk}(m; r)) = 1$ holds with all but negligible probability.

We remark that without loss of generality we could have assumed that the opening algorithm simply outputs its randomness r as the decommitment, and the verification algorithm simply checks if $c = \texttt{Com}_{pk}(m; r)$. However, we will find our more general notation more convenient for our purposes. When clear form the context, we will sometimes omit pk from our notation. Regular commitment schemes have two security properties:

- **Hiding:** No PPT adversary (who knows pk) can distinguish the commitments to any two message of its choice: $Com_{pk}(m_1), Com_{pk}(m_2)$. That is, $\texttt{Com}_{pk}(m)$ reveals "no information" about m.
- **Binding:** Having the knowledge of pk, it is computationally hard for the PPT adversary $\mathcal{A}$ to come up with c, m, d, m', d' such that (c, d) and (c, d') are valid commitment pairs for m and m', but $m \neq m'$ (such a tuple is said to cause a collision). That is, $\mathcal{A}$ cannot find a value c which it can open in two different ways.

4.2 *Σ*-Protocols

Let $\mathcal{R} = (x, w)$ be some NP-relation (i.e., it is efficiently testable to see if $(x, w) \in \mathcal{R}$ and $|w| \leq poly(|x|)$). We usually call x the input, and w the witness (for x). Consider a three move protocol run between a PPT prover $\mathbf{P}$, with input $(x, w) \in \mathcal{R}$, and a PPT verifier $\mathbf{V}$ with input

x, of the following form. **P** chooses a random string r_p, computes $a = \mathtt{Start}(x, w; r_p)$, and sends a to **V**. **V** then chooses a random string c (called "challenge") from some appropriate domain E (see below) and sends it to **P**. Finally, **P** responds with $z = \mathtt{Finish}(x, w, c; r_p)$. The verifier **V** then computes and returns a bit $b = \mathtt{Check}(x, a, c, z)$. We require that `Start`, `Finish`, and `Check` be polynomial-time algorithms, and that $|c| \leq poly(|x|)$. Such a protocol (given by procedures `Start`, `Finish`, and `Check`) is called a Σ-Protocol for $\mathcal{R}$ if it satisfies the following properties, called completeness, special soundness, and special honest-verifier zero-knowledge:

- **Completeness:** If $(x, w) \in \mathcal{R}$ then the verifier outputs $b = 1$ (with all but negligible probability).
- **Special Soundness:** There exists a PPT algorithm Extract, called the (knowledge) extractor, such that it is computationally infeasible to produce an input tuple (x, a, c, z, c', z') such that $e \neq e'$ both lie in the proper "challenge" domain, $\mathtt{Check}(x, a, c, z) = \mathtt{Check}(x, a, c', z') = 1$, and yet $\mathtt{Extract}(x, a, c, z, c', z')$ fails to output a witness w such that $(x, w) \in \mathcal{R}$. Intuitively, if some prover can correctly respond to two different challenges e and e0 on the same first flow a, then the prover must "know" a correct witness w for x (in particular, x has a witness).
- **Special HVZK:** There exists a PPT algorithm `Simul`, called the simulator, such that for any $(x, w) \in \mathcal{R}$ and for any fixed challenge c, the following two distributions are computationally indistinguishable. The first distribution (x, a, c, z) is obtained by running an honest prover **P** (with some fresh randomness r_p) against a verifier whose challenge is fixed to c. The second distribution (x, a, c, z) is obtained by computing the output $(a, z) \leftarrow \mathtt{Simul}(x, c)$ (with fresh randomness r_s). Intuitively, this says that for any a-priori fixed challenge c, it is possible to produce a protocol transcript computationally indistinguishable from an actual run with the prover (who knows w).

Using the known Fiat-Shamir transformation, the above knowledge proof protocols can be converted into digital signatures. They can be proven secure in the random oracle model due to the fork lemma.

4.3 Prove knowledge of committed element in pairings Groups

This commitment scheme is similar to the discrete logarithm commitment scheme. Let (p, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$, g_1, g_2, e) $\leftarrow \mathtt{PairingGen}(1^\lambda)$. The public commitment key is $pk = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, g_1, g_2, e)$.

To commit a group element $x \in \mathbb{G}_1$, one computes the commitment

$$A = e(x, g_2).$$

To open the commitment A, the committer shows $m \in \mathbb{G}_1$ to the verifier. The verifier checks that $x \overset{?}{\in} \mathbb{G}_1$ and $e(x, g_2) \overset{?}{=} A$. The verifier outputs 1 if both checks hold; otherwise outputs 0. Clearly, the commitment scheme is computationally hiding and binding.

Similar to the knowledge proof of discrete logarithm, we present a knowledge proof of the knowledge of the committed m in A, and denote the protocol by

$$PK\{x | A = e(x, g_2)\}.$$

The 3-move protocol is as follows.

