Practical Group Signatures without Random Oracles

Giuseppe Ateniese[*]
ateniese@cs.jhu.edu

Jan Camenisch[†]
jca@zurich.ibm.com

Breno de Medeiros[‡]
breno@cs.fsu.edu

Susan Hohenberger[§]
srhohen@mit.edu

October 24, 2005

Abstract

We present the first *practical* group signature scheme which is provably secure in the standard model. We provide a new ideal/real-world definition of security for group signatures, encapsulating all the standard properties of unforgeability, anonymity, unlinkability, and exculpability. Our scheme is provably secure under this definition assuming Strong LRSW, q-EDH, and Strong SXDDH. Evidence for the security of any cryptographic assumption we introduce is provided by proving it secure in the generic model. The Strong LRSW and Strong SXDDH assumptions require a special property from bilinear groups (DDH hardness in both pairing groups) which is conjectured to hold for certain MNT curve implementations.

Our signatures are very short (independent of the number of group members), costing roughly 35% more bits than the shortest known group signatures *with* random oracles due to Boneh, Boyen, and Shacham. Our signatures support the standard open algorithm allowing group managers to discover the identity of a signer in $O(n)$ time, where n is the number of group members. We show how to reduce this complexity to $O(\log n)$ by adding a small, constant number of additional bits, and then achieve complexity $O(1)$ under an additional assumption, which also holds in the generic group model.

Keywords: Group signatures, random oracles, standard model, group signature security definition.

1 Introduction

We propose a new group signature scheme that is provably secure in the standard model. While proof-of-concept constructions secure in the standard model have been described by Bellare et al. [9, 11] and many practical group signature schemes secure in the random oracle model exist [23, 4, 13, 32], this is the first *practical* construction secure in the standard model.

[*] Department of Computer Science; The Johns Hopkins University; 3400 N. Charles Street; Baltimore, MD 21218, USA.

[†] IBM Research; Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland.

[‡] Department of Computer Science; Florida State University; 105-D James Love Bldg Tallahassee; FL 32306-4530 USA.

[§] CSAIL; MIT; 32 Vassar Street, Room G694, Cambridge, MA, 02139, USA. Susan's work was performed at IBM Zurich Research, CH-8803 Rüschlikon, Switzerland.

Group signatures are an important cryptographic primitive. Fast and secure implementations of this primitive are especially important given their crucial privacy-preserving role in the current Trusted Computing Group's *anonymous attestation* efforts [16]. Thus, finding efficient schemes that are provably-secure without relying on random oracles is a timely and well-motivated problem.

An Ideal/Real-World Security Definition. An independent and important contribution of this paper is the security model under which our security proof is established. We extend existing security models for group signatures [9, 32, 11] to the universally composable/reactive framework. The UC framework, introduced in an early form in [8], and elaborated and refined in a series of latter works [38, 22, 39], has the property that it enable proofs of security of protocols that are valid not only when the protocol is executed in isolation, but also in the context of concurrent execution and in composition with other arbitrary protocols. A particularly useful formulation of the UC framework combines it with formal methods approach, called *secure reactive systems*, due to Pfitzmann and Waidner [38, 39]. This formulation has the advantage of leading itself to higher-level (and more understandable) proofs. We adopt their methodology in our proof of security. Our scheme is secure under concurrent executions with the exception of the Join protocol, which requires zero-knowledge proofs of knowledge.

As with other cryptographic primitives, general understanding of the security requirements for group signatures schemes evolved over the years. For instance, some early schemes were defeated by subtle attacks against the unforgeability of group membership certificates; only later it became apparent that the *coalition-resistance property* (i.e., the requirement that group members not be able to jointly generate valid member signing keys/certificates without interaction with the group manager) requires that member certificates include signatures from the group manager [5]. Over time, several security requirements were identified, such as unforgeability, anonymity, unlinkability, traceability, coalition-resistance, and freedom from framing, which in combination were taken to specify the security model of group signature schemes. Some significant practical constructions were first proven secure in this semi-formal model, including [4]. What this semi-formal description lacked was an explicit attack model, that went beyond "list of requirements approach" to a unified view of group signature security.

Bellare, Micciancio and Warinschi [9] introduced a more modern security formalism for static group signature schemes based on adversarial games. In practice, that formalization subsided the several requirements of previous works into fewer ones (namely, traceability and anonymity). However, the BMW model is incomplete, not capturing the exculpability requirement that was postulated as early as in the original definition of group signatures by Chaum and van Heyst [23].

The BMW model has been subsequently extensively adapted (via incorporating non-framing requirement, changing the proof model to the random oracle setting and/or to dynamic groups) to allow for security proofs of recently proposed group signature schemes. For instance, the security model of *short group signatures*, recently proposed by Boneh, Boyen, and Shacham [13] and Camenisch and Lysyanskaya [19], includes substantial extensions and modifications of the original BMW model.

The first formal model to apply to the case of dynamic groups was introduced by Kiayias et al. [33, 32]. The formalization works in the random oracle model, not the standard model, but it captures the security requirements of practical group signatures, and it can be readily applied to formally prove the security of practical schemes. Indeed, [33] includes a security proof of a variant of the scheme in [4]. More recently, Bellare, Shi, and Zhang [11] have furthered the BMW model to cover the case of dynamic groups. However, the paper constructs only proof-of-concept schemes based on black-box zero-knowledge primitives, and the existence of a *practical* group signature scheme provably-secure in the *standard* model has remained an open problem.

In this paper, we borrow concepts from these previous works and introduce an alternative security model based on the UC/Reactive models. The security properties of our proposed scheme are therefore stronger than those proven under the other models, as they remain valid in settings where the group signature scheme is composed arbitrarily with other cryptographic protocols in concurrent execution (excluding the Join protocol).

Remark 1: We have not explored the issue of efficient revocation– i.e., removing members from the group –in our security model or in our construction. This omission is a topic for future work.

Remark 2: Our group signature scheme involves a single group manager, and does not separate the functions of enrollment and tracing. Group signature schemes are called *separable* (for instance, see [9, 32, 11]) if they allow for such separation of roles. It is possible to convert our scheme into one that provides separability: It only requires that users first register the *tracing value*, obtaining a signed receipt from the *tracing manager*, that is then provided to the *enrollment manager* during the execution of the Join protocol. The security proofs can then be adapted, and are somewhat longer (by having to deal with more sub-cases), but not fundamentally different. We have succumbed to minimalism by deciding not to incorporate separability directly within our security model: We do not deny its value, only observe that our construction (and model) can support it with relatively straightforward modifications.

Our Construction. We now provide intuition for understanding our construction. A basic group signature scheme has a single group manager and many group members (users) who interact through the standard protocols: Setup, Join, GroupSign, GroupVerify, and Open. All keys and global parameters are established during Setup, then during Join the group manager gives a user a *signing certificate* that allows that user to execute GroupSign on behalf of the group. Signatures generated honestly will be anonymous to everyone except the group manager. If the true identity behind a group signature ever need be known, the group manager can run Open to find the user's identity and prove this fact soundly to any third party.

Now, we explain how to build an efficient group signature scheme without random oracles. Let $S = (Gen, Sign,\ Verify)$ be an efficient, standard signature scheme secure in the standard model; for example, [18, 13, 19]. Let the group manager have keypair (GPK, GSK) and a user have keypair (pk, sk) both generated according to Gen. Consider the following scheme. During the Join protocol, the group manager gives the user a signature of her public key as the signing certificate: $Sign_{GSK}(pk)$. When the user wishes to sign a message m, the user creates her *personal stamp* on m as $Sign_{sk}(m)$ and then outputs the group signature: $(Sign_{GSK}(pk), pk, Sign_{sk}(m))$.

Clearly this scheme is unforgeable, but not anonymous. One problem is that the first two elements of this signature, the user's signing certificate and the user's public key, are always the same for the same user. We solve this problem by having the group manager and the user use different standard signature schemes, S^1 and S^2, to create the signing certificate and personal stamp, respectively. The trick here is to find schemes S^1 and S^2 with special properties that allow us to make the basic structure above anonymous.

First, we implement S^1 with the bilinear-map-based signature scheme due to Camenisch and Lysyanskaya [19] (CL). More specifically, we use the Ateniese et al. [3] extension of this scheme (CL$^+$) which operates as follows. Suppose $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map of prime order p, where g generates $\mathbb{G}_1$ and $\tilde{g}$ generates $\mathbb{G}_2$. Select random $s, t \in \mathbb{Z}_p$ and set $sk = (s, t)$ and $pk = (\tilde{g}^s, \tilde{g}^t)$. To sign a message $m \in \mathbb{Z}_p^*$, select random $a \in \mathbb{G}_1$ and output the tuple $(a, a^t, a^{s+stm}, a^m, a^{mt})$. Verify signature (A, B, C, D, E) by checking that: (1) $e(A, \tilde{g}^t) = e(B, \tilde{g})$, (2) $e(D, \tilde{g}^t) = e(E, \tilde{g})$, and (3) $e(AE, \tilde{g}^s) = e(C, \tilde{g})$.

Now a user's signing certificate is a CL$^+$ signature from the group manager on the user's secret key sk. (Fortunately, the CL$^+$ scheme inherits the efficient CL protocol for getting a signature on a message without revealing the message to the signer. This prevents the group manager from later framing that user.) In the group signature structure $(Sign^1_{GSK}(pk), pk, Sign^2_{sk}(m))$, we replace $(Sign^1_{GSK}(pk), pk)$ with $CL^+_{GSK}(sk) = (a, a^t, a^{s+st(sk)}, a^{sk}, a^{t(sk)})$, which is the group manager's signature on sk which can be thought of as including an obfuscated version of the user's public key (a, a^{sk}). As observed by Ateniese et al. [3], these signatures can be unlinkably, re-randomized by choosing a random $r \in \mathbb{Z}_p$ and computing $(a^r, (a^t)^r, (a^{s+st(sk)})^r, (a^{sk})^r, (a^{t(sk)})^r)$, assuming DDH is hard in $\mathbb{G}_1$. Thus, the user releases a random-looking copy of her signing certificate with each group signature she signs. Now, it remains to find a scheme S^2 that will verify with respect to many obfuscated versions of the user's public key.

We implement S^2 with a new signature scheme secure in the standard model which is based on the *weak* signatures of Boneh and Boyen [13] (BB). By weak, we mean the Boneh-Boyen signature scheme proven adaptively-secure against chosen-message attack *only* for messages logarithmic in the security parameter. The scheme works as follows. Again, we have $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Select a random $sk \in \mathbb{Z}_p$ and a random generator $g \in \mathbb{G}_1$ and output $pk = (g, g^{sk})$. To sign a message $m \in \{0, 1\}^{\log |p|}$, output $A = \tilde{g}^{1/(sk+m)}$. Verify by checking that $e(pkg^m, A) = e(g, \tilde{g})$.

Let us consider our group signature structure using the weak BB signatures implement S^2 (ignore for the moment their restricted message space). The construction is $(CL^+_{GSK}(sk); BB_{sk}(m)) = (a, a^t, a^{s+st(sk)}, a^{sk}, a^{t(sk)}; \tilde{g}^{1/sk+m}) = (A, B, C, D, E; F)$, which can be verified by checking the CL$^+$ signature as normal and then checking that $e(DA^m, F) = e(A, \tilde{g})$. This is great, except that the BB signatures are deterministic and thus it will be obvious when the same user signs the same message twice. Our security definition requires that the user be guaranteed more privacy than this. Thus, we need to tweak this construction, and along the way, enable longer messages.

In their paper [13], Boneh and Boyen present one method for adapting the weak scheme to longer messages. In this paper, we present another method, which we denote BB$^+$, that is more suited to our purposes. To sign a message $m \in \mathbb{Z}_p^*$, select a random $v \in \mathbb{Z}_p$ and output the tuple $(g^v, \tilde{g}^{1/(sk+v)}, \tilde{g}^{1/(v+m)})$. Verify signature (A, B, C) by checking that: (1) $e(pkA, B) = e(g, \tilde{g})$ and (2) $e(Ag^m, C) = e(g, \tilde{g})$.

Now the construction is of the form $(CL^+_{GSK}(sk); BB^+_{sk}(m))$, but to tie things together, we use the same (random) generator for the CL$^+$ components and the first of the BB$^+$ ones. This results in the signature $(a, a^t, a^{s+st(sk)}, a^{sk}, a^{t(sk)}; a^v, \tilde{g}^{1/sk+v}, \tilde{g}^{1/(v+m)})$ for some message $m \in \mathbb{Z}_p^*$, where $a \in \mathbb{G}_1$ and $v \in \mathbb{Z}_p$ are randomly chosen for each new signature. With this construction, we are able to prove both anonymity and security for longer messages.

We have now described the entire construction, except the Open algorithm; that is, how the group manager uncovers the user identity behind a signature. The simpliest method is for the user to give the group manager a *tracing value* $\tilde{g}^{sk}$ during the Join protocol. Later, the group manager can open a signature $(a, a^t, a^{s+st(sk)}, a^{sk}, a^{t(sk)}; a^v, \tilde{g}^{1/sk+v}, \tilde{g}^{1/(v+m)}) = (A, B, C, D, E; F, G, H)$ by testing if $e(A, \tilde{g}^{sk}) = e(D, \tilde{g})$ for each user. This algorithm runs in $O(n)$ time, where n is the number of group members. This is not very satisfactory, so in Section 6, we describe two sublinear Open algorithms. The first has complexity $O(\log n)$ under the same cryptographic assumptions as this basic scheme; the second reaches complexity $O(1)$ under an additional assumption.

Length of Signatures. The signatures produced by this scheme are very short. Assuming that the bitlength of elements in $\mathbb{G}_1$ is 171, and that the curves implemented in the MIRACL library are used [40], the basic scheme achieves roughly the same level of security as a 1024-bit RSA signature [13]. For these curves, the bitlengths of elements in $\mathbb{G}_2$ are roughly three times that of $\mathbb{G}_1$, and our scheme would then take approximately 2052 bits to represent each signing value, which comprise of six elements in $\mathbb{G}_1$ and two elements in $\mathbb{G}_2$. If the newer curves of embedding degree 12 [7] are used, one could employ 256-bit curves to achieve RSA security equivalent to 3072 bits. These new curves have better ratios, with $\log |\mathbb{G}_2| / \log |\mathbb{G}_1| = 2$. In this case, our signatures would take 2560 bits to represent, effectively shorter than a plain RSA signature with the same security parameter. [1] In contrast, the short group signatures of Boneh, Boyen, and Shacham [13] *with* random oracles– the shortest secure implementation known –would take 1533 bits for the RSA-1024 security level (or 1022 implementing the simplification for XDDH groups), and approximately 2298 bits (or about 1532 bits with the XDDH simplification) for the RSA-3072 security level.

Our Complexity Assumptions and the Generic Group Model. Our constructions are provably secure under the Strong LRSW, q-EDH, and Strong SXDDH assumptions as defined in Section 3. The security of the Strong LRSW and Strong SXDDH depend on having a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ where DDH is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$. Good candidates for such bilinear mappings are certain MNT curve implementations where no efficient isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ are known [6, 3, 30]. Our first assumption, Strong LRSW, was previously used along with the assumption of a mapping with DDH hard in $\mathbb{G}_1$ and $\mathbb{G}_2$ by Ateniese, Camenisch, and de Medeiros [3]. To these assumptions, we add two more.

The q-EDH assumption is an extension of the q-SDH assumption of Boneh and Boyen [13]. We show in the proof of Theorem 7.1 that Boneh and Boyen's result for q-SDH can be extended to show that q-EDH is just as hard to solve in generic groups. The Strong SXDDH assumption states that DDH remains hard in $\mathbb{G}_1$ and $\mathbb{G}_2$ even when some additional information about the DDH instance is available. We show in the proof of Theorem 7.3 that this problem is also hard in the generic group model. This generic group proof, however, is more involved than the q-EDH one, and requires some new ideas.

2 Group Signature Security Definition

Notation: if P is a protocol between parties A and B, then $P(A(x), B(y))$ denotes that A's input is x and B's input is y.

A group signature scheme consists of the usual types of players: a group manager $\mathcal{GM}$ and a user $\mathcal{U}_i$. These players can execute the algorithms: GroupSetup, UserKeyGen, Join, GroupSign, GroupVerify, Open, and VerifyOpen. We now specify the input-output specifications for these algorithms as well as providing some informal intuition for what they do.

[1] We remark that we are comparing the difficulty of solving the discrete logarithm and that of factoring. A more concrete security assessment of our constructions would take into consideration the efficiency of the security reductions, and the generic model estimates of the relative strengths of the underlying cryptographic assumptions vis-a-vis algorithms for the discrete logarithm.

Let *params* be some global parameters generated during a setup phase; ideally *params* would be empty.

- The $\mathsf{GroupSetup}(1^k, params)$ algorithm is a key generation algorithm for the group manager $\mathcal{G}M$. It takes as input the security parameter 1^k and outputs the key pair $(pk_{\mathcal{G}M}, sk_{\mathcal{G}M})$. (Assume that $sk_{\mathcal{G}M}$ contains the *params*, so we do not have to give *params* explicitly to the group manager again.)
- The $\mathsf{UserKeyGen}(1^k, params)$ algorithm is a key generation algorithm for a group member $\mathcal{U}$, which outputs $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$. (Assume that $sk_{\mathcal{U}}$ contains the *params*, so we do not have to give *params* explicitly to the user again.)
- In the $\mathsf{Join}(\mathcal{U}(pk_{\mathcal{G}M}, sk_{\mathcal{U}}), \mathcal{G}M(pk_{\mathcal{U}}, sk_{\mathcal{G}M}))$ protocol, the user $\mathcal{U}$ joins the signatory group managed by $\mathcal{G}M$. The user's output is a personalized group signing credential $C_{\mathcal{U}}$, or an error message. $\mathcal{G}M$'s output is some information $T_{\mathcal{U}}$ which will allow the group manager to revoke the anonymity of any signatures produced by $\mathcal{U}$. The group manager maintains a database D for this revocation information, to which it adds the record $(pk_{\mathcal{U}}, T_{\mathcal{U}})$.
- The $\mathsf{GroupSign}(sk_{\mathcal{U}}, C_{\mathcal{U}}, m)$ algorithm allows group members to sign messages. It takes as input the user's secret key $sk_{\mathcal{U}}$, the user's signing credential $C_{\mathcal{U}}$, and an arbitrary string m. The output is a group signature σ.
- The $\mathsf{GroupVerify}(pk_{\mathcal{G}M}, m, \sigma)$ algorithm allows to publicly verify that σ is a signature on message m generated by some member of the group associated with group public key $pk_{\mathcal{G}M}$.
- The $\mathsf{Open}(sk_{\mathcal{G}M}, m, \sigma)$ algorithm allows the group manager, with $sk_{\mathcal{G}M}$, to identify the group member $\mathcal{U}$ who was responsible for creating the signature σ on message m. The output is a member identity $pk_{\mathcal{U}}$ or an error message.
- In the $\mathsf{VerifyOpen}(\mathcal{G}M(sk_{\mathcal{G}M}, m, \sigma, pk, Q), \mathcal{V}(pk_{\mathcal{G}M}, m, \sigma, pk))$ protocol, $\mathcal{G}M$ convinces a verifier that the user with public key pk was responsible for creating the signature σ on message m. The verifier outputs either 1 (accept) or 0 (reject).

In addition to supporting the above algorithms, a group signature scheme must also be *correct* and *secure*. Correctness is fairly straightforward. Informally, if an honest user runs Join with an honest group manager, then neither will output an error message. If an honest user runs GroupSign, then the output will be accepted by an honest verifier running GroupVerify. If a signature passes GroupVerify and a honest manager runs Open, then the result will be accepted by an honest verifier running VerifyOpen.

2.1 The Group Signature Ideal Functionality, $\mathcal{F}_{gs}$

We now define our security model for group signatures. We use the ideal/real world model as in the models of multiparty computation [20, 21, 22] and reactive systems [38, 39]. Let us briefly recall this model.

In the real world, there are a number of parties who together execute some cryptographic protocol. Some number of these parties may be corrupted by the adversary $\mathcal{A}$ (all corrupted parties are combined into this single adversary). Each party receives its input and reports its output to the environment $\mathcal{Z}$. The environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ may arbitrarily interact.

In the ideal world, we have the same parties. As before, each party receives its input and reports its output to the environment. However, instead of running a cryptographic protocol, the parties provide their inputs and receive their outputs from a trusted party $\mathcal{T}$. The specification for how $\mathcal{T}$ behaves is formalized as an ideal functionality. In a moment, we will describe such a functionality for group signatures $\mathcal{F}_{gs}$.

We say that a cryptographic protocol securely implements an ideal functionality if for every real-world adversary $\mathcal{A}$ and every environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$, which controls the same parties in the ideal world as $\mathcal{A}$ does in the real world, such that $\mathcal{Z}$ cannot distinguish whether it is interacting in the real world with $\mathcal{A}$ or in the ideal world with $\mathcal{S}$.

We now describe $\mathcal{F}_{gs}$. In addition to the environment $\mathcal{Z}$, we have the following types of players: a group manager $\mathcal{G}M$ and a user $\mathcal{U}_i$. We work in the *non-adaptive* setting.

- Non-adaptive Setup: Each user $\mathcal{U}_i$ tells the functionality $\mathcal{F}_{gs}$ whether or not it is corrupted. Optionally, in this stage the global parameters are broadcast to all parties.

We define the stateful, group signature functionality, $\mathcal{F}_{gs}$, to behave as follows:

- GroupSetup: Upon receiving ($\mathcal{G}M$, "group setup") from $\mathcal{G}M$, send ($\mathcal{G}M$, "group setup") to $\mathcal{S}$.
- UserKeyGen: Similarly, upon receiving ($\mathcal{U}_i$, "keygen") from $\mathcal{U}_i$, forward the request to $\mathcal{S}$ and return its output.
- Join: Upon receiving ($\mathcal{U}_i$, "enroll") from $\mathcal{U}_i$, ask the group manager $\mathcal{G}M$ if $\mathcal{U}_i$ may join the group. The $\mathcal{G}M$ responds with $res_i \in \{0,1\}$. Record the pair $(\mathcal{U}_i, res_i)$ in database D and return res_i to $\mathcal{U}_i$. Additionally, if the group manager is corrupted, then register a special user corrupt-$\mathcal{G}M$.
- GroupSign: Upon receiving ($\mathcal{U}_i$, "sign"$, m$), where m is an arbitrary string, check that $\mathcal{U}_i$ is a valid member of the group by checking that the entry for $\mathcal{U}_i$ in D has $res_i = 1$. If not, deny the command. Otherwise, tell the simulator $\mathcal{S}$ that GroupSign has been envoked on message m. If the $\mathcal{G}M$ is corrupted, also tell the simulator the identity $\mathcal{U}_i$. Ask $\mathcal{S}$ for a signature index id. Record the entry $(\mathcal{U}_i, m, id)$ in database L and return the value id to $\mathcal{U}_i$.
- GroupVerify: Upon receiving ($\mathcal{U}_i$, "verify"$, m, id$) from $\mathcal{U}_i$ (or $\mathcal{G}M$), search database L for an entry containing message m, and if one exists, return 1. Otherwise, return 0.
- Open: This ideal operation combines both the Open and VerifyOpen cryptographic protocols. Upon receiving ($\mathcal{U}_i$, "open"$, m, id$) from $\mathcal{U}_i$, search database L for an entry $(\mathcal{U}_j, m,\ id)$ for any $\mathcal{U}_j$. Ask $\mathcal{G}M$ if it will allow $\mathcal{F}_{gs}$ to open id for user $\mathcal{U}_i$. If $\mathcal{G}M$ agrees and $\mathcal{U}_j \neq$ corrupt-$\mathcal{G}M$, then output the identity $\mathcal{U}_j$. Otherwise, output $\perp$.

Designing an ideal functionality for group signatures $\mathcal{F}_{gs}$ was not a straightforward task. In particular, we had to think about how $\mathcal{F}_{gs}$ should respond when asked to open a signature in different scenarios. First, how should $\mathcal{F}_{gs}$ respond when asked to open a signature on a message that many parties have signed? Second, how should it respond when asked to open a re-randomization of a valid signature? Will $\mathcal{F}_{gs}$ even know who originally signed it?

Let us provide some intuition for understanding this model. Informally, the properties that we capture are unforgeability, anonymity, and exculpability. First, it may not be obvious, but this definition is general enough to capture unforgeability under adaptive chosen message attack [31] without *requiring* schemes to be *strongly* unforgeable [2]. Recall that in strongly unforgeable schemes a new signature on a previously signed message is considered a forgery; while in the standard notion, and for most applications, it is not. At first glance, one might assume that this definition requires strong unforgeability because GroupVerify only returns valid for signatures that $\mathcal{F}_{gs}$ saw being created. Indeed, this would be an easier functionality to write. However, to broaden our scope, we can mentally treat these *id*'s as indices of signature equivalence classes, i.e., including a signature and all valid re-randomizations of it.

The definition also captures exculpability (i.e., even a rogue group manager cannot frame an honest user). Consider that the environment $\mathcal{Z}$ may instruct a user to sign any messages of its choosing and may interact freely with the adversary $\mathcal{A}$. Our model, however, enforces that unless an honest user $\mathcal{U}_i$ requested a signature on m (i.e., sent ("sign", m) to $\mathcal{F}_{gs}$), then for all values of id, the Open command on $(\mathcal{U}_i, m, id)$ will return $\perp$.

Furthermore, there is a strong anonymity guarantee for a user: unless the group manager is corrupted, the users remain anonymous. When the group manager is honest, the simulator must create signatures for $\mathcal{A}$ knowing only the message contents, but not the identity of the honest user.

Finally, the definition ensures that, whenever the group manager is honest, he will be able to open all group signatures. During the Open command, $\mathcal{F}_{gs}$ only asks $\mathcal{S}$ for permission to execute the opening if the group manager is corrupted. Thus, if a user honestly runs the verification algorithm and accepts a signature as valid, then this user may be confident that an honest $\mathcal{G}M$ will later be able to open it, reveal the identity of the original signer, and prove this fact to the user.

3 Preliminaries and Complexity Assumptions

Notation: The notation $G = \langle g \rangle$ means that g generates the group G.

3.1 Bilinear Maps

Let BilinearSetup be an algorithm that, on input the security parameter 1^k, outputs the parameters for a bilinear mapping as $\gamma = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g})$, where $\mathbb{G}_1 = \langle g \rangle$ and $\mathbb{G}_2 = \langle \tilde{g} \rangle$. We follow the notation of Boneh, Lynn, and Shacham [15]:

1. $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ are all (multiplicative) groups of prime order $p = \Theta(2^k)$;
2. each element of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ has a unique binary representation;
3. e is an efficiently computable bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ such that:
   - (Bilinear) for all $g \in \mathbb{G}_1$, $\tilde{g} \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$, $e(g^a, \tilde{g}^b) = e(g, \tilde{g})^{ab}$;
   - (Non-degenerate) if g is a generator of $\mathbb{G}_1$ and $\tilde{g}$ is a generator of $\mathbb{G}_2$, then $e(g, \tilde{g})$ generates $\mathbb{G}_T$.

In addition to the standard bilinear map properties above, in this paper, we restrict our attention to maps with the following additional condition.

The parameters required for any of these options (e.g., the hash function for option (c)) are assumed to be global information outside the control of the group manager.

Assumption 1 (Symmetric External Decisional Diffie-Hellman (SXDDH) [6, 3, 30]) *The classic Decisional Diffie-Hellman (DDH) problem is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$. This implies that there do* not *exist efficiently computable isomorphisms $\psi : \mathbb{G}_1 \rightarrow \mathbb{G}_2$ or $\psi' : \mathbb{G}_2 \rightarrow \mathbb{G}_1$.*

The asymmetric version of this assumption, simply called XDDH, only requires that DDH be hard in $\mathbb{G}_1$ [29, 41, 35, 13, 6]. (This assumption was previously called XDH [6]; for clarity, we insert the word "Decisional.") The SXDDH assumption was recently introduced by Ballard et al. [6]. Let us say a few words about this new assumption. Most of the initial cryptographic interest in bilinear groups was because they allowed for a group G in which DDH was easy, and yet CDH was believed to remain hard.

We stress that the pairing is not sufficient to decide the traditional DDH problem in either $\mathbb{G}_1$ or $\mathbb{G}_2$. Instead the latter property is obtained via isomorphisms $\psi : \mathbb{G}_1 \rightarrow \mathbb{G}_2$ (also called distortion maps). When such an isomorphism is available, a "distorted pairing" can be defined as $e(g, \psi(g))$ for $g \in \mathbb{G}_1$.

Efficient distortion maps always seem to exist for bilinear implementations over the popular supersingular curves, however, they do not appear to be present for *all* pairing groups, in particular those recently discovered over MNT curves [43, 28, 30]. When distortion maps are not available, it is an open problem whether the groups $\mathbb{G}_1$, $\mathbb{G}_2$ are DDH-easy.

3.2 Complexity Assumptions

The security of our construction in Section 5 is based on the following assumptions about bilinear groups. For the two assumptions that we introduce, we argue in Section 7 that these assumptions are reasonable to make, by showing that they hold in the generic group model [36, 42].

Throughout this section, suppose that $\mathsf{BilinearSetup}(1^k) \rightarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g})$, where $\mathbb{G}_1 = \langle g \rangle$ and $\mathbb{G}_2 = \langle \tilde{g} \rangle$, is public information. Assume that $(\mathbb{G}_1, \mathbb{G}_2)$ are SXDDH-hard groups.

Unforgeability Assumptions. The LRSW assumption is a computational discrete-logarithm assumption originally introduced by Lysyanskaya et al. [34] and used in many subsequent works. Recently, a stronger form of the LRSW assumption, called *Strong LRSW*, was introduced by Ateniese et al. [3]. Strong LRSW only holds in SXDDH-hard groups.