Step 1 The prover (i.e., the committer) randomly select $r \in \mathbb{G}_1$ and sends $B = e(r, g_2)$ to the prover.
Step 2 The verifier challenges the prover with $c \in \mathbb{Z}_p^*$.
Step 3 The prover responses with $s = rx^c$.
Step 4 The verifier checks that $s \overset{?}{\in} \mathbb{G}_1$ and $e(s, g_2) \overset{?}{=} BA^c$.The verifier outputs 1 if both checks hold; otherwise outputs 0.

The completeness of the above protocol is obvious. Now we prove the soundness and zero-knowledge.

Theorem 1. *The above knowledge proof protocol $PK\{x|A = e(m, g_2)$ is Σ-protocol.*

Proof. We first show the *special soundness* by construction of an efficient knowledge extractor if the malicious can respond to different challenges $c \neq c'$ on the same first flow. Let $s \neq s' \in \mathbb{G}_1$ be the two different responses. From the verification equations, it holds that $e(s, g_2) = BA^c$ and $e(s', g_2) = BA^{c'}$. Assume that $r \in \mathbb{G}_1$ satisfies that $e(r, g_2) = B$. Then we have that $s = rx^c$ and $s' = rx^{c'}$. It follows that $sx^{-c} = s'x^{-c'}$. Hence, $x^{c-c'} = s/s'$. Since $c - c' = 0$, $x = (s/s')^{\frac{1}{c-c'}}$. The knowledge is extracted.

Then we prove the *special HVZK*, i.e., there exist an efficient simulator `sim`, which on input public parameters *pk* and A, outputs a simulated transcript (B', c', s') indistinguishable from the output (B, c, s) of a real run of the protocol. The simulation works as follows. Randomly select $s' \leftarrow \mathbb{G}_1, c' \leftarrow \mathbb{Z}_p^*$. Compute $B' = e(s', g_2)/A^{c'}$. Clearly, $e(s', g_2) = B'A^{c'}$, and (B', c', s') has identical distribution as the output (B, c, s) of a real run of the protocol. This completes the proof.

4.4 Prove equality of committed elements in pairing groups

In this subsection, we present a zero-knowledge proof of the equality of committed group elements. Let the prover committed two values $A = e(x, g_2)$ and $B = e(x, h_2)$, where h_2 is an independent generator of $\mathbb{G}_2$. The prover can prove it to an honest verifier in a zero-knowledge meaner. We denote the protocol by

$$PK\{x|A = e(x, g_2) \wedge B = e(x, h_2)\}.$$

The protocol works as follows.

- **Step 1** The prover randomly selects $r \leftarrow \mathbb{G}_1$ and sends $D = e(r, g_2)$ and $E = e(r, h_2)$ to the verifier.
- **Step 2** The verifier challenges the prover with $c \leftarrow \mathbb{Z}_p^*$.
- **Step 3** The prover responses with $s = rm^c$.
- **Step 4** The verifier checks that $s \overset{?}{\in} \mathbb{G}_1$, $e(s, g_2) \overset{?}{=} DA^c$ and $e(s, g_2) \overset{?}{=} EB^c$. The verifier outputs 1 if all checks hold; otherwise, output 0.

The completeness of the protocol is straightforward. For the security, we have the following result.

Theorem 2. *The above protocol $PK\{x|A = e(x, g_2) \wedge B = e(x, h_2)\}$ is a Σ-protocol.*

Proof. We omit is as it is very similar to the previous theorem.

4.5 Prove knowledge of Pedersen commitment of element in pairing groups

The Pedersen commitment of discrete logarithms is a widely used commitment scheme. In this subsection, we present a Pedersen commitment of a group element in SXDH pairing groups. Let $\Upsilon = (p,\ \mathbb{G}_1,\ \mathbb{G}_2,\ \mathbb{G}_2,\ g_1,\ g_2,\ e) \leftarrow \texttt{PairingGen}(1^\lambda)$ be pairing groups of Type 3. Given random values $h_2 \in \mathbb{G}_2$, the Pedersen commitment of of a secret element $X \in \mathbb{G}_1$ is

$$A = e(X, g_2)e(R, h_2)$$

where r is randomly chosen from $\mathbb{G}_1$.