Assumption 2 (Strong LRSW [3]) *Let $X, Y \in \mathbb{G}_2, X = \tilde{g}^x, Y = \tilde{g}^y$. Let $O_{X,Y}(\cdot)$ be an oracle that takes as input a value $m \in \mathbb{Z}_p^*$, and outputs an LRSW-tuple (a, a^x, a^{y+yxm}) for a random $a \in \mathbb{G}_1$. Then for all probabilistic polynomial-time adversaries $\mathcal{A}^{(\cdot)}$ and all $m \in \mathbb{Z}_p^*$,*

$$\begin{aligned}\Pr[x \xleftarrow{R} \mathbb{Z}_p, y \xleftarrow{R} \mathbb{Z}_p, X = \tilde{g}^x, Y = \tilde{g}^y, (a_1, a_2, a_3, a_4, a_5) \leftarrow \mathcal{A}^{O_{X,Y}}(g, \tilde{g}, X, Y) :\\ m \notin Q \;\wedge\; a_1 \in \mathbb{G}_1 \;\wedge\; a_2 = a_1^x \;\wedge\; a_3 = a_1^{y+yxm} \;\wedge\; a_4 = a_1^m \;\wedge\; a_5 = a_1^{mx}] < 1/\text{poly}(k),\end{aligned}$$

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

The q-Strong Diffie-Hellman (q-SDH) assumption, as introduced by Boneh and Boyen [12], states that: for all probabilistic polynomial-time adversaries $\mathcal{A}$, and all $c \in \mathbb{Z}_p^*$:

$$\Pr[x \xleftarrow{R} \mathbb{Z}_p : \mathcal{A}(g, \tilde{g}, g^x, \ldots, g^{(x^q)}, \tilde{g}^x, \ldots, \tilde{g}^{(x^q)}) = (c, \tilde{g}^{1/(x+c)})] < 1/\text{poly}(k).$$

This formulation is no stronger (indeed, perhaps weaker) than the Boneh and Boyen presentation of q-SDH [12], in which $\mathcal{A}$ is given input $(g, \tilde{g}, \tilde{g}^x, \ldots, \tilde{g}^{(x^q)})$ *and* one explicitly assumes that there exists an efficiently computable distortion map $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$. Here, we *require* that such a distortion map does *not* exist and instead explicitly provide values in both $\mathbb{G}_1$ and $\mathbb{G}_2$. Now, suppose we strengthen the above q-SDH assumption as follows: to win, the adversary need not explicitly provide $\tilde{g}^{1/(x+c)}$, but can instead provide some related values.

Assumption 3 (q-Extended Diffie-Hellman (q-EDH)) *For all probabilistic polynomial-time adversaries $\mathcal{A}$, all $v, c \in \mathbb{Z}_p^*$, and all values $h \in \mathbb{G}_1$ such that $h \neq 1$,*

$$\Pr[x \xleftarrow{R} \mathbb{Z}_p : \mathcal{A}(g, \tilde{g}, g^x, \ldots, g^{(x^q)}, \tilde{g}^x, \ldots, \tilde{g}^{(x^q)}) = (c, h, h^v, \tilde{g}^{1/(x+v)}, \tilde{g}^{1/(v+c)})] < 1/\text{poly}(k).$$

In Theorem 7.1, we show that q-EDH is hard in generic groups; this result holds independently of whether or not DDH is hard in either $\mathbb{G}_1$ or $\mathbb{G}_2$. Obviously, in any group where q-EDH holds, q-SDH also holds.

Anonymity Assumption. The anonymity of our group signatures is based on a single assumption: SXDDH holds even when the adversary is given access to oracles revealing some additional information about the DDH instance.

Assumption 4 (Strong SXDDH) *Let $g \in \mathbb{G}_1$, $\tilde{g} \in \mathbb{G}_2$, and $x \in \mathbb{Z}_p$. Let $O_x(\cdot)$ be an oracle that takes as input $m \in \mathbb{Z}_p^*$ and outputs $(g^v, \tilde{g}^{1/(x+v)}, \tilde{g}^{1/(v+m)})$ for a random $v \in \mathbb{Z}_p^*$. Let $Q_y(\cdot)$ be an oracle that takes the same input type and outputs $(g^r, g^{ry}, g^{rv}, \tilde{g}^{1/(y+v)}, \tilde{g}^{1/(v+m)})$ for a random $r, v \in \mathbb{Z}_p^*$. Then for all probabilistic polynomial-time adversaries $\mathcal{A}^{(\cdot)}$, and for randomly chosen $g \in \mathbb{G}_1$, $\tilde{g} \in \mathbb{G}_2$, and $x, y \in \mathbb{Z}_p$,*

$$|\Pr[\mathcal{A}^{O_x, Q_x}(g, g^x, \tilde{g}) = 1] - \Pr[\mathcal{A}^{O_x, Q_y}(g, g^x, \tilde{g}) = 1]| < 1/\text{poly}(k).$$

In Theorem 7.3, we show that Strong SXDDH is hard in generic groups. We observe, because it will be useful later, that the Strong SXDDH assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ implies CDH in $\mathbb{G}_1$, as well as CoCDH in $(\mathbb{G}_1, \mathbb{G}_2)$; that is, given $(g, \tilde{g}, g^x, \tilde{g}^y)$, it is hard to compute $\tilde{g}^{xy}$.

This ends our description of the assumptions on which our scheme is based.

4 Key Building Blocks: CL and BB Signatures

As mentioned in Section 1, our group signature scheme is built out of the standard signature schemes of Camenisch and Lysyanskaya [19] (CL) and Boneh and Boyen [12] (BB). Both of these building blocks are efficient, bilinear map based schemes, secure in the plain model (i.e., without random oracles.)

4.1 Camenisch-Lysyanskaya Signatures

Recall the Camenisch-Lysyanskaya (CL) signature scheme [19]. Let the security parameter be 1^k. The global parameters are the description of a bilinear mapping $params = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g})$, where $\mathbb{G}_1 = \langle g \rangle$ and $\mathbb{G}_2 = \langle \tilde{g} \rangle$, obtained by running $\mathsf{BilinearSetup}(1^k)$.

- **Key generation:** Choose random $s, t \in \mathbb{Z}_p$. Set $pk = (\tilde{g}^s, \tilde{g}^t)$ and $sk = (s, t)$.
- **Signing:** Choose random $a \in \mathbb{G}_1$, output $\sigma = (a, a^t, a^{s+stm})$ as the signature on $m \in \mathbb{Z}_p^*$.
- **Verification:** On input a purported signature $\sigma = (A, B, C)$ and a message m, accept if and only if: (1) $e(B, \tilde{g}) = e(A, \tilde{g}^t)$ and (2) $e(C, \tilde{g}) = e(A, \tilde{g}^s)e(B, \tilde{g}^s)^m$.

This scheme was proven unforgeable under adaptive chosen-message attack under the LRSW assumption [19]. It was also shown to support a useful protocol, which we now describe.

Let us review the basic Pedersen commitment scheme [37], in which the public parameters are a group G of prime order p, and two generators g, h of G. In order to commit to the value $m \in \mathbb{Z}_p$, pick a random $r \in \mathbb{Z}_p$ and set $C = \mathrm{PedCom}(m; r) = g^m h^r$. When the randomness is not relevant, we will sometimes omit it and write $C = \mathrm{PedCom}(m)$.

CL signatures support an efficient two-party protocol for obtaining a CL signature on the value committed to in a Pedersen commitment. The common inputs are $C = \mathrm{PedCom}(m; r)$ and the verification key of the signer GPK. The signer additionally knows the corresponding signing key GSK, while the receiver additionally knows m and r. As a result of this protocol, the receiver obtains the signature $\sigma_{GSK}(m)$, while the signer does not learn anything about m. For our current purposes, it will not matter *how* this protocol actually works.

Now, we make two extensions to the CL signatures. We denote this new scheme as CL$^+$.

- **Key generation:** Same as before. We have $GPK = (\tilde{g}^s, \tilde{g}^t)$ and $GSK = (s, t)$.
- **Signing:** Choose random $a \in \mathbb{G}_1$, output $\sigma = (a, a^t, a^{s+stm}, a^m, a^{mt})$ as the signature on *hidden* message $m \in \mathbb{Z}_p^*$.
- **Verification:** On input a purported signature $\sigma = (A, B, C, D, E)$ accept that σ authenticates the message hidden as $\log_A(D)$ if and only if:

$$e(B, \tilde{g}) = e(A, \tilde{g}^t), \quad e(D, \tilde{g}^t) = e(E, \tilde{g}), \quad e(C, \tilde{g}) = e(A, \tilde{g}^s)e(E, \tilde{g}^s).$$

- **Re-Randomization:** On input a signature $\sigma = (A, B, C, D, E)$, choose a random $r \in \mathbb{Z}_p^*$ and output $(A^r, B^r, C^r, D^r, E^r)$.

Ateniese et al. [3] previously observed that when CL$^+$ signatures are set in bilinear groups where SXDDH holds in $\mathbb{G}_1$ and $\mathbb{G}_2$, this re-randomization is *unlinkable*. We will formally argue this later in Lemma 5.3.

4.2 Boneh-Boyen Signatures

Recall the weak Boneh-Boyen (BB) signature scheme [12]. Let the security parameter be 1^k. The global parameters are the description of a bilinear mapping $params = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ obtained by running $\mathsf{BilinearSetup}(1^k)$ (here, we ignore the generators output by $\mathsf{BilinearSetup}$).

- **Key generation:** Select random generators $g \in \mathbb{G}_1$ and $\tilde{g} \in \mathbb{G}_2$. Select random $sk \xleftarrow{R} \mathbb{Z}_p^*$. Set $pk = (g, \tilde{g}, g^{sk})$.
- **Signing:** On input a secret key sk and a message $m \in \mathbb{Z}_p^*$, output the signature $\tilde{g}^{1/(sk+m)}$.
- **Verification:** On input a public key $(g, \tilde{g}, g^{sk})$, a message m, and a purported signature σ, accept if it holds that $e(\sigma, g^{sk}g^m) = e(g, \tilde{g})$, and reject otherwise.

In this presentation, we reverse the roles of $\mathbb{G}_1$ and $\mathbb{G}_2$ from the original description [12]. This scheme was proven unforgeable only against weak chosen-message attack under the q-SDH assumption [12], where the adversary must submit all of his signature queries in advance of the public key generation. Subsequently, Dodis and Yampolskiy [25] used the fact that this scheme can handle adaptive queries by limiting the message space to strings of length $O(\log k)$, where k is the security parameter; i.e., to a size where all possible messages may be enumerated in advance by the reduction. In either case [12, 25], the authors reduced this scheme to an assumption which they then argued was hard in the generic group model. In Theorem 7.1, we show that a simple extension of this "weak" signature scheme [12] is actually existentially unforgeable under *adaptive* chosen-message attack in the generic group model.

5 Our Basic Group Signature Construction

Notation: BB and CL$^+$, respectively, denote Boneh-Boyen [12] signature scheme and our Section 4 modifications of the Camenisch-Lysyanskaya [19] signature scheme. When we write $s = Sign^{CL+}_{GSK}(m; a)$, we mean that s is a CL$^+$ signature under secret key GSK on message m using base a; that is, $s = (a, a^t, a^{s+stm}, a^m, a^{mt})$ for $GSK = (s, t)$. Similarly, when we write $s = Sign^{BB}_{sk}(m; \tilde{g})$, we mean that s is a BB signature under secret key sk on message m; that is, $s = \tilde{g}^{1/(sk+m)}$.

- Global parameters: bilinear groups $\mathbb{G}_1 = \langle a \rangle, \mathbb{G}_2 = \langle \tilde{g} \rangle$ of order p, where $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.
- Group Manager's Keys: (GPK, GSK), one CL$^+$ signature key pair.
- User's Keys: (pk, sk), where $sk \in \mathbb{Z}_p^*$.
- Join: The user's certificate is $Sign^{CL+}_{GSK}(sk; a)$, for a random $a \in \mathbb{G}_1$ chosen by the Group Manager. This is obtained without the Group Manager learning sk.
- GroupSign: For a message $m \in \mathbb{Z}_p^*$, the user chooses a random $v, r \in \mathbb{Z}_p^*$, and outputs his group signature (where $a^r = b$):

$$\begin{aligned}&\big(Rand(Sign^{CL+}_{GSK}(sk; a), r),\ a^{vr},\quad Sign^{BB}_{sk}(v; \tilde{g}),\quad Sign^{BB}_{v}(m; \tilde{g})\big)\\ &\quad = \big(b, b^t, b^{s+st(sk)}, b^{sk}, b^{t(sk)}, b^v,\qquad \tilde{g}^{1/(sk+v)},\qquad \tilde{g}^{1/(v+m)}\big)\end{aligned}$$

- GroupVerify: Check that each of the three signature components are valid. The public keys for the BB signatures are derived from the CL$^+$ components.

Figure 1: An overview of our basic group signature construction; we show how to open signatures in constant time in the next section.

We now describe the first practical group signature scheme without random oracles; for now, we settle for an open algorithm with complexity linear in the number of group members. Let $params = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g})$, where $\mathbb{G}_1 = \langle g \rangle$ and $\mathbb{G}_2 = \langle \tilde{g} \rangle$, be the output of BilinearSetup(1^k). Figure 1 contains a high-level overview.

GroupSetup$(1^k, params)$ The group manager establishes the public parameters for the Pedersen commitment scheme [37] and adds those to *params*. Then, the group manager executes $Gen^{CL+}(1^k, params)$ to obtain:

$$GPK = (params, \mathcal{S} = \tilde{g}^s, \mathcal{T} = \tilde{g}^t) \quad , \quad GSK = (s, t).$$

UserKeyGen$(1^k, params)$ Each user $\mathcal{U}$ selects random $sk \in \mathbb{Z}_p^*$ and random $h \in \mathbb{G}_1$, and outputs the public key $pk = (h, e(h, \tilde{g})^{sk})$.

Join$(\mathcal{U}_i(GPK, sk_i), \mathcal{GM}(pk_i, GSK))$ In this interactive protocol, the user's inputs are her secret key sk_i and the public key of the group manager GPK. Likewise, the group manager receives as input GSK and pk_i. They interact as follows:

1. $\mathcal{U}_i$ submits her public key pk_i. The user provides a zero-knowledge proof of knowledge of her corresponding secret key sk_i using any proof technique that is *extractable*. We discuss several techniques for extractable proofs later in this section.
2. $\mathcal{U}_i$ submits her tracing information $Q_i = \tilde{g}^{sk_i}$ to $\mathcal{GM}$. Let $pk_i = (p_1, p_2)$. If $e(p_1, Q_i) \neq p_2$ or sk_i was already in D, $\mathcal{GM}$ aborts. Otherwise, $\mathcal{GM}$ enters Q_i in database D.
3. The user sends a commitment $A = \text{PedCom}(sk_i)$ to $\mathcal{GM}$. The user and $\mathcal{GM}$ run the CL protocol for obtaining $\mathcal{GM}$'s signature on the committed value contained in commitment A. $\mathcal{GM}$ picks a random $r \in \mathbb{Z}_p^*$ and sets $f_1 = g^r$. Then, $\mathcal{GM}$ computes $Sign^{CL+}_{GSK}(sk_i; f_1) = (f_2, f_3)$ and sends all three values to the user. If the CL signature (f_1, f_2, f_3) does not verify for message sk_i, the user aborts.
4. Next, the user locally computes the values $f_4 = f_1^{sk_i}$ and $f_5 = f_2^{sk_i}$.
5. At the end of this protocol, the user obtains the following membership certificate:

$$C_i = (f_1, \ldots, f_5) = (a, a^t, a^{s+st(sk_i)}, a^{sk_i}, a^{(sk_i)t}).$$

As an extra security precaution against an adversary choosing keys maliciously, we could optionally have the $\mathcal{GM}$ re-randomize each user's secret key during this protocol. Briefly, this would work as follows: *after* $\mathcal{U}_i$ sends $\text{PedCom}(sk_i)$ to $\mathcal{GM}$, $\mathcal{GM}$ sends random $a, b \in \mathbb{Z}_p$ to $\mathcal{U}_i$ along with a CL signature on the new secret key $sk'_i = a(sk_i) + b \bmod p$. Our scheme is secure even for maliciously chosen key pairs, so this step is not actually required for the proof of Theorem 5.1.

GroupSign(sk_i, C_i, m) A user with secret key sk_i and membership certificate $C_i = (f_1, \ldots, f_5)$ may sign a message $m \in \mathbb{Z}_p^*$ as follows. (Technically, the message space does not include zero, but this can be easily patched; for example, add a 1 to the end of each message.)

1. The user re-randomizes her certificate C_i using a random value $r \in \mathbb{Z}_p$. That is, she computes $(a_1, \ldots, a_5) = (f_1^r, \ldots, f_5^r)$.
2. Next, the user chooses a random $v \in \mathbb{Z}_p^*$ and sets $a_6 = a_1^v$.
3. The user certifies v by generating a BB signature on v using his secret key sk_i; this signature is $Sign^{BB}_{sk_i}(v; \tilde{g}) = \tilde{g}^{1/(sk_i+v)}$. We denote this value as a_7.
4. Next, the user treats the value v as his "one-time" signing key and computes a BB signature on m using key v; this signature is $Sign^{BB}_v(m; \tilde{g}) = \tilde{g}^{1/(v+m)}$. We denote this value as a_8.
5. Finally, the user outputs the group signature $(a_1, \ldots, a_8)$ on message m:

$$(b, b^t, b^{s+st(sk_i)}, b^{sk_i}, b^{(sk_i)t}, b^v, \tilde{g}^{1/(sk_i+v)}, \tilde{g}^{1/(v+m)}),$$

where we denote $b = a^r$.

GroupVerify(GPK, m, σ) To verify that $\sigma = (a_1, \ldots, a_8)$ is a group signature on m, do:

1. Use helper a_5, to test that (a_1, a_2, a_3) is a valid CL$^+$ signature for public key GPK where the message is the exponent of a_4 (base a_1). That is, verify the following relations:

   $$e(a_1, \mathcal{T}) = e(a_2, \tilde{g}), \qquad e(a_4, \mathcal{T}) = e(a_5, \tilde{g}), \qquad e(a_1 a_5, \mathcal{S}) = e(a_3, \tilde{g}).$$

2. Check that a_7 is a valid BB signature for public key $(a_1, \tilde{g}, a_4)$ where the message is the exponent of a_6 (base a_1). That is, verify the relation:

   $$e(a_4 a_6, a_7) = e(a_1, \tilde{g}).$$

3. Check that a_8 is a valid BB signature for public key $(a_1, \tilde{g}, a_6)$ on message m. That is, verify the relation:

   $$e(a_6 a_1^m, a_8) = e(a_1, \tilde{g}).$$

4. If all checks pass, accept; otherwise, reject.

Open(GSK, m, σ) On input any valid signature $\sigma = (a_1, \ldots, a_8)$ and tracing database D, $\mathcal{GM}$ may run the following algorithm to identify the signer. For each entry $Q_i \in D$, the group manager checks whether $e(a_4, \tilde{g}) = e(a_1, Q_i)$. If a match is found, then $\mathcal{GM}$ outputs $\mathcal{U}_i$ as the identity of the original signer.

Obviously, this algorithm takes time on the order of $2n$ pairings for groups with n members; we will show how to significantly reduce this running time in Section 6.

VerifyOpen$(\mathcal{GM}(GSK, m, \sigma, pk_i, Q_i), \mathcal{V}(GPK, m, \sigma, pk_i))$ First, $\mathcal{GM}$ checks that σ is a valid group signature; that is, $\mathsf{GroupVerify}(GPK, \sigma, m) = 1$. Next, $\mathcal{GM}$ checks that $\mathcal{U}_i$ is responsible for creating σ; that is, using tracing information $Q_i = \tilde{g}^{sk_i}$ from database D and $pk_i = (p_1, p_2)$, test that $e(p_1, Q_i) = p_2$. If both of these conditions hold, then $\mathcal{GM}$ proceeds to convince a verifier that $\mathcal{U}_i$ was responsible for σ. There are at least two ways to conduct this proof:

1. *Total Anonymity Revocation:* As a first option, $\mathcal{GM}$ can simply publish the tracing information Q. Then anyone can verify that the user with public key $pk = (p_1, p_2)$ must be responsible by checking that: (1) $e(p_1, Q) = p_2$, and (2) $e(a_1, Q) = e(a_4, \tilde{g})$.
2. *Partial Anonymity Revocation:* Alternatively, $\mathcal{GM}$ and a verifier can engage in a zero-knowledge proof of knowledge protocol that $\mathcal{GM}$ knows a value Q such that above two relations hold. Specifically, this is the following proof protocol: a proof of knowledge of a value $\alpha \in \mathbb{G}_2$ such that $e(p_1, \alpha) = p_2$ and $e(a_1, \alpha) = e(a_4, \tilde{g})$.

   This ZK proof of knowledge can be done efficiently as follows [1]. Here, $\mathcal{GM}$ is the prover and $\mathcal{V}$ is the verifier.

   (a) $\mathcal{V}$ selects a random challenge $c \in \mathbb{Z}_p$ and sends $C = \text{PedCom}(c)$ to $\mathcal{GM}$.
   (b) $\mathcal{GM}$ selects a random $r \in \mathbb{Z}_p$ and sends $(t_1, t_2) = (e(p_1^r, \tilde{g}), e(a_1^r, \tilde{g}))$ to $\mathcal{V}$.
   (c) $\mathcal{V}$ sends c, along with the opening of commitment C.
   (d) $\mathcal{GM}$ verifies that C opens to c and, if so, sends $s = (\tilde{g}^{sk})^c \tilde{g}^r$ to $\mathcal{V}$.
   (e) $\mathcal{V}$ accepts if and only if: (1) $e(p_1, s) = (p_2)^c t_1$ and (2) $e(a_1, s) = e(a_4, \tilde{g})^c t_2$.

   This proof can be made non-interactive by using either the Fiat-Shamir [26] or Fischlin transformations [27]. In our proof of Theorem 5.1, we will focus on the *interactive* version of this protocol.

Towards a Concurrent Join *Protocol.* In the above description we have specified that the group manager runs the Join protocol sequentially with the different users. The reason for this is technical, i.e., to prove security we require that the users' secrets keys sk_i are extractable from them. To this end we require the users to commit to their secret key and then prove knowledge of them. If one uses the standard proof of knowledge protocol for the latter, extracting the users' secret keys requires rewinding of the users. It is well known that if these proofs of knowledge protocols are run concurrently with many users, then extracting all the secret keys can take time exponential in the number of users. So, one way to prevent this is to require that the group manager run all the Join protocols sequentially. However, there are alternatives which allow for concurrent execution of these proofs and thus also of the Join protocol.

First of all, one could require the group manager to run the protocol concurrently only with a limited numbers of users, i.e., by defining time intervals within which the group manager runs the protocol concurrently with a logarithmic number of users and enforcing a time-out if a protocol does not finish within this time interval.

Second, one can apply one of the various construction that transform a standard proof of knowledge protocol (or Σ-protocol) into one that can be executed concurrently.

1. *Common random reference string.* Assuming that the parties have a common random reference string available, one can interpret this as the key for an encryption scheme such that the corresponding secret key is not know to any party. Alternatively, one could have a (distributed) trusted third party generate such a public key (cf. [24]). Then, the users would be required to verifiably encrypt their secret key sk_i under this reference public key (e.g., using the techniques of Camenisch and Damgård [17]). For extraction of the secret keys in the security proof, the reference string would need to be patched such that the simulator knows the reference decryption key and thus can extract the users' secret keys by simple decryption.

2. *Non-concurrent setup phase.* When having a common random reference string or a trusted third party is impractical, each user can instead generate their own public key and then prove knowledge of the corresponding secret key in a setup phase where non-concurrent execution can be guaranteed (e.g, because the user's part is run by an isolated smart card). Then, during the Join protocol, the user would verifiably encrypt her secret key sk_i under *her own* public key pk_i.

3. *Assuming random oracles for* Join *only.* A third alternative that comes to mind, in the random oracle model, is to apply Fischlin's results [27]. Fischlin recently presented a transformation for turning any standard proof of knowledge (or Σ-protocol) into a non-interactive proof in the random oracle model that supports an online extractor (i.e., no rewinding).

Theorem 5.1 *In the standard model, the above group signature scheme is correct and secure (i.e., it realizes $\mathcal{F}_{gs}$ from Section 2) under the Strong LRSW, the q-EDH, and the Strong SXDDH assumptions.*

Another way of interpreting Theorem 5.1 is to say that our scheme is secure for any SXDDH groups where the best discrete-logarithm algorithms are generic, exponential-time algorithms. This is the current situation for many elliptic curve implementations.

Proof. Our goal is to show that for every adversary $\mathcal{A}$ and environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that $\mathcal{Z}$ cannot distinguish whether it is interacting in the real world with $\mathcal{A}$ or the ideal world with $\mathcal{S}$. The proof is structured in two parts. First, for arbitrary fixed $\mathcal{A}$ and $\mathcal{Z}$, we describe a simulator $\mathcal{S}$. Then, we argue that $\mathcal{S}$ satisfies our goal.

Recall that the simulator interacts with the ideal functionality $\mathcal{F}_{gs}$ on behalf of all corrupted parties in the ideal world, and also simulates the real-world adversary $\mathcal{A}$ towards the environment. $\mathcal{S}$ is given black-box access to $\mathcal{A}$. In our description, $\mathcal{S}$ will use $\mathcal{A}$ to simulate conversations with $\mathcal{Z}$. Specifically, $\mathcal{S}$ will directly forward all messages from $\mathcal{A}$ to $\mathcal{Z}$ and from $\mathcal{Z}$ to $\mathcal{A}$.

The simulator will be responsible for handling several different operations within the group signature system. The operations are triggered either by messages from $\mathcal{F}_{gs}$ to any of the corrupted parties in the ideal system (and thus these messages are sent to $\mathcal{S}$) or when $\mathcal{A}$ wants to send any messages to honest parties. In our description, $\mathcal{S}$ will simulate the (real-world) honest parties towards $\mathcal{A}$. (Although, ideal honest parties actually exist outside of the control of $\mathcal{S}$, our simulator will intercept and change these messages before passing them onto the real-world adversary $\mathcal{A}$.)

Finally, we assume that when a signature is created, it becomes public information. Likewise, whenever a signature is opened, the corresponding identity is announced to all. (However, each user may still require individual proof from the $\mathcal{G}M$ that this identity is correct.)

Notation: The simulator $\mathcal{S}$ may need to behave differently depending on which parties are corrupted. There are two parties of interest: the group manager and a user. We adopt previous notation [16] for this: a capital letter denotes that the corresponding party is not corrupted and a small letter denotes that it is. For example, by "Case (Gu)" we refer to the case where the group manager is honest, but the user is corrupted.

We will refer to a user as $\mathcal{U}_i$ and a user's public key as pk_i. We assume throughout that a party in possession of one of these two identifiers is also in possession of the other.

We now describe how the simulator $\mathcal{S}$ behaves. Intuitively, when the group manager is corrupt, $\mathcal{S}$ will sign messages for whatever user $\mathcal{F}_{gs}$ tells it. When the group manager is honest, however, $\mathcal{S}$ will be asked to sign messages (and then later open them to) unknown users. In this case, $\mathcal{S}$ will sign all messages using the same secret key, which we denote sk^*. Then, whenever $\mathcal{S}$ is told to open this signature to a particular user later revealed by $\mathcal{F}_{gs}$, it will fake the corresponding proof.

Non-Adaptive Setup: Each party that $\mathcal{S}$ corrupts reports to $\mathcal{F}_{gs}$ that it is corrupted. The global parameters $\mathsf{BilinearSetup}(1^k) \to (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g}) = params$, where $\mathbb{G}_1 = \langle g \rangle$ and $\mathbb{G}_2 = \langle \tilde{g} \rangle$, are broadcast to all parties.

Simulation of the Real World's Setup: The group manager has an associated key pair (GPK, GSK). Regardless of the honesty of the group manager, $\mathcal{S}$ sets up any public parameters needed later for the registration of a user's key in Join (e.g., the hash function used in the Fischlin transformation).

Case (g): If the group manager is corrupted, then $\mathcal{S}$ receives the group key GPK from $\mathcal{A}$.

Case (G): If the group manager is honest, then $\mathcal{S}$ runs the $\mathsf{GroupSetup}$ algorithm to generate a group public key GPK which it then gives to $\mathcal{A}$. Note that in this case $\mathcal{S}$ knows the corresponding secrets and the relation of the Pedersen commitment bases. (Although, an ideal group manager exists outside of $\mathcal{S}$, the simulator will internally act as a real-world manager toward $\mathcal{A}$.)

Simulation of Honest Parties' Setup: Each party must have an associated key pair (pk_i, sk_i).

Case (u): If user $\mathcal{U}_i$ is corrupted, then $\mathcal{S}$ receives the user's public key pk_i from $\mathcal{A}$.

Case (U): If $\mathcal{U}_i$ is honest, then $\mathcal{S}$ runs the UserKeyGen algorithm to generate a public key pk_i which it then gives to $\mathcal{A}$. (Although, ideal honest parties exist outside of $\mathcal{S}$, the simulator will internally create real-world public keys for them.)

Simulation of the Join Protocol: In this operation, a user asks the group manager, via $\mathcal{F}_{gs}$, if it can join the group and receives an answer bit.

Case (gu): If both the group manager and the user are corrupt, then $\mathcal{S}$ does nothing.

Case (Gu): The group manager is honest, but the user is corrupt. Then, $\mathcal{A}$ will start by sending $\mathcal{S}$ the public key $pk = (p_1, p_2)$ and tracing information Q associated with some corrupt user $\mathcal{U}_i$. $\mathcal{S}$ will verify the tracing information by checking that $e(p_1, Q) = p_2$. If this check does not pass, $\mathcal{S}$ returns an error message to the corrupt user and ends the Join protocol. Otherwise, $\mathcal{S}$ stores the pair (pk, Q) in a database D. Now $\mathcal{S}$, acting as the honest group manager with knowledge of GSK, executes the remainder of the real-world Join protocol with $\mathcal{A}$, exiting with an error message when necessary according to the protocol. If $\mathcal{S}$ does not output an error message, then $\mathcal{S}$ submits ($\mathcal{U}_i$,"enroll") to $\mathcal{F}_{gs}$.

Case (gU): The group manager is corrupt, but the user is honest. $\mathcal{S}$ will be triggered in this case by $\mathcal{F}_{gs}$ asking if some honest user $\mathcal{U}_i$ may enroll. $\mathcal{S}$ will internally simulate a real-world version of $\mathcal{U}_i$ towards $\mathcal{A}$ using the key pair $\mathcal{S}$ generated for $\mathcal{U}_i$ during the user setup phase. If $\mathcal{A}$ stops before the end of the protocol, $\mathcal{S}$ returns the answer "no" to $\mathcal{F}_{gs}$. If the CL$^+$ signature obtained by $\mathcal{S}$ during step 2 of the Join protocol verifies, then $\mathcal{S}$ records this certificate and returns "yes" to $\mathcal{F}_{gs}$. Otherwise, it returns "no".

Case (GU): If both the group manager and the user are honest, then $\mathcal{S}$ does nothing.