Similar to the classic Pedersen commitment, we argue that this commitment is unconditionally hiding and computationally binding. On the one hand, given $A \in \mathbb{G}_3$ and g_2, h_2 as independent generators of $\mathbb{G}_2$, for any $X' \in \mathbb{G}_1$, there exist an $R' \in \mathbb{G}_1$ such that $e(R', h_2) = A/e(X', g_2)$. Hence, it is unconditionally hiding. On the other hand, if the committer can output two pairings $(X, R) \neq (X', R')$ such that $A = e(X, g_2)e(R, h_2)$ and $A = e(X', g_2)e(R', h_2)$, then we have $e(x, g_2)/e(x', g_2) = e(r', h_2)/e(r, h_2)$. Hence, we can output a pair $(x/x', r'/r)$ satisfying $e(x/x', g_2) = e(r'/r, h_2)$. However, this is infeasible in SXDH pairing groups under the SIBP assumption. Hence the above commitment is computationally binding.

We provide a knowledge proof of the Pedersen commitment SXDH pairing groups. We denote the protocol by

$$PK\{X, R|A = e(X, g_2)e(R, h_2)\}.$$

The protocol run as follows.

- **Step 1** The prover randomly selects $V, W \leftarrow \mathbb{G}_1$ and sends $D = e(V, g_2)e(W, h_2)$ to the verifier.
- **Step 2** The verifier challenges the prover with $c \leftarrow \mathbb{Z}_p^*$.
- **Step 3** The prover responses with $S = VX^c$,$Z = WR^c$.
- **Step 4** The verifier checks that $S, Z \stackrel{?}{\in} \mathbb{G}_1$ and $e(S, g_2)e(Z, h_2) \stackrel{?}{=} DA^c$. The verifier outputs 1 if both checks hold; otherwise, output 0.

The completeness of the protocol is straightforward. For the security, we have the following result.

Theorem 3. *The above protocol* $PK\{x, r, \bar{r}|A = e(x, g_2)e(r, h_2)\}$ *is a* Σ*-protocol.*

Proof. We omit is as it is very similar to the previous theorem.

4.6 Prove knowledge of discrete logarithm of Pedersen commitment in pairing groups

Slightly modifying the above protocol, we achieve zero-knowledge proof protocol

$$PK\{X, R, x|A = e(X, g_2)e(R, h_2) \wedge X = g_1^x\},$$

where a prover proves the knowledge of $X, R \in \mathbb{G}_1$ and $x \leftarrow \mathbb{Z}_p$ satisfying $A = e(X, g_2)e(R, h_2)$ and $X = g_1^x$ without leakage of X, R or x. The modified protocol is as follows.

- **Step 1** The prover randomly selects $\gamma \leftarrow \mathbb{Z}_p, W \leftarrow \mathbb{G}_1$ and sends $D = e(g_1^\gamma, g_2)e(W, h_2)$ to the verifier.

– **Step 2** The verifier challenges the prover with $c \leftarrow \mathbb{Z}_p^*$.
– **Step 3** The prover responses with $\sigma = \gamma + cx$, $Z = WR^c$.
– **Step 4** The verifier finally checks that $\sigma \overset{?}{\in} \mathbb{Z}_p$, $Z \overset{?}{\in} \mathbb{G}_1$, $e(g_1^\sigma, g_2)e(Z, h_2) \overset{?}{=} DA^c$. The verifier outputs 1 if all checks hold; otherwise, output 0.

The completeness of the protocol is straightforward. For the security, we have the following result.

Theorem 4. *The above protocol* $PK\{X, R, x|A = e(X, g_2)e(R, h_2) \wedge X = g_1^x\}$ *is a* Σ*-protocol.*

We omit is as it is very similar to the previous theorem. If we view x as the identify or public key of the prover and α its private key, the above modified protocol may be useful for anonymous systems. The protocol may also be useful in other applications due to the homomorphic property about A, X, x.

5 Proposed Group Decryption Scheme

In this section, we propose a anonymous group decryption scheme following the definition.

– **ParaGen:** $\Upsilon = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, g_1, g_2, e) \leftarrow \texttt{PairingGen}(1^\lambda)$. $\mathcal{H}(\cdot) : \{0,1\}^* \rightarrow \mathbb{Z}_p$ is cryptographic hash function. Let h_2 be an independent generator of $\mathbb{G}_2$. The globe parameters are $\pi = \{\Upsilon, \mathcal{H}, h_2\}$.
– **GKeyGen:** Randomly select $x, y \leftarrow \mathbb{Z}_p^*$. Compute $X = g_2^x, Y = g_2^y$. The public and secret key pair of the group manager is

$$gpk = (X, Y), gsk = (x, y).$$

– **UKeyGen:** Choose at random $u \leftarrow \mathbb{Z}_p^*$. Compute $U = e(g_1, g_2)^u$. The public and secret key pair of the user is