Simulation of the GroupSign Operation: Let id be a counter initialized to zero. In this operation, a user anonymously obtains a signature on a message via $\mathcal{F}_{gs}$. When an honest member of the group requests to sign a message m, $\mathcal{F}_{gs}$ will forward ("sign",m) to $\mathcal{S}$. When $\mathcal{A}$ outputs a real-world signature, $\mathcal{S}$ will be responsible for translating it into the ideal world.

Here, we denote by sk^* the special signing key that $\mathcal{S}$ uses to sign all messages, for *all* honest parties when the group manager is honest.

Case (u): The user is corrupt. When $\mathcal{A}$ outputs a valid signature $\sigma = (a_1, \ldots, a_8)$ on message m, $\mathcal{S}$ tests if it is a (partial) re-randomization of any previous signature; that is, for all signatures $(b_1, \ldots, b_8)$ on message m in L, test if $a_7 = b_7$ (this corresponds to the value $\tilde{g}^{1/(sk+v)}$). If any match is found, $\mathcal{S}$ takes no further action.

However, when no match is found, $\mathcal{S}$ must register the signature with $\mathcal{F}_{gs}$. To do so, $\mathcal{S}$ must first discover the signer of σ. For every registered user, $\mathcal{S}$ uses the tracing information in database D to check if $e(a_1, Q_i) = e(a_4, \tilde{g})$. Suppose a match is found for some $Q_i = \tilde{g}^{sk_i}$.

1. If $sk_i = sk^*$, then the simulation has failed. $\mathcal{S}$ aborts and outputs "Failure 2".
2. If $sk_i \neq sk^*$ and $\mathcal{U}_i$ is honest, the simulation has failed. $\mathcal{S}$ aborts and outputs "Failure 3".
3. If $\mathcal{U}_i$ is corrupted, then $\mathcal{S}$ records $(\mathcal{U}_i, \sigma, m, id)$ in L, and sends $(\text{"sign"}, m, id)$ on behalf of corrupt $\mathcal{U}_i$ to $\mathcal{F}_{gs}$. $\mathcal{S}$ increments the counter id.

If no match for any registered user was found, then:

- Subcase (gu): If the group manager is corrupt, $\mathcal{S}$ records (corrupt-$\mathcal{G}M, \sigma, m, id$) in L, and sends ("sign", m, id) on behalf of corrupt-$\mathcal{G}M$ to $\mathcal{F}_{gs}$. $\mathcal{S}$ increments the counter id.
- Subcase (Gu): If the group manager is honest, the simulation has failed. $\mathcal{S}$ aborts and outputs "Failure 4".

Case (gU): The group manager is corrupt, but the user is honest. Since the group manager is corrupt, $\mathcal{F}_{gs}$ additionally tells $\mathcal{S}$ the identity $\mathcal{U}_i$ of the honest user requesting a signature. Then $\mathcal{S}$ generates a real-world group signature σ for the simulated $\mathcal{U}_i$, using that user's certificate (obtained during Join) and that user's secret key (which $\mathcal{S}$ created during the user setup phase). $\mathcal{S}$ records this entry $(\mathcal{U}_i, \sigma, m, id)$ in an internal database L. Finally, $\mathcal{S}$ provides $\mathcal{A}$ with the real-world signature (σ, m), and returns the "ideal signature" id to $\mathcal{F}_{gs}$ and increments id.

Case (GU): Both the group manager and the user are honest. As stated above, $\mathcal{S}$ is triggered by a request ("sign",m) from $\mathcal{F}_{gs}$. This time the ideal-world identity of the honest user is not known to $\mathcal{S}$. However, $\mathcal{S}$ still needs to provide $\mathcal{A}$ with *some* group signature, thus it proceeds as follows. $\mathcal{S}$ generates a real-world group signature σ using the secret key GSK of the group manager (which $\mathcal{S}$ created during the group setup phase) and the secret key of the first honest group member sk^* that it simulates towards $\mathcal{A}$ (which $\mathcal{S}$ also created during the user setup phase). Since all signatures are considered public information, $\mathcal{S}$ must forward the values (σ, m) to $\mathcal{A}$. As before, $\mathcal{S}$ records the entry $(?, \sigma, m, id)$ in an internal database L. Finally, $\mathcal{S}$ returns the "ideal signature" id to $\mathcal{F}_{gs}$ and increments id.

Simulation of the GroupVerify Operation: The simulator does not take any action on this operation. $\mathcal{A}$ will be able to verify all real-world signatures within its view itself. Furthermore, $\mathcal{F}_{gs}$ verifies signatures for honest users without informing $\mathcal{S}$.

Simulation of the Open Operation: The simulator is triggered on this operation in a variety of ways. There are two parties to consider: the group manager and the user requesting the opening (i.e., the verifier).

On the request ("open",σ, m) from a corrupted user, $\mathcal{S}$ first runs its GroupSign algorithm for receiving (σ, m) from $\mathcal{A}$.

Case (gu): Both the group manager and the verifier are corrupted. $\mathcal{S}$ does nothing.

Case (gU): The group manager is corrupted, but the verifier is honest. $\mathcal{F}_{gs}$ asks $\mathcal{S}$ (as the corrupted group manager) if it may open the ideal-world tuple $(\mathcal{U}_i, m, id)$. (Recall that if $\mathcal{U}_i =$ corrupt-$\mathcal{G}M$, then $\mathcal{F}_{gs}$ refuses to open the signature.) $\mathcal{S}$ searches its database L for an entry $(\mathcal{U}_j, \sigma, m, id)$, where σ is a real-world signature on m for some user $\mathcal{U}_j$. Since the id's are unique, only one such entry will exist. Next, $\mathcal{S}$, acting as an honest, real-world verifier toward $\mathcal{A}$, engages $\mathcal{A}$ in the VerifyOpen protocol with common input $(\mathcal{U}_i, m, \sigma)$. Here we choose the interactive, *partial anonymity revocation* VerifyOpen protocol. If $\mathcal{S}$, as an honest verifier, does not accept this proof, then $\mathcal{S}$ tells $\mathcal{F}_{gs}$ to refuse to open this signature. If $\mathcal{S}$ accepts this verification from $\mathcal{A}$ and $\mathcal{U}_i = \mathcal{U}_j$, then $\mathcal{S}$ tells $\mathcal{F}_{gs}$ to open the signature. Finally, if $\mathcal{S}$ accepts this verification and yet $\mathcal{U}_i \neq \mathcal{U}_j$, then our simulation has failed. $\mathcal{S}$ aborts and outputs "Failure 1".

Case (Gu): The group manager is honest, but the verifier is corrupted. Since the verifier is corrupted, it may ask about the openings of any signatures it likes. (For example, it may re-randomize a valid signature, etc.) Suppose $\mathcal{A}$, acting as a corrupt verifier, requests an opening on (m, σ). The first thing that $\mathcal{S}$ does is to check if σ is a valid group signature (according to the real-world verification algorithm) on m under the group public key GPK. If it is not, then $\mathcal{S}$ returns an error message, $\perp$, to $\mathcal{A}$. Otherwise, $\mathcal{S}$ proceeds.

Now, $\mathcal{S}$ must figure out which user, if any, is responsible for σ. First, $\mathcal{S}$ uses its tracing database to test if any registered user is responsible for σ. Specifically for $\sigma = (a_1, \ldots, a_8)$ and tracing information Q_j, $\mathcal{S}$ checks if $e(a_1, Q_j) = e(a_4, \tilde{g})$.

If σ opens to some registered user $\mathcal{U}_j$, then there are three cases.

1. $\mathcal{U}_j$ is corrupted. This is not considered a forgery. $\mathcal{S}$ honestly runs the real-world VerifyOpen protocol with $\mathcal{A}$ on common inputs $(\mathcal{U}_j, m, \sigma)$. This transaction can be completely simulated by $\mathcal{S}$ without involving $\mathcal{F}_{gs}$.
2. $\mathcal{U}_j$ is honest, and $sk_j = sk^*$. Here, $\mathcal{S}$ needs to further differentiate if σ is a forgery or merely a re-randomization of a previous signature. (Observe that the first part of our signatures may be re-randomized.) To do this, $\mathcal{S}$ searches database L and complies a list of all entries $(?, \sigma_i, m, id_i)$ containing message m. Next, $\mathcal{S}$ checks whether $\sigma = (a_1, \ldots, a_8)$ is derived from any $\sigma_i = (b_{1,i}, \ldots, b_{8,i})$ by checking if $a_7 = b_{7,i}$.
   (a) If $\mathcal{S}$ finds a match for some entry i, then it sends the request ("open", m, id_i) to $\mathcal{F}_{gs}$. Suppose $\mathcal{F}_{gs}$ returns the identity $\mathcal{U}_x$. Now, $\mathcal{S}$ must prove this opening to $\mathcal{A}$. $\mathcal{S}$ did not know who the ideal-world signer was at the time it created σ under sk^* (recall that our simulator creates all signatures using sk^*), thus it must now fake a real-world VerifyOpen opening towards $\mathcal{A}$. That is, $\mathcal{S}$ must open $\sigma = (a_1, \ldots, a_8)$ to user $\mathcal{U}_x$ with $pk_x = (h_x, e(h_x, \tilde{g})^{sk_x})$.
   $\mathcal{S}$ simulates the interactive, *partial anonymity revocation* VerifyOpen proof as follows [1]. Let $pk_x = (p_1, p_2)$. Recall that this is proof of knowledge of a value $\alpha \in \mathbb{G}_2$ such that $e(p_1, \alpha) = p_2$ and $e(a_1, \alpha) = e(a_4, \tilde{g})$.
      i. $\mathcal{A}$ selects a random challenge $c \in \mathbb{Z}_p$ and sends $C = \mathrm{PedCom}(c)$ to $\mathcal{S}$.
      ii. $\mathcal{S}$ selects a random $r \in \mathbb{Z}_p$ and sends $(t_1, t_2) = (e(p_1^r, \tilde{g}), e(a_1^r, \tilde{g}))$ to $\mathcal{A}$.
      iii. $\mathcal{A}$ sends c along with the opening of commitment C.
      iv. $\mathcal{S}$ verifies that C opens to c and, if so, sends $s = (\tilde{g}^{sk_x})^c \tilde{g}^r$ to $\mathcal{A}$.
      v. $\mathcal{A}$ accepts if and only if: (1) $e(p_1, s) = (p_2)^c t_1$ and (2) $e(a_1, s) = e(a_4, \tilde{g})^c t_2$.

   (b) If $\mathcal{S}$ does not find a match for any entry i, then $\mathcal{A}$ has succeeded in a forgery against user $\mathcal{U}_j$ with $sk_j = sk^*$. The simulation fails. $\mathcal{S}$ aborts and outputs "Failure 2".
3. $\mathcal{U}_j$ is honest, and $sk_j \neq sk^*$. $\mathcal{S}$ immediately knows σ is a forgery, because $\mathcal{S}$ signs for all honest users with the key sk^*. The simulation fails. $\mathcal{S}$ aborts and outputs "Failure 3".

If σ does not open to any registered user, then $\mathcal{A}$ has succeeded in creating a valid group signature for a non-registered user. That is, for all tracing information Q_i known to $\mathcal{S}$ and letting $\sigma = (a_1, \ldots, a_8)$, we have $e(a_1, Q_i) \neq e(a_4, \tilde{g})$. In this case, $\mathcal{S}$ aborts and outputs "Failure 4".

Case (GU): Both the group manager and the verifier are honest. $\mathcal{S}$ does nothing. $\mathcal{S}$ will not even know that this transaction has taken place.

This ends our description of simulator $\mathcal{S}$. It remains to show that $\mathcal{S}$ works; that is, under the Strong LRSW, the q-EDH, and the Strong SXDDH assumptions, the simulator will not abort, except with negligible probability, and that the environment will not be able to distinguish between the real and ideal worlds.

Claim 5.2 *Conditioned on the fact that $\mathcal{S}$ never aborts, $\mathcal{Z}$ cannot distinguish between the real world and the ideal world under the Strong LRSW, the q-EDH, and the Strong SXDDH assumptions.*

Proof. To see this, let us explore each operation. First, we observe that in GroupSetup and UserKeyGen, the simulator $\mathcal{S}$ performs all key generation operations as the respective players in the real world would do. The simulator never deviates from the actions of any honest player during Join and it need not take any action during GroupVerify. In the real world, anyone may verify a signature autonomously. The remaining operations to consider are GroupSign and VerifyOpen.

Let us begin with GroupSign. In this operation, $\mathcal{S}$ only needs to take action when it must translate an ideal-world signature into a real-world signature, and vice versa, for $\mathcal{A}$. When the user is corrupted, $\mathcal{S}$ submits "sign" requests for $\mathcal{A}$ whenever it outputs a new signature. There is nothing here for $\mathcal{A}$ to observe.

When the user is honest, however, then $\mathcal{S}$ must generate real-world signatures towards $\mathcal{A}$. When the group manager is corrupted, then $\mathcal{F}_{gs}$ tells $\mathcal{S}$ which user is signing the message, and thus $\mathcal{S}$ may perfectly generate a real-world signature for $\mathcal{A}$. $\mathcal{S}$ is only forced to deviate in case (GU) when it must simulate both the honest group manager and honest signer towards $\mathcal{A}$. The problem is that $\mathcal{S}$ does not know which user is requesting a signature on some message m; thus $\mathcal{S}$ always signs with the same honest user key sk^*. By Lemma 5.3, we know that neither $\mathcal{A}$ nor $\mathcal{Z}$ can distinguish between this homogeneous, ideal-world distribution of signatures and the heterogeneous, real-world distribution.

Now, it remains to consider VerifyOpen. In this operation, $\mathcal{S}$ only takes action when one of the two parties is corrupted. In the case (gU), $\mathcal{S}$ behaves exactly as an honest verifier would towards $\mathcal{A}$; that is, $\mathcal{S}$ finds the (single) σ associated with id, and acts as an honest verifier towards $\mathcal{A}$. We will later argue that it does not abort, due to Failure 1, in this step.

The case (Gu), however, is more complicated. Suppose $\mathcal{S}$ is being asked by $\mathcal{A}$ to open (m, σ). If σ opens to a corrupted user or does not open to any registered user, then $\mathcal{S}$ behaves exactly as an honest $\mathcal{GM}$ would. However, what happens when σ opens to an honest user? We will later argue that $\mathcal{S}$ is not forced to abort due to Failures 2, 3, or 4. Even conditioned on this fact, $\mathcal{S}$ will almost always be forced to deviate since it signed using key sk^* for all honest users and now must open the signatures to whichever honest party $\mathcal{F}_{gs}$ dictates. Suppose $\mathcal{S}$ is told to open $\sigma = (a_1, \ldots, a_8)$ to some honest user $\mathcal{U}_i$, where $sk_i \neq sk^*$, then $\mathcal{S}$ must fake the (interactive, partial anonymity revocation) VerifyOpen protocol toward $\mathcal{A}$. $\mathcal{S}$ succeeds in doing this, in the usual way, by requiring $\mathcal{A}$ to commit to his challenge and then resetting $\mathcal{A}$ after seeing the challenge. That is, after seeing $c \in \mathbb{Z}_p$, $\mathcal{S}$ chooses a random value $s \in \mathbb{G}_2$ and computes $t_1 = e(p_1, s)/p_2^c$ and $t_2 = e(a_1, s)/e(a_4, \tilde{g})^c$, where $pk_i = (p_1, p_2)$. Now $\mathcal{S}$ rewinds $\mathcal{A}$ to right after it sent a commitment to c, then sends (t_1, t_2), receives c with a valid opening, and returns the response s. Indeed, $\mathcal{S}$ only fails in this step in the unlikely event that $\mathcal{A}$ is able to break the binding property of the Pedersen commitments (i.e., CDH in $\mathbb{G}_1$).