$$upk = U, usk = u.$$

– **Join:** A user $\mathcal{U}$ can join a group and become a group member via the following protocol with the group manager $\mathcal{GM}$.
  1. $\mathcal{U}$ sends $E = g_1^u, T = g_2^u$ to $\mathcal{GM}$ via a confidential channel and proves the knowledge of decryption key: $\varrho = ZK\{u|E = g_1^u\}$.
  2. $\mathcal{GM}$ checks the validity of ϱ and $e(E, g_2) = e(g_1, T) = U$. If both checks are successful and T has been not in its local database, $\mathcal{GM}$ adds (T, U) in its local database, and sends $S = (a_1, a_2, a_3, a_4, a_5)$ to $\mathcal{U}$ as its group certificate corresponding to U, where

$$a_1 = g_1^\gamma, a_2 = E^\gamma, a_3 = a_1^x, a_4 = a_2^x, a_5 = (a_1 a_4)^y$$

     for a randomly chosen value $\gamma \leftarrow \mathbb{Z}_p^*$. Else, $\mathcal{GM}$ aborts the Join protocol.
  3. The user checks the validity of the group certificate $S = (a_1, a_2, a_3, a_4)$:

$$e(a_1, T) = e(a_2, g_2), e(a_3, g_2) = e(a_1, X), e(a_4, g_2) = e(a_2, X), e(a_5, g_2) = e(a_1 a_4, Y).$$

     If any equation does not hold, the Join protocol fails. Else, the user computes a knowledge signature

$$\sigma = KS\{u, T|e(a_1, T) = e(a_2, g_2) \wedge e(g_1, T) = U \wedge a_1^u = a_2\}(gpk||upk||S)$$

on a message of the group public key, the user's own public key and the corresponding certificate. The user $\mathcal{U}$ who has become a group member has a member public key and secret key pair

$$mpk = \{s, U, \sigma\}, msk = u.$$

- **Encryption:** Let a sender want to send a secret message $m \in G_1$ to a group member $\mathcal{U}$. It can verifiably send it to $\mathcal{U}$ without leaking the identity of $\mathcal{U}$ as follows.
  1. Membership check: The sender verifies the validity of S and σ. If any check fails, the sender aborts.
  2. Message commitment: For $m \in \mathbb{G}_1$, commit the secret message m as follows:

$$\delta \leftarrow \mathbb{G}_1, c_0 = e(m, g_2)e(\delta, h_2).$$

  3. Key Re-randomization: Randomly select $r \leftarrow \mathbb{Z}_p^*$ and re-randomize the group certificate of U by computing:

$$c_1 = a_1^r, c_2 = a_2^r, c_3 = a_3^r, c_4 = a_4^r, c_5 = a_5^r.$$

  4. Message encryption: Randomly choose $s \leftarrow \mathbb{Z}_p^*$, compute

$$c_6 = a_1^s, c_7 = a^{-1}b_2^s.$$

  5. Encryption proof: Prove that (c_0, c_6, c_7) has been correctly generated by compute the knowledge signature

$$c_8 = KS\{M, s|e(c_7, g_2)c_0 = e(M, g_2)e(\delta, h_2)\wedge c_6 = b_1^s \wedge M = b_2^s\}(c_0||c_1||c_2||c_3||c_4||c_5||c_6||c_7),$$

  which is equivalent the following knowledge signature:

$$c_8' = KS\{m, s|c_0 = e(m, g_2)e(\delta, h_2) \wedge c_6 = b_1^s \wedge c_7 = m^{-1}b_2^s\}(c_0||c_1||c_2||c_3||c_4||c_5||c_6||c_7).$$

  Output $c = (c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8)$ as the resulting ciphertext of message m to the anonymous group member $\mathcal{U}$.
- **Encryption Verification** Any verifier can verify the validity of the ciphertext as follows:
  1. Check that

$$e(c_1, T) = e(c_2, g_2), e(c_3, g_2) = e(c_1, X), e(c_4, g_2) = e(c_2, X), e(c_5, g_2) = e(c_1 c_4, Y).$$

  2. Check that c_8 is a valid knowledge signature as defined.

  If any check fails, the ciphertext is rejected. Else it is accepted.
- **Decryption:** The group member decrypts a ciphertext c as follows:
  1. Check that $c_2 = c_1^u$;
  2. Check that is a valid knowledge signature as defined;
  3. Check that

$$e(c_1, T) = e(c_2, g_2), e(c_3, g_2) = e(c_1, X), e(c_4, g_2) = e(c_2, X), e(c_5, g_2) = e(c_1 c_4, Y).$$

  If any check fails, the group member $\mathcal{U}$ aborts the Decryption procedure. Else, it outputs

$$m = c_6^u / c_7.$$

– **Receiver Tracing:** The group manager can trace the recipient as follows. It check whether there exists $(T,\mathcal{U})$ in its local database such that

$$e(c_1, T) = e(c_2, g_2).$$

If so, output U. Else output an error message.
– **Receiver-Tracing Proof:** The group manager can prove to a verifier that it has correctly traced the recipient with the following zero-knowledge proof:

$$ZK\{T|e(c_1, T) = e(c_2, g_2) \wedge e(g_1, T) = U\}.$$

The correctness of the scheme follows from a straightforward verification. Now we consider the security of the scheme.

Theorem 5. *The above group decryption scheme is semantically secure against chosen ciphertext attacks in the random oracle model under the DDH assumption and the Strong LRSW assumption in SXDH pairing groups.*

Proof. We prove that a successful attacker in the semantical security game of our scheme can be used as a subroutine to break the DDH assumption in $\mathbb{G}_1$.

Assume that we are given a DDH challenge $g_1, g_1^\alpha, g_1^\beta, g_1^\delta \in \mathbb{G}_1$, where $\mathbb{G}_1$ is from pairing groups $\Upsilon = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, g_1, g_2, e) \leftarrow \texttt{PairingGen}(1^\lambda)$. We are required to answer whether $\delta = \alpha\beta$ or not. We first use a DDH challenge to simulate the oracles that the attacker may query and then use the attacker's reply to answer the DDH challenge.

Setup. We randomly choose $h_2 \in \mathbb{G}_2$. The globe parameters are $\pi = \{\Upsilon, \mathcal{H}, h_2\}$, where the hash function $\mathcal{H}(\cdot) : \{0,1\}^* \to \mathbb{Z}_p$ is modeled as a random oracle. It is simulated in the standard way. That is, we maintain an $\mathcal{H}$-list and for any query ξ, if ξ is not in the list, we reply with a random number $c \in \mathbb{Z}_p$ and add (ξ, c) to the $\mathcal{H}$-list. If the ξ has been in the list, we forward the precious reply to maintain consistent. We also randomly select $x, y \leftarrow \mathbb{Z}_p^*$ and compute $X = g_2^x, Y = g_2^y$. The public and secret key pair of the group manager is $gpk = (X, Y), gsk = (x, y)$. π and gpk are sent to the attacker.

UkeyGen Oracle. Let the system contain at most $n = poly(\lambda)$ users. We randomly choose an integer $1 \le i_0 \le n$. For the attacker's query $i \ne i_0$, we behave as the real scheme and send the corresponding public key to the attacker. If $i = i_0$, we send $e(g_1^\alpha, g_2)$ as the i_0-th user's public key.

Join Oracle. For the attacker's query $upk \ne g_1^\alpha$ in the user public key list L_U, we behave as the real scheme. For query g_1^α, we randomly choose two strings in the ciphertext space to respectively simulate the ciphertexts of g_1^α and g_2^α. We randomly select $(c, s) \in \mathbb{Z}_p \times \mathbb{Z}_p^*$ to simulate the (non-interactive) knowledge proof $ZK\{\alpha|E = g_1^\alpha\}$. Add $(g_1^s E^c, c)$ to the $\mathcal{H}$-list. Use (x, y), we can generate the group certificate $S = (a_1, a_2, a_3, a_4, a_5)$ for $E = g_1^\alpha$ as the real scheme. We can also similarly simulate the knowledge signature $\sigma = KS\{\alpha, T|e(a_1, T) = e(a_2, g_2) \wedge e(g_1, T) = U \wedge a_1^\alpha = a_2\}(gpk||E||S)$ in the random oracle model without knowing α, T.

Corruption Oracle. Whenever the attacker queries a group member's secret key msk, if mpk corresponds to g^α, we declare failure and aborts the protocol, denoted by a bad event *Failure 1*. Else we can correctly answer with the corresponding msk since we have generated such group member's public and secret key pair as the real scheme. Here, *Failure 1* happens with probability $\varepsilon_1 = \frac{n_c}{n}$, where n_c is the number of corrupted members.

Encryption Oracle. We do as the real scheme but keep a list to record all the ciphertext we have produced.

Decryption Oracle. When the attacker queries this oracle with $c = (c_1, \cdots, c_8)$, if it is not corresponding to the user public key g_1^α, we can reply with the corresponding message m as the real scheme. If it is corresponding to the user public key g_1^α but $(c_1, \cdots, c_7, c_8')$ is in the ciphertext list but $c_8 \neq c_8'$, we reply with the plaintext m corresponding to $(c_1, \cdots, c_7, c_8')$ if and only if the encryption verification on c holds. If $(c_1, \cdots, c_7)$ is not in the ciphertext list but the encryption verification on c holds, we run the attacker two times using the standard rewinding technique [**?**] to extract M, s. We reply with Mc_7^{-1} as the corresponding plaintext.