□

This concludes our proof of Claim 5.2. It remains to show that, except with negligible probability, $\mathcal{S}$ will not abort. Recall that $\mathcal{S}$ may abort under the following conditions:

- Failure 1: $\mathcal{A}$ *breaks exculpability.* We argue that it is not possible for a dishonest group manager to falsely open a signature; i.e., $\mathcal{A}$ is not able to successfully complete the Verify-Open protocol with $\mathcal{S}$ on common input $(\mathcal{U}_i, m, \sigma)$ where $\mathcal{U}_i$ is not he real signer. Here, the simulation fails, because $\mathcal{F}_{gs}$ will only open signatures honestly.

  We now argue that, for a given VerifyOpen instance $(\mathcal{U}_i, m, \sigma)$, an adversary that can cause Failure 1 with probability ε can be used to break the Co-CDH assumption with probability $\geq (\varepsilon - 1/p)^2$. (Recall from Section 3.2 that Co-CDH is implied by the Strong SXDDH assumption.) On Co-CDH input $(g, \tilde{g}, g^x, \tilde{g}^y)$, the goal is to compute $\tilde{g}^{xy}$ and the simulator proceeds as follows.

  1. Step 1: $\mathcal{S}$ initiates the VerifyOpen protocol with $\mathcal{A}$ on input $(\mathcal{U}_i, m, \sigma)$, setting $pk_i = (g^z, e(g^{zx}, \tilde{g}^y))$, for random $z \in \mathbb{Z}_p$, and computing σ as a valid signature on m for the user with sk^*.
  2. Step 2: $\mathcal{S}$ commits to all zeros, as $C = \text{PedCom}(0^{|p|})$.
  3. Step 3: After receiving (t_1, t_2) from $\mathcal{A}$, $\mathcal{S}$ using its knowledge of the relation of the Pedersen public parameters to fake the openings as:
     - selects a random challenge $c_1 \in \mathbb{Z}_p$, opens C to c_1, and obtains $\mathcal{A}$'s response s_1.
     - rewinds $\mathcal{A}$, selects a different random challenge $c_2 \in \mathbb{Z}_p$, opens C to c_2, and obtains $\mathcal{A}$'s response s_2.
  4. Step 4: $\mathcal{S}$ computes and outputs $(s_1/s_2)^{1/(c_1 - c_2)}$ (which hopefully corresponds to $\tilde{g}^{xy}$).

  In Step 1, the adversary cannot tell that it was given a signature under sk^* instead of sk_i due to Lemma 5.3. The fake openings in Step 4 are perfectly indistinguishable from an honest opening due to the perfect hiding property of Pedersen commitments. If both $((t_1, t_2), c_1, s_1)$ and $((t_1, t_2), c_2, s_2)$ are valid transcripts, then $\mathcal{S}$ outputs $\tilde{g}^{xy}$ in Step 4 with probability $\geq (\varepsilon - 1/p)^2$. Our bound of $(\varepsilon - 1/p)^2$ comes from the well-known Reset Lemma [10], where the advantage of $\mathcal{A}$ was given as ε and the size of the challenge set is p.

- Failure 2: $\mathcal{A}$ *creates a forgery against the honest user with* sk^*. Here, $\mathcal{A}$ produces a signature $\sigma = (a_1, \ldots, a_8)$ and a message m such that $\mathsf{GroupVerify}(GPK, \sigma, m) = 1$, σ opens to $\mathcal{U}^*$ (i.e., $e(a_1, Q^*) = e(a_4, \tilde{g})$), and yet $\mathcal{S}$ never gave $\mathcal{A}$ this user's signature on m. This scenario occurs with only negligible probability under the q-EDH assumption, regardless of whether the group manager is corrupted.

  Recall that q-EDH takes as input $(h, h^x, \ldots, h^{x^q}, \tilde{h}, \tilde{h}^x, \ldots, \tilde{h}^{x^q})$, where $\langle h \rangle = \mathbb{G}_1$ and $\langle \tilde{h} \rangle = \mathbb{G}_2$, and the goal is to produce a tuple $(c, a, a^v, \tilde{h}^{1/(x+v)}, \tilde{h}^{1/(v+c)})$ for any $a \in \mathbb{G}_1$ and any $v, c \in \mathbb{Z}_p^*$. This reduction follows a similar technique of Boneh and Boyen [12].

  Let τ be the number of honest users in the system. Let q_S be the number of signature queries that $\mathcal{A}$ requests. Finally, let i be a counter initialized to zero. When $\mathcal{A}$ succeeds with probability ε, then $\mathcal{B}$ solves the q-EDH problem (where $q = q_S - 1$) with probability ε/τ. $\mathcal{B}$ proceeds as follows:

  1. *Setup:* $\mathcal{B}$ must establish the global parameters and key generation.
     (a) Select q_S random elements $v_i \in \mathbb{Z}_p^*$; denote this set $V = \{v_1, \ldots, v_{q_S}\}$.

(b) Using the q-EDH input and the set V, compute:

$$g \leftarrow \prod_{i=0}^{q-1} X_{(1,i)}^{\gamma_i} = h^{f(x)} \quad , \quad \tilde{g} \leftarrow \prod_{i=0}^{q-1} X_{(2,i)}^{\gamma_i} = \tilde{h}^{f(x)}$$
$$g^x \leftarrow \prod_{i=1}^{q} X_{(1,i)}^{\gamma_{i-1}} = h^{xf(x)} \quad , \quad \tilde{g}^x \leftarrow \prod_{i=1}^{q} X_{(2,i)}^{\gamma_{i-1}} = \tilde{h}^{xf(x)}$$

where $X_{(1,i)} = h^{x^i} \in \mathbb{G}_1$, $X_{(2,i)} = \tilde{h}^{x^i} \in \mathbb{G}_2$, and $\gamma_0, \ldots, \gamma_{q-1} \in \mathbb{Z}_p$ are the coefficients of the polynomial $f(x) = \prod_{i=1}^{q-1}(x + v_i) = \sum_{i=0}^{q-1} \gamma_i x^i$. (We are safe to assume $f(x) \neq 0$, since otherwise some $v_i \in V$ is $-v_i = x$, and thus $\mathcal{B}$ can easily break q-EDH.)

(c) Output $(g, \tilde{g})$ as the public parameters for the group signature scheme, and $GPK = (\tilde{S}, \tilde{T}) = (\tilde{g}^s, \tilde{g}^t)$ on behalf of the group manager. If $\mathcal{GM}$ is corrupt, GPK is given to $\mathcal{S}$ by $\mathcal{A}$. Setup all remaining keys and parameters as $\mathcal{S}$ would normally do

(d) Guess which of the τ honest users $\mathcal{A}$ will attack. Give this user $\mathcal{U}^*$ the public key $pk^* = (g^r, e(g^r, \tilde{g}^x))$, for random $r \in \mathbb{Z}_p$. (Logically this assigns the user's secret key as $sk^* = x$.)

(e) Obtain group certificates for all honest users; $\mathcal{B}$ fakes the proof of knowledge of sk^* using any of the techniques discussed in Section 5 (Join). Finally, $\mathcal{B}$ submits the *correct* tracing information, $Q^* = \tilde{g}^{sk^*} = \tilde{g}^x$, for this user.

2. *Signing:* When $\mathcal{A}$ requests a signature from a user not associated with $sk^* = x$, sign as normal. Now, when $\mathcal{A}$ asks for a group signature on $m \in \mathbb{Z}_p^*$ from the honest user associated with secret key $sk^* = x$ and certificate $(f_1, \ldots, f_5)$, do:

(a) Using the q-EDH input and the next element $v_i \in V$, compute:

$$\sigma_i \leftarrow \prod_{j=0}^{q-2} X_{(2,j)}^{\delta_j} = \tilde{h}^{f_i(x)} = (\tilde{h}^{f(x)})^{1/(x+v_i)} = \tilde{g}^{1/(x+v_i)}$$

where $\delta_0, \ldots, \delta_{q-2}$ are the coefficients of the polynomial $f_i(x) = f(x)/(x + v_i) = \sum_{j=0}^{q-2} \delta_j x^j$.

(b) Select a random $r \in \mathbb{Z}_p$. Output the group signature on m as

$$(f_1^r, f_2^r, f_3^r, f_4^r, f_5^r, (f_1)^{rv_i}, \sigma_i, \tilde{g}^{1/(v_i+m)}).$$

(c) Finally, update the counter as $i = i + 1$.

3. *Opening:* $\mathcal{B}$ honestly executes the VerifyOpen protocol with $\mathcal{A}$.

4. *Output:* Suppose $\mathcal{A}$ produces a valid signature $\sigma' = (a_1, \ldots, a_8)$ for a *new* message $m' \in \mathbb{Z}_p^*$ for the user with key $sk^* = x$. Then $\mathcal{B}$ outputs (m', a_1, a_6, a_7, a_8) to solve the q-EDH problem.

It is easy to observe that $\mathcal{B}$ perfectly simulates the group signature world for $\mathcal{A}$. $\mathcal{B}$ has probability $1/\tau$ of choosing which honest user $\mathcal{A}$ will forge against. Thus, when $\mathcal{A}$ succeeds with probability ε, then $\mathcal{B}$ solves the q-EDH problem with probability ε/τ.

- Failure 3: $\mathcal{A}$ *creates a forgery against a user with* $sk_j \neq sk^*$*.* Proof that this failure occurs with only negligible probability follows directly from that of Failure 2. Indeed, $\mathcal{A}$ has strictly less information at its disposal; that is, $\mathcal{A}$ never sees real signatures under key sk_j.

- Failure 4: $\mathcal{A}$ *creates a valid signature for a non-registered user.* In this case, $\mathcal{A}$ produces a signature-message pair (σ, m) such that $\mathsf{GroupVerify}(GPK,\ \sigma, m) = 1$ and yet it cannot be opened by $\mathcal{S}$ to any registered user. We now argue that this is not possible under the Strong LRSW assumption, except with negligible probability. Suppose we are given $(g, \tilde{g}, \tilde{g}^s, \tilde{g}^t)$ as the Strong LRSW input.

  Instead of running GroupSetup, let the public parameters $g, \tilde{g} \in params$ and the public key $GPK = (\tilde{S}, \tilde{T}) = (\tilde{g}^s, \tilde{g}^t)$. During the UserKeyGen operation, for any honest users, $\mathcal{S}$ queries the Strong LRSW oracle $O_{\tilde{S},\tilde{T}}$ on a random $sk_i \in \mathbb{Z}_p$ to obtain a membership certificate $(a, a^t, a^{s+st(sk_i)})$, for any $a \in g$. (This tuple is, in fact, a CL signature on sk_i [19].) $\mathcal{S}$ now uses sk_i as the secret key for this honest user.

  When $\mathcal{S}$ is asked to execute Join with an *honest* user, $\mathcal{S}$ simply finds the corresponding CL signature and uses it to output the certificate $(a, a^t, a^{s+st(sk_i)}, (a)^{sk_i}, (a^t)^{sk_i})$. When $\mathcal{S}$ is asked to execute Join with a *corrupted* user, $\mathcal{S}$ extracts the user's secret key sk_j using any of the techniques discussed in Section 5 (Join), queries the Strong LRSW oracle on input sk_j, and uses the oracle's output to create a valid certificate for this corrupt user. Now, the adversary can sign any message for a corrupt user, and $\mathcal{S}$ can honestly respond to any GroupSign call for an honest user.

  Suppose that Failure 1 has occurred during VerifyOpen, meaning that $\mathcal{A}$ output a signature $\sigma = (a_1, \ldots, a_8)$ such that the following relations hold:

  $$e(a_1, \tilde{T}) = e(a_2, \tilde{g}), \quad e(a_4, \tilde{T}) = e(a_5, \tilde{g}), \quad e(a_1 a_5, \tilde{S}) = e(a_3, \tilde{g})$$

  and yet $\mathcal{S}$ did *not* query $O_{\tilde{S},\tilde{T}}$ on the corresponding secret key; that is, for all sk_i known to $\mathcal{S}$, we have $a_1^{sk_i} \neq a_4$. Then, $\mathcal{S}$ may output $(a_1, a_2, a_3, a_4, a_5)$ to break the Strong LRSW assumption.

Combining Claim 5.2 with the above arguments that $\mathcal{S}$ will not abort, except with negligible probability, concludes our main proof.

□

We end by proving a Lemma used in the above proof. Intuitively, this Lemma captures the anonymity of our signatures. In the below, the values $u_1, \ldots, u_\tau$ may be thought of as the secret keys of τ different honest users.

Lemma 5.3 (Anonymity of Signatures) *Suppose we have the group signature parameters from Section 5; that is, security parameter 1^k, params, and GPK. Suppose $u_1, \ldots, u_\tau$ are random elements of $\mathbb{Z}_p$. Let $O_{u_1,\ldots,u_\tau}(\cdot,\cdot)$ be an oracle that takes as input a message $m \in \mathbb{Z}_p^*$ and an index $1 \leq i \leq \tau$, and outputs a group signature $(a_1, \ldots, a_8)$ on m with user secret key u_i. Then, under the Strong SXDDH assumptions, for all probabilistic polynomial-time adversaries $\mathcal{A}$, the following value is negligible in k:*

$$|Pr[\mathcal{A}^{O_{u_1,u_2,\ldots,u_\tau}}(params,\, GPK,\, \{pk_i\}_{i\in[1,\tau]}) = 1] - Pr[\mathcal{A}^{O_{u_1,u_1,\ldots,u_1}}(params,\, GPK,\, \{pk_i\}_{i\in[1,\tau]}) = 1]|.$$

Proof. We argue in two stages. First, if $\mathcal{A}$ can distinguish between oracles $O_{u_1,u_2,\ldots,u_\tau}$ and $O_{u_1,u_1,\ldots,u_1}$, then we can create an adversary $\mathcal{B}$ that can distinguish between oracles O_{u_1,u_2} and O_{u_1,u_1}. Next, we show that adversary $\mathcal{B}$ can be used to break the Strong SXDDH assumption. Overall, if $\mathcal{A}$ succeeds with probability ε, then we can break Strong SXDDH with probability $\geq \varepsilon/\tau$.

Stage One. First, we make the simple hybrid argument that given $\mathcal{A}$, which can distinguish the signatures of τ distinct honest users from those of a single user, we can create an adversary $\mathcal{B}$ that can distinguish the signatures of only 2 distinct users from those of a single user. Indeed, by the hybrid argument, we know that if $\mathcal{A}$ distinguishes with probability ε, then for some $1 \leq \ell \leq \tau$, $\mathcal{A}$ can distinguish with probability $\geq \varepsilon/\tau$ between the oracle instantiated with ℓ u_1's followed by $\tau - \ell$ different seeds and the oracle instantiated with $\ell + 1$ u_1's followed by $\tau - \ell - 1$ different values. The obvious reduction follows; that is, the two oracles of $\mathcal{B}$ will be applied to this hybrid point for $\mathcal{A}$. $\mathcal{B}$ will then return whatever answer $\mathcal{A}$ does.