Trace Oracle. When the attacker queries this oracle with a valid ciphertext $c = (c_1, \cdots, c_8)$, we trace it with the tracing trapdoors in the tracing list. If all the trapdoors fail to trace the recipient, we output the group member corresponding to g_1^α as the recipient. Else we trace the recipient as the real scheme. This simulation fails if the ciphertext is not generated for any recipient in the group member list which implies that the attacker has successfully forge a group member certificate results into a solution to the Strong LRSW assumption. Hence, this *Failure 2* happens with negligible probability ε_3 assuming the Strong LRSW assumption.

TrProof Oracle. When the attacker queries this oracle with a valid ciphertext and the identity of traced recipient mpk, if mpk does not correspond to g_1^α, we reply as the real scheme. If mpk corresponds to g_1^α, we use the standard simulation of the zero-knowledge proof in the random oracle model to reply the attacker.

In the challenge phase, the attacker queries with a tuple (m_0, m_1, mpk), where m_0, m_1 are in the message space and $mpk = (a_1, a_2, a_3, a_4, a_5, upk, \sigma)$ is a valid group member's public key and has never been queried to the Corruption Oracle. If $upk \neq e(g_1^\alpha, g_2)$, we declare failure and denoted it by a bad event *Failure 3* which happens with probability $1 - \frac{1}{n}$. Else we randomly choose a bit $b \in \{0, 1\}$ and do the following.

- Commit the chosen message m_b with $c_0^* = e(m, g_2)e(\delta, h_2)$ for a random $\delta \leftarrow \mathbb{G}_1$.
- Compute $c_1^* = g_1^\gamma, c_2^* = (g_1^\alpha)^\gamma, c_3^* = a_1^x, c_4^* = a_2^x, c_5^* = (c_1^* c_4^*)^y$ for a randomly chosen value $\gamma \leftarrow \mathbb{Z}_p^*$.
- Randomly choose $s \leftarrow \mathbb{Z}_p^*$, compute $c_6^* = (g_1^\beta)^\gamma, c_7^* = m_b^{-1}(g_1^\delta)^\gamma$.
- Simulate the knowledge signature in the random oracle model:

$$c_8^* = KS\{M, s|e(c_7^*, g_2)c_0^* = e(M, g_2)e(\delta, h_2) \wedge c_6^* = (c_1^*)^s \wedge M = (c_2^*)^s\}(c_0^*||c_1^*||c_2^*||c_3^*||c_4^*||c_5^*||c_6^*||c_7^*).$$

- Output $c^* = (c_0^*, c_1^*, c_2^*, c_3^*, c_4^*, c_5^*, c_6^*, c_7^*, c_8^*)$ as the challenge ciphertext of message m_b to the group member mpk.

After receiving the challenge ciphertext c^*, the attacker can still query above oracles but mpk cannot be queried to the Corruption Oracle and c^* cannot be queried to the Decryption Oracle. We answer these queries as above.

Finally, the attacker will output its guess bit b'. We conclude that $(g_1, g_1^\alpha, g_1^\beta, g_1^\delta)$ is a DDH tuple in $\mathbb{G}_1$ if and only if $b' = b$. Note that c^* is valid ciphertext of m_b under the group member public key mpk if and only if $(g_1, g_1^\alpha, g_1^\beta, g_1^\delta)$ is a DDH tuple. We answer successfully whenever the attacker has a correct guess. Let the attacker win this semantical security game with probability ε. Hence, we win the DDH challenge with probability $(1 - \varepsilon_1)(1 - \varepsilon_2)(1 - \varepsilon_3)\varepsilon \approx (1 - \frac{n_c}{n})\frac{1}{n}\varepsilon$. This completes the proof.

Theorem 6. *The above group decryption scheme is anonymous in the random oracle model under the DDH assumption and the Strong LRSW assumption in SXDH pairing groups.*

Proof. We prove that a successful attacker in the anonymity game of our scheme can be used as a subroutine to break the DDH assumption in $\mathbb{G}_1$.

Assume that we are given a DDH challenge $g_1, g_1^\alpha, g_1^\beta, g_1^\delta \in \mathbb{G}_1$, where $\mathbb{G}_1$ is from pairing groups $\Upsilon = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, g_1, g_2, e) \leftarrow \texttt{PairingGen}(1^\lambda)$. We are required to answer whether $\delta = \alpha\beta$ or not. We first use a DDH challenge to simulate the oracles that the attacker may query and then use the attacker's reply to answer the DDH challenge.

The simulation is the same as the previous proof in the semantical security. In the challenge phase, the attacker queries with a tuple $(m, mpk_{i_0}, mpk_{i_1})$, where m is in the message space and $mpk_{i_b} = (a_{1,b}, a_{2,b}, a_{3,j}, upk_{i_b}, \sigma_{i_b})$ are valid group member's public keys for $b = 0, 1$ and have never been queried to the Corruption Oracle. If $upk_{i_b} \neq e(g_1^\alpha, g_2)$ for $b = 0, 1$, we declare failure and denoted it by a bad event which happens with probability $1 - \frac{2}{n}$. Else $upk_{i_b} = e(g_1^\alpha, g_2)$. We compute the challenge ciphertext as follows.

- Commit the chosen message m with $c_0^* = e(m, g_2)e(\delta, h_2)$ for a random $\delta \leftarrow \mathbb{G}_1$.
- Compute $c_1^* = g_1^\beta, c_2^* = g_1^\delta, c_3^* = c_1^x, c_4^* = c_2^x, c_5^* = (c_1c_4)^y$ for a randomly chosen value $\gamma \leftarrow \mathbb{Z}_p^*$.
- Randomly choose $s \leftarrow \mathbb{Z}_p^*$, compute $c_6^* = (c_1^*)^s, c_7^* = m^{-1}(c_2^*)^s$.
- Simulate the knowledge signature in the random oracle model using the standard simulating technique:

$$c_8^* = KS\{M, s | e(c_7^*, g_2)c_0^* = e(M, g_2)e(\delta, h_2) \wedge c_6^* = (c_1^*)^s \wedge M = (c_2^*)^s\}(c_0^*||c_1^*||c_2^*||c_3^*||c_4^*||c_5^*||c_6^*||c_7^*).$$

- Output $c^* = (c_0^*, c_1^*, c_2^*, c_3^*, c_4^*, c_5^*, c_6^*, c_7^*, c_8^*)$ as the challenge ciphertext of message m_b to the group member mpk.

After receiving the challenge ciphertext c^*, the attacker can still query above oracles but mpk cannot be queried to the Corruption Oracle and c^* cannot be queried to the Decryption Oracle. We answer these queries as above.

Finally, the attacker will output its guess bit b'. We conclude that $(g_1, g_1^\alpha, g_1^\beta, g_1^\delta)$ is a DDH tuple in $\mathbb{G}_1$ if and only if $b' = b$. Note that c^* is valid ciphertext of m under the group member public key mpk_{i_b} if and only if $(g_1, g_1^\alpha, g_1^\beta, g_1^\delta)$ is a DDH tuple. We answer successfully whenever the attacker has a correct guess. Let the attacker win this semantical security game with probability ε. Similarly, we win the DDH challenge with probability $(1 - \frac{n_c}{n})\frac{2}{n}\varepsilon$. This completes the proof.

Theorem 7. *The above group decryption scheme is traceable in the random oracle model under the Strong LRSW assumption in SXDH pairing groups.*

Proof. Assume that we are given a DDH challenge $g_1, X = g_2^x, Y = g_2^y \in \mathbb{G}_1 \times \mathbb{G}_2^2$, where $\mathbb{G}_1, \mathbb{G}_2$ are from pairing groups $\Upsilon = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, g_1, g_2, e) \leftarrow \texttt{PairingGen}(1^\lambda)$. We are required to output a tuple $(a_1 \in \mathbb{G}_1, a_2 = a_1^x, , a_3 = a_1^u, a_4 = a_1^{ux}, a_3 = a_1^{y+uxy})$ for a value $u \in \mathbb{Z}_P^*$ which has never been queried to the LRSW oracle. We first use the LRSW challenge and LRSW oracle to simulate the oracles that the attacker may query and then use the attacker's reply to answer the strong LRSW challenge.

The Setup is the same as the previous proof. We generate the users' key pairing as the real scheme. When attacker requires to add these users into the group, we ask the strong LRSW oracle to obtain the corresponding group certificates and compute the rest parts as the real scheme. All the other can be perfectly simulated as we know the group members' secret keys. Finally, the attacker will output a valid ciphertext $c' = (c_1' \cdots, c_8')$ which we cannot traced.

Note that the condition 1 in the Encryption Verification procedure guarantees that $(c_1' \cdots, c_5')$ is a LRSW tuple. Hence, $\log_{c_1'} c_2' = u'$ is not the secret key of any group member and hence has never been queried to LRSW oracle. Therefore, $(c_1' \cdots, c_5')$ can be used to successfully answer the strong LRSW challenge. This contradicts to the strong LRSW assumption in SXDH pairing groups and completes the proof.

6 Applications

We return to the application scenarios in the introduction. Our above group decryption scheme can be used in these applications. We just show these applications in a high level.

For the applications of the gateway censorship and the identity escrow, the above group decryption scheme can be exploited directly. For the anonymous priced data purchase, the above scheme needs slight modifications. The user may first generate its user public key and apply for group member certificate from a third trusted party. Whenever the user purchases priced data from a dealer, it does Step 3 in the Encryption procedure and sends the corresponding $(c_1, \cdots, c_5)$. Finally, the dealer as the sender and the user as the anonymous recipient in the notion of group decryption can finish the rest of the decryption scheme as usual.