Stage Two. Now, we show that $\mathcal{B}$ can be used to create another adversary $\mathcal{C}$ that breaks Strong SXDDH. On Strong SXDDH input $(g, g^x, \tilde{g})$, the adversary $\mathcal{C}$ proceeds as follows:

1. Generate group public key GPK as $(\tilde{S}, \tilde{T}) = (\tilde{g}^s, \tilde{g}^t)$ for random $s, t \in \mathbb{Z}_p$. Give GPK to $\mathcal{B}$; store $GSK = (s, t)$. (Remember, anonymity only makes sense when the group manager is honest, so the adversary does not get to set these keys.)

2. Query Q_y on a random input, disregard all output except (h, h^y) for some $h \in \mathbb{G}_1$.

3. Generate the two user keys as $pk_1 = (g, e(g^x, \tilde{g}))$ for user $\mathcal{U}_1$ and $pk_2 = (h, e(h^y, \tilde{g}))$ for user $\mathcal{U}_2$. Give pk_1, pk_2 to $\mathcal{B}$. (This first key could be re-randomized away from the public parameters by choosing a random $r \in \mathbb{Z}_p$ and setting $pk_1 = (g^r, e(g^x, \tilde{g})^r)$. This has no effect on the remainder, and for clarity we omit it.)

4. When $\mathcal{B}$ requests a signature for index $i \in \{1, 2\}$ on $m \in \mathbb{Z}_p^*$, if $i = 1$, use $O_x(\cdot)$ to do:

   (a) Query $O_x(m)$ to obtain the output $(g^v, \tilde{g}^{1/(x+v)}, \tilde{g}^{1/(v+m)})$, where $v \in \mathbb{Z}_p^*$ is a fresh random value chosen by the oracle. Denote this output as $(f_6, \ldots, f_8)$.
   (b) Using $GSK = (s, t)$, compute the remaining parts of the group signature: $f_2 = g^t$, $f_3 = g^s(g^x)^{st}$, $f_4 = g^x$, and $f_5 = (g^x)^t$.
   (c) Select a random $r \in \mathbb{Z}_p^*$, and return the signature $(g^r, f_2^r, f_3^r, f_4^r, f_5^r, f_6^r, f_7, f_8)$.

   If $i = 2$, use oracle $O_y(\cdot)$ to do:

   (a) Query $Q_y(m)$ to obtain the output $(a, a^y, a^v, \tilde{g}^{1/(y+v)}, \tilde{g}^{1/(v+m)})$, where $a \in \mathbb{G}_1$ and $v \in \mathbb{Z}_p^*$ are fresh random values chosen by the oracle. Denote this output as $(f_1, f_4, f_6, \ldots, f_8)$.
   (b) Using $GSK = (s, t)$, compute the remaining parts of the group signature: $f_2 = a^t$, $f_3 = a^s(a^x)^{st}$, and $f_5 = (a^y)^t$.
   (c) Return the signature $(f_1, \ldots, f_8)$.

5. Eventually, $\mathcal{B}$ will attempt to distinguish whether he's been talking to oracle $O_{x,x}$ or oracle $O_{x,y}$. If $\mathcal{B}$ says that he's been talking to oracle $O_{x,x}$, then output 1 corresponding to "$x = y$". Otherwise, output 0 corresponding to "$x \neq y$".

It is easy to see that the stage 2 simulation is perfect; the output is always correct and perfectly distributed. Indeed, $\mathcal{C}$ and $\mathcal{B}$ will succeed with identical probabilities. This concludes our proof. We find that if any $\mathcal{A}$ can break the anonymity of our signatures with probability ε, then $\mathcal{A}$ can be used to break Strong SXDDH with probability at least ε/τ, where τ is the number of honest users in the system. □

6 Opening Signatures in Sublinear Time

The Open algorithm for the previous scheme takes $O(n)$ pairings for a signing group of n members. Practically, this precludes this scheme from being used for many applications with *large* groups. We now remedy this situation by explaining how to alter the basic scheme to allow for faster openings.

First, we present an Open algorithm with complexity logarithmic in the number of group members. This improvement requires no new assumptions from the Section 5 scheme, but does add two elements in $\mathbb{G}_1$ to the signature length. Next, we present a constant-time Open algorithm. The cost for this is that: (1) the signature length increases by one element in $\mathbb{G}_1$ and two elements in $\mathbb{G}_2$, and (2) a slightly stronger anonymity assumption is required.

As before, let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g})$, where $\mathbb{G}_1 = \langle g \rangle$ and $\mathbb{G}_2 = \langle \tilde{g} \rangle$, be the global parameters for the bilinear groups.

Open Algorithm with $O(\log n)$ Complexity from Trees. The intuition here is that group members are logically divided into a 2-level tree; then to revoke the anonymity of a signature, the group manager first locates the right branch and then the right leaf (user) for that branch. For a balanced tree, this results in a search time of $2(\log n)$. Now we present the details. During the Join protocol, the group manager secretly assigns each user to one of $(\log n)$ logical branches. Each branch is associated with a unique ID as a value $ID \in \mathbb{Z}_p^*$. Now, the group manager and the user run a protocol such that at the end the user obtains a CL$^+$ signature on the pair of messages (sk, ID) without learning its branch identity ID and the group manager does not learn the user's secret key sk. Following Camenisch and Lysyanskaya [19] this CL$^+$ signature would be of the following form for $GPK = (\tilde{g}^s, \tilde{g}^t, \tilde{g}^z, \tilde{g}^{tz})$, $GSK = (s, t, z)$, and some $a \in \mathbb{G}_1$:

$$(a_1, \ldots, a_7) = (a, a^t, a^{s+st(sk)+stz(ID)}, a^{sk}, a^{t(sk)}, a^{ID}, a^{tz(ID)})$$

This CL$^+$ signature would be used as the user's certificate. Let the user submit tracing information $Q_j = \tilde{g}^{sk_j}$ during the Join protocol as before. Then to open a group signature, the group manager now does: For each branch identity ID_i, check if $e(a_6, \tilde{g}) = e(a_1, \tilde{g}^{ID_i})$; then for each member of the matching branch, check if $e(a_4, \tilde{g}) = e(a_1, Q_j)$.

Under the DDH assumption in $\mathbb{G}_1$, a user's branch identity remains remains hidden from everyone except the group manager, so full anonymity is preserved. By the Strong LRSW assumption, a user cannot change which branch he is associated with, and thus the group manager will be able to find him (i.e., open the signature).

Theorem 6.1 *In the plain model, the above extension to the Section 5 scheme is correct and secure (i.e., it realizes $\mathcal{F}_{gs}$ from Section 2) assuming that the Strong LRSW, the q-EDH, and the Strong SXDDH assumptions.*

Open Algorithm with $O(1)$ Complexity from Encryption. The intuition here is to have the signer include an encryption of her identity under the group manager's encryption key as part of every signature. The trick is to do this in such a way that the *correctness* of the encryption is publically-verifiable, and yet, the *anonymity* of the signer is preserved.

Let $(eGPK, eGSK)$ be Elgamal encryption keys generated by the group manager, where $eGSK \in \mathbb{Z}_p$ and $eGPK = \tilde{g}^{eGSK}$. Then in addition to a regular signature from Section 5, a user would add an Elgamal encryption of their identity as the last three items:

$$\begin{aligned}\big(&Rand(Sign^{CL+}_{GSK}(sk; a), r),\ a^{vr}, Sign^{BB}_{sk}(v; \tilde{g}), Sign^{BB}_{v}(m; \tilde{g}), \qquad a^{cr}, Enc^{Elgamal}_{eGPK}(sk; \tilde{g}, c)\big)\\ &= \big(b, b^t, b^{s+st(sk)}, b^{sk}, b^{t(sk)}, b^v, \tilde{g}^{1/(sk+v)}, \tilde{g}^{1/(v+m)}, \qquad b^c, \tilde{g}^{sk+c}, (eGPK)^c\big).\end{aligned}$$

where $a^r = b \in \mathbb{G}_1$. Then during verification of the signature $\sigma = (a_1, \ldots, a_{11})$, in addition to the usual checks, a verifier must be sure that the ciphertext is correctly formed as:

$$e(a_1, a_{10}) = e(a_4, \tilde{g})e(a_9, \tilde{g}) \quad \text{and} \quad e(a_9, eGPK) = e(a_1, a_{11}).$$

Finally, the group manager may, at any time, open the signature by simply decrypting the last portion as $a_{10}/(a_{11})^{1/eGSK} = \tilde{g}^{sk}$, which reveals the user's identity. Recall that the group manager obtains this same "tracing information" from the user during the Join protocol.

Theorem 6.2 *In the plain model, the above extension to the Section 5 scheme is correct and secure (i.e., it realizes $\mathcal{F}_{gs}$ from Section 2) assuming that the Strong LRSW, the q-EDH, and (an extension of) the Strong SXDDH assumption.*

Under the SXDDH assumption, Elgamal encryption is secure in $\mathbb{G}_2$. The question here is whether or not these signatures remain anonymous. The extension of the Strong SXDDH assumption required for anonymity requires that the Strong SXDDH oracles O and Q, from Definition 4 in Section 3, produce enough information to produce the encryption. Specifically, we change the oracles as follows: Select a value $eGPK \in \mathbb{G}_2$ at random and give as input the adversary. Let $O'_x(\cdot)$ be an oracle that takes as input $m \in \mathbb{Z}_p^*$ and outputs $(g^r, g^{rx}, g^{rv}, \tilde{g}^{1/(x+v)}, \tilde{g}^{1/(v+m)}, g^{rc}, \tilde{g}^{sk+c}, eGPK^c)$ for a random $r, v, c \in \mathbb{Z}_p^*$. Then, we say that on input $(g, g^x, \tilde{g}, eGPK)$, the adversary cannot distinguish access to oracles $(O'_x(\cdot), O'_y(\cdot))$ from $(O'_x(\cdot), O'_x(\cdot))$.

The proof of Theorem 7.3 that Strong SXDDH is hard in generic groups can be modified to cover this extended version as well.

An Alternative $O(1)$ Implementation. Instead of using Elgamal encryption in the above, we could cut the signature length by one element in $\mathbb{G}_2$ by employing the recent cryptosystem of Boneh, Goh, and Nissim [14]. However, this would also require that we use *composite* bilinear groups where the subgroup decision problem is hard [14]; *this in addition to our other complexity assumptions.* Boneh et al. [14] use supersingular curves to implement their BGN cryptosystem, where SXDDH is known to be easy, thus this would require implementing the BGN cryptosystem in a different curve (which may be hard). A few other structural changes also come into play since the BGN cryptosystem is only designed for very short messages.

7 Generic Security of the New Assumptions

To provide more confidence in our scheme and the assumptions we make, we prove lower bounds on their complexity for generic groups in the sense of Shoup [36, 42].

Let us begin by recalling the basics. We follow the notation and general outline of Boneh and Boyen [12]. In the generic group model, elements of the bilinear groups $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ are encoded as unique random strings. Thus, the adversary cannot directly test any property other than equality. Oracles are assumed to perform operations between group elements, such as performing the group operations in $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$. The opaque encoding of the elements of $\mathbb{G}_1$ is defined as the function $\xi_1 : \mathbb{Z}_p \to \{0,1\}^*$, which maps all $a \in \mathbb{Z}_p$ to the string representation $\xi_1(a)$ of $g^a \in \mathbb{G}_1$. Likewise, we have $\xi_2 : \mathbb{Z}_p \to \{0,1\}^*$ for $\mathbb{G}_2$ and $\xi_T : \mathbb{Z}_p \to \{0,1\}^*$ for $\mathbb{G}_T$. The adversary $\mathcal{A}$ communicates with the oracles using the ξ-representations of the group elements only.

We now address the q-EDH assumption. This is an extension of the q-SDH shown to be hard in the generic group model by Boneh and Boyen [12]. We will closely follow their original proof. In fact, we achieve the same asymptotic complexity bound for q-EDH as was shown for q-SDH.

Theorem 7.1 (q-EDH is Hard in Generic Groups) *Let $\mathcal{A}$ be an algorithm that solves the q-EDH problem in the generic group model, making a total of q_G queries to the oracles computing the group action in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, and the oracle computing the bilinear pairing e. If $x \in \mathbb{Z}_p^*$ and ξ_1, ξ_2, ξ_T are chosen at random, then the probability ε that $\mathcal{A}(p, \xi_1(1), \xi_1(x), \ldots, \xi_1(x^q), \xi_2(1), \xi_2(x), \ldots, \xi_2(x^q))$ outputs $(c, \xi_1(y), \xi_1(y \cdot v), \xi_2(\frac{1}{x+v}), \xi_2(\frac{1}{v+c}))$ with $c \in \mathbb{Z}_p^*$, is bounded by*

$$\varepsilon \leq \frac{(q_G + 2q + 2)^2 q}{p} = O\left(\frac{(q_G)^2 q + q^3}{p}\right).$$

Proof. Consider an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ in the following game.

$\mathcal{B}$ maintains three lists of pairs $L_1 = \{(F_{1,i}, \xi_{1,i}) : i = 0, \ldots, \tau_1 - 1\}$, $L_2 = \{(F_{2,i}, \xi_{2,i}) : i = 0, \ldots, \tau_2 - 1\}$, $L_T = \{(F_{T,i}, \xi_{T,i}) : i = 0, \ldots, \tau_T - 1\}$, such that, at step τ in the game, we have $\tau_1 + \tau_2 + \tau_T = \tau + 2q + 2$. The $F_{1,i}$ and $F_{2,i}$ are polynomials of degree $\leq q$ in $\mathbb{Z}_p[x]$, and $F_{T,i}$ are polynomials of degree $\leq 2q$ in $\mathbb{Z}_p[x]$. The $\xi_{1,i}$, $\xi_{2,i}$, and $\xi_{T,i}$ are set to unique random strings in $\{0,1\}^*$. Of course, we start the q-EDH game at step $\tau = 0$ with $\tau_1 = q + 1$, $\tau_2 = q + 1$, and $\tau_T = 0$. These correspond to the polynomials $F_{1,0} = F_{2,0} = 1$ and $F_{1,i} = F_{2,i} = x^i$ for $i = 1$ to q, and the random strings $\xi_{1,0}, \ldots, \xi_{1,q}$ and $\xi_{2,0}, \ldots, \xi_{2,q}$.

$\mathcal{B}$ begins the game with $\mathcal{A}$ by providing it with the $2q + 2$ strings $\xi_{1,0}, \ldots, \xi_{1,q}, \xi_{2,0}, \ldots, \xi_{2,q}$. Now, we describe the oracles $\mathcal{A}$ may query.

Group action: $\mathcal{A}$ inputs two group elements $\xi_{1,i}$ and $\xi_{1,j}$, where $0 \leq i, j \leq \tau_1$, and a request to multiply/divide. $\mathcal{B}$ sets $F_{1,\tau_1} \leftarrow F_{1,i} \pm F_{1,j}$. If $F_{1,\tau_1} = F_{1,u}$ for some $u \in \{0, \ldots, \tau_1 - 1\}$, then $\mathcal{B}$ sets $\xi_{1,\tau_1} = \xi_{1,u}$; otherwise, it sets ξ_{1,τ_1} to a random string in $\{0,1\}^* \setminus \{\xi_{1,0}, \ldots, \xi_{1,\tau_1 - 1}\}$. Finally, $\mathcal{B}$ returns ξ_{1,τ_1} to $\mathcal{A}$, adds $(F_{1,\tau_1}, \xi_{1,\tau_1})$ to L_1, and increments τ_1. Group actions for $\mathbb{G}_2$ and $\mathbb{G}_T$ are handled the same way.