For the application of anonymous fair exchange in the trust third party model between two party A, B, it is a bit intricate. In this case, the two parties first generate their user public keys and apply for group member certificates from a group manager. Then each party does Step 3 in the Encryption procedure and sends the corresponding outputs $(c_1, \cdots, c_5)$ to the other party. Each party then does the rest of the Encryption procedure and also generates group signature on the output of the Encryption procedure using $(c_1, \cdots, c_5)$ from the other party as its public verification key and $\log_{c_1} c_2$ as its secret signing key. This group signature can be generated as those in [1]. Both parties let the trusted third party escrow their anonymous ciphertexts and group signatures before exchange of their secret messages. After the trusted party validates the ciphertexts and group signatures, each party can exchange their secret message with the other party by sending its ciphertexts and the appended group signatures to the other party. If A maliciously aborts the protocol after receiving the secret from B, B can obtains the committed secret from the third party. In this protocol, only the group manger can trace the two parties and the identities of A and B are anonymous to others including the trusted third party who escrows the ciphertexts.

7 Conclusion

In this paper, we formalized the notion of group decryptions. It allows a sender to verifiably encrypt a committed message to any intended member of a group managed by a group manager while the recipient of the ciphertext keeps anonymous. In case of dispute, the group manger can verifiably open the identity of the recipient. We proposed the first anonymous group decryption scheme from pairing groups secure in the random oracle model. Our scheme has constant complexity in both computation and the communication. To achieve our scheme, we presented several building blocks such as commitment schemes to commit a group element in pairing groups and the corresponding zero-knowledge proof protocols. These sub-protocols are efficient and of independent interest.

References

1. G. Ateniese, J. Camenisch, S. Hohenberger and B. de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, 2005. http://eprint.iacr.org/.
2. G. Ateniese, J. Camenisch, and B. de Medeiros. Untraceable RFID tags via insubvertible encryption. In ACM CCS, pp. 92-101, 2005.
3. M. Bellare, A. Boldyreva, A. Desai and D. Pointcheval. Key-privacy in public-key encryption. Asiacrypt'01, LNCS 2248, pp. 566-582. Springer-Verlag, 2001.
4. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. Asiacrypt'01, LNCS 2248, pp. 514-532. Springer-Verlag, 2001.
5. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definition, simplified requirements and a construction based on general assumptions. Eurocypt'03, LNCS 2656, pp. 614-629. Springer-Verlag, 2003.
6. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Proc. 1st ACM Conference on Computer and Communications Security, pp. 62-73. ACM Press, 1993.
7. M. Bellare and P. Rogaway. Optimal Asymmetric encryption-How to encrypt with RSA. Eurocrypt'95, LNCS 921, Springer-Verlag, 1995.
8. X. Boyen and B. Waters. Compact Group Signatures Without Random Oracles. EUROCRYPT 2006, LNCS 4004, pp. 427C444. Springer-Verlag,2006. Earlier version available at: http://eprint.iacr.org/2005/381.pdf.
9. D. Chaum and E. van Heyst. Group Signatures. Eurocrypt'91, LNCS 547, pp. 257-265. Springer-Verlag, 1991.
10. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. Crypto'98, LNCS 1462, Springer-Verlag, 1998.
11. T. ElGamal. A public key cryptosystem and signature scheme based on discrete logarithms. IEEE Transaction on Information Theory, Vol. 31, 1985, pp.467-472.
12. S. D. Galbraith, K. G. Paterson and N. P. Smart. Pairings for cryptographers. Cryptology ePrint Archive, Report 2006/165, 2006. http://eprint.iacr.org/.
13. S. D. Galbraith and V. Rotger. Easy decision Diffie-Hellman groups. Journal of Computation and Mathematics, 7:201-218, 2004.
14. A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. SAC'99, LNCS 1758, pp. 184-199. Springer-Verlag, 1999.
15. J. K. Liu, P. P. Tsang, D. S. Wong, and R. W. Zhu. Universal custodian-hiding verifiable encryption for discrete logarithms. ICISC 2005, LNCS 3935, pp.389 - 409. Springer-Verlag, 2006.
16. J. Liu, V. Wei, and D. Wong. Custodian-hiding verifiable encryption. In WISA 2004, pages 54C67. Springer-Verlag, 2004. LNCS Vol. 3325.
17. R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In Proc. Asiacrypt'01, LNCS 2248, pp. 552-565, Springer-Verlag, 2001.
18. V. Shoup. OAEP Reconsidered. In Proceedings of Crypto'01, Lecture Notes in Computer Science Vol. 2139, pages 239-259. Springer-Verlag, 2001
19. E. R. Verheul. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. Eurocrypt'2001, LNCS 2045, pp. 195-210. Springer-Verlag, 2001.