Pairing: $\mathcal{A}$ inputs two group elements $\xi_{1,i}$ and $\xi_{2,j}$, where $0 \leq i \leq \tau_1$ and $0 \leq j \leq \tau_2$. $\mathcal{B}$ sets $F_{T,\tau_T} \leftarrow F_{1,i} \cdot F_{2,j}$. If $F_{T,\tau_T} = F_{T,u}$ for some $u \in \{0, \ldots, \tau_T - 1\}$, then $\mathcal{B}$ sets $\xi_{T,\tau_T} = \xi_{T,u}$; otherwise, it sets ξ_{T,τ_T} to a random string in $\{0,1\}^* \setminus \{\xi_{T,0}, \ldots, \xi_{T,\tau_T - 1}\}$. Finally, $\mathcal{B}$ returns ξ_{T,τ_T} to $\mathcal{A}$, adds $(F_{T,\tau_T}, \xi_{T,\tau_T})$ to L_T, and increments τ_T.

We assume SXDDH holds in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ and therefore no ismorphism oracles exist. However, our subsequent analysis would be the same even if such oracles were present.

Eventually $\mathcal{A}$ stops and outputs a tuple of elements $(c, \xi_{1,a}, \xi_{1,b}, \xi_{2,d}, \xi_{2,f})$, where $0 \leq a, b \leq \tau_1$ and $0 \leq d, f \leq \tau_2$. To later test the correctness of $\mathcal{A}$'s output within the framework of this game, $\mathcal{B}$ computes the polynomials:

$$\begin{aligned} F_{T,*} &= \left(\frac{F_{1,b}}{F_{1,a}} + x\right) \cdot F_{2,d} - 1. \\ F_{T,\circ} &= \left(\frac{F_{1,b}}{F_{1,a}} + c\right) \cdot F_{2,f} - 1. \end{aligned}$$

Intuitively, this correspond to the equalities "$e(h^x h^v, \tilde{g}^{1/(x+v)}) = e(h, \tilde{g}) = e(h^v h^c, \tilde{g}^{1/(v+c)})$", where h denotes the element of $\mathbb{G}_1$ represented by $\xi_{1,a}$, h^v denotes the element of $\mathbb{G}_1$ represented by $\xi_{1,b}$, $\tilde{g}^{1/(x+v)}$ denotes the element of $\mathbb{G}_2$ represented by $\xi_{2,d}$, and $\tilde{g}^{1/(v+c)}$ denotes the element of $\mathbb{G}_2$ represented by $\xi_{2,f}$.

Analysis of $\mathcal{A}$'s Output. For $\mathcal{A}$'s response to *always* be correct, then $F_{T,*}(x) = F_{T,\circ}(x) = 0$ for any value of x. We now argue that it is *impossible* for $\mathcal{A}$ to achieve this. By solving polynomials $F_{T,*} = 0$ and $F_{T,\circ} = 0$ in terms of $F_{1,b}/F_{1,a}$, we have the equation:

$$\frac{1}{F_{2,d}} + x = \frac{1}{F_{2,f}} + c \tag{1}$$

We multiply out the denominators and re-arrange equation (1) to obtain:

$$F_{2,d} \cdot F_{2,f} \cdot x - F_{2,d} \cdot F_{2,f} \cdot c + F_{2,d} - F_{2,f} = 0 \tag{2}$$

Now, the polynomials $F_{2,d}$ and $F_{2,f}$ must be some linear combination of polynomials corresponding to $\mathcal{A}$'s input, such as:

$$\begin{aligned} F_{2,d} &= d_0 + d_1 x + d_2 x^2 + \cdots + d_q x^q \\ F_{2,f} &= f_0 + f_1 x + f_2 x^2 + \cdots + f_q x^q \end{aligned}$$

First, we claim that for equation (2) to hold, it must be the case that $d_1 = d_2 = \cdots = d_q = 0$ and $f_1 = f_2 = \cdots = f_q = 0$, otherwise the first term will not be able to be canceled. This leaves us to consider d_0 and f_0. We immediately see that *one* of these must be zero to cancel the first term; next it is apparent that if $d_0 = 0$, then it must be the case that $f_0 = 0$ and vice versa. However, the only way for $\mathcal{A}$ to return a correct tuple with $F_{2,d} = F_{2,f} = 0$ is for $c = x$ (i.e., $1/(x+v) = 0$ and $1/(v+c) = 0$); since c is a constant, obviously this cannot hold for every value of x. Thus, we conclude that $\mathcal{A}$'s success depends *solely* on his luck when x is instantiated.

Analysis of $\mathcal{B}$'s Simulation. At this point $\mathcal{B}$ chooses a random $x^* \in \mathbb{Z}_p^*$. $\mathcal{B}$ now tests (in equations 3,4,5) if its simulation was perfect; that is, if the instantiation of x by x^* does *not* create any equality relation among the polynomials that was not revealed by the random strings provided to $\mathcal{A}$. $\mathcal{B}$ also tests (in equations 6, 7) whether or not $\mathcal{A}$'s output was correct. Thus, $\mathcal{A}$'s overall success is bounded by the probability that any of the following holds:

$$\begin{aligned} F_{1,i}(x^*) - F_{1,j}(x^*) &= 0, \quad \text{for some } i, j \text{ such that } F_{1,i} \neq F_{1,j}, & (3) \\ F_{2,i}(x^*) - F_{2,j}(x^*) &= 0, \quad \text{for some } i, j \text{ such that } F_{2,i} \neq F_{2,j}, & (4) \\ F_{T,i}(x^*) - F_{T,j}(x^*) &= 0, \quad \text{for some } i, j \text{ such that } F_{T,i} \neq F_{T,j}, & (5) \\ F_{T,*}(x^*) &= 0, & (6) \\ F_{T,\circ}(x^*) &= 0. & (7) \end{aligned}$$

We observe that $F_{T,*}$ and $F_{T,\circ}$ are non-trivial polynomials of degree at most $\leq 2q+1$. Each polynomial $F_{1,i}$ and $F_{2,i}$ has degree at most q. Thus, the polynomial $(F_{1,h} \cdot x)$ has degree at most $(q+1)$. The polynomial $(F_{1,h} \cdot x \cdot F_{2,j})$ has degree at most $(2q+1)$, which is the dominating term.

For fixed i and j, the first two cases occur with probability $\leq q/p$; and the third occurs with probability $\leq 2q/p$. (Indeed, $F_{1,i} - F_{1,j}$ and $F_{2,i} - F_{2,j}$ are polynomials of degree at most q, while $F_{T,i} - F_{T,j}$ may have degree up to $2q$.) Finally, the fourth and fifth cases happen with probability $\leq (2q+1)/p$. Now summing over all (i,j) pairs in each case, we bound $\mathcal{A}$'s overall success probability $\varepsilon \leq \binom{\tau_1}{2}\frac{q}{p} + \binom{\tau_2}{2}\frac{q}{p} + \binom{\tau_T}{2}\frac{2q}{p} + \frac{2(2q+1)}{p}$. Since $\tau_1 + \tau_2 + \tau_T \leq q_G + 2q + 2$, we end with $\varepsilon \leq (q_G + 2q + 2)^2(q/p) = O((q_G^2 + q^2)(q/p))$. □

The following corollary is immediate.

Corollary 7.2 *Any adversary that breaks the q-EDH assumption with constant probability $\varepsilon > 0$ in generic groups of order p such that $q < o(\sqrt[3]{p})$ requires $\Omega(\sqrt{\varepsilon p/q})$ generic group operations.*

We now turn our attention from a computational to a decisional problem. Recall from Section 2 that the Strong SXDDH assumption involves oracle $O_x(\cdot)$ that take as input a value $m \in \mathbb{Z}_p^*$ and returns $(g^v, \tilde{g}^{1/(x+v)}, \tilde{g}^{1/(v+m)})$ for $v \in \mathbb{Z}_p^*$ randomly chosen by the oracle, and an oracle $Q_y(\cdot)$ that takes the same type of input and returns $(a, a^y, a^v, \tilde{g}^{1/(y+v)}, \tilde{g}^{1/(v+m)})$, for $a \in \mathbb{G}_1$ and $v \in \mathbb{Z}_p^*$ chosen randomly by the oracle. These random values are freshly chosen at each invocation of the oracle.

Theorem 7.3 (Strong SXDDH is Hard in Generic Groups) *Let $x \in \mathbb{Z}_p^*$, $b \in \{0,1\}$, and ξ_1, ξ_2, ξ_T be chosen at random. Also, if $b = 1$, set $y = x$, but if $b = 0$, then set y to be a value selected randomly from $\mathbb{Z}_p^* \setminus x$. Let $\mathcal{A}$ be an algorithm that solves the Strong SXDDH problem in the generic group model, making a total of q_G queries to the oracles computing the group action in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, the oracle computing the bilinear pairing e, and the two oracles $O_x(\cdot)$ and $Q_y(\cdot)$ as described above. Then the probability ε that $\mathcal{A}(p, \xi_1(1), \xi_1(x), \xi_2(1)) = b$ is bounded by*

$$\varepsilon \leq \frac{1}{2} + \frac{(q_G+3)^2(3q_G)}{p} = \frac{1}{2} + O\left(\frac{q_G^3}{p}\right).$$

Proof. $\mathcal{B}$ maintains the lists L_1, L_2, and L_T as in the proof of Theorem 7.1. The only twist now is that we will let the $F_{1,i}, F_{2,i}$ and $F_{T,i}$'s be *rational functions* (i.e, fractions whose numerators and denominators are polynomials); and all polynomials are multivariate polynomials in $\mathbb{Z}_p[x, y, \ldots]$ where additional variables will be dynamically added. (Consider the bit b as not yet set.) At step τ in the game, we now have $\tau_1 + \tau_2 + \tau_T = \tau + 3$, where at $\tau = 0$, we set $\tau_1 = 2$, $\tau_2 = 1$, and $\tau_T = 0$. These correspond to the polynomials $F_{1,0} = F_{2,0} = 1$ and $F_{1,1} = x$. $\mathcal{B}$ also selects unique, random strings $\xi_{1,0}, \xi_{1,1}$, and $\xi_{2,0}$.

$\mathcal{B}$ begins the game with $\mathcal{A}$ by providing it with the strings $\xi_{1,0}, \xi_{1,1}$, and $\xi_{2,0}$. $\mathcal{A}$ may, at any time, make the group action or pairing queries as in the proof of Theorem 7.1. $\mathcal{A}$ may additionally query the following two oracles. Let $\tau_v = 1$ and $\tau_w = 1$ be counters.

Oracle $O_x(\cdot)$: $\mathcal{A}$ inputs m in $\mathbb{Z}_p^*$. $\mathcal{B}$ chooses a new *variable* v_{τ_v} and sets $F_{1,\tau_1} \leftarrow v_{\tau_v}$. If $F_{1,\tau_1} = F_{1,u}$ for some $u \in \{0, \ldots, \tau_1 - 1\}$, then $\mathcal{B}$ sets $\xi_{1,\tau_1} = \xi_{1,u}$; otherwise, it sets ξ_{1,τ_1} to a random string in $\{0,1\}^* \setminus \{\xi_{1,0}, \ldots, \xi_{1,\tau_1 - 1}\}$. $\mathcal{B}$ sends ξ_{1,τ_1} to $\mathcal{A}$ and adds $(F_{1,\tau_1}, \xi_{1,\tau_1})$ to L_1.

Next, $\mathcal{B}$ set $F_{2,\tau_2} \leftarrow 1/(x + v_{\tau_v})$ and $F_{2,\tau_2+1} \leftarrow 1/(v_{\tau_v} + m)$. For $j \in \{0,1\}$, if $F_{2,\tau_2+j} = F_{2,u}$ for some $u \in \{0, \ldots, \tau_2 - 1 + j\}$, then $\mathcal{B}$ sets $\xi_{2,\tau_2+j} = \xi_{2,u}$; otherwise, it sets ξ_{2,τ_2+j} to a random string in $\{0,1\}^* \setminus \{\xi_{2,0}, \ldots, \xi_{2,\tau_2-1+j}\}$. $\mathcal{B}$ sends $(\xi_{2,\tau_2}, \xi_{2,\tau_2+1})$ to $\mathcal{A}$, and adds $(F_{2,\tau_2}, \xi_{2,\tau_2})$ and $(F_{2,\tau_2+1}, \xi_{2,\tau_2+1})$ to L_2.

Finally, $\mathcal{B}$ adds one to τ_1, two to τ_2, and one to τ_v.

Oracle $Q_y(\cdot)$: $\mathcal{B}$ responds similarly to the former, except it chooses new *variables* r_{τ_w} and w_{τ_w}, and sets $F_{1,\tau_1} \leftarrow r_{\tau_w}$, $F_{1,\tau_1+1} \leftarrow r_{\tau_w} \cdot y$, $F_{1,\tau_1+2} \leftarrow r_{\tau_w} \cdot w_{\tau_w}$, $F_{2,\tau_2} \leftarrow 1/(y + w_{\tau_w})$, and $F_{2,\tau_2+1} \leftarrow 1/(w_{\tau_w} + m)$. At the end, $\mathcal{B}$ adds three to τ_1, two to τ_2, and one to τ_w.

Eventually $\mathcal{A}$ stops and outputs a guess $b' \in \{0,1\}$.

Analysis of $\mathcal{A}$'s Output. First, we argue that, provided $\mathcal{B}$'s simulation is perfect, the bit b' is independent of b; that is, $\mathcal{A}$ *cannot* output a string such that the corresponding polynomial is *always* equal when $x = y$ ($b = 1$) and non-zero otherwise ($b = 0$). We show this for each group $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$. Showing this for $\mathbb{G}_T$ is the hardest case. Here, we sum over all expressions containing i or j.

Group $\mathbb{G}_1$: The polynomials corresponding to elements in $\mathbb{G}_1$ that the adversary can compute as a linear combination of elements in its view are:

$$F_{1,a} = a_0 + a_1 \cdot x + a_{2,i} \cdot v_i + a_{3,j} \cdot r_j + a_{4,j} \cdot r_j \cdot y + a_{5,j} \cdot r_j \cdot w_j \tag{8}$$

where $i = 1$ to τ_v and $j = 1$ to τ_w. For $F_{1,a} = 0$, both a_1 and $a_{4,j}$ must be zero whether y is replaced by x or not; otherwise those terms cannot be canceled. The remaining polynomial does not contain the variables x or y.

Group $\mathbb{G}_2$: The polynomials corresponding to elements in $\mathbb{G}_2$ that the adversary can compute as a linear combination of elements in its view are:

$$F_{2,b} = b_0 + \frac{b_{1,i}}{x + v_i} + \frac{b_{2,i}}{v_i + m_i} + \frac{b_{3,j}}{y + w_j} + \frac{b_{4,j}}{w_j + m_j} \tag{9}$$

where $i = 1$ to τ_v, $j = 1$ to τ_w, and each $m_i, m_j \in \mathbb{Z}_p^*$ was chosen by the adversary. Suppose $F_{2,b} = 0$. We multiply out the denominators in equation (9) to obtain:

$$\begin{aligned} F'_{2,b} \;=\; & b_0(x+v_i)(v_i+m_i)(y+w_j)(w_j+m_j) + b_{1,i}(v_i+m_i)(y+w_j)(w_j+m_j) + \\ & b_{2,i}(x+v_i)(y+w_j)(w_j+m_j) + b_{3,j}(x+v_i)(v_i+m_i)(w_j+m_j) + \\ & b_{4,j}(x+v_i)(v_i+m_i)(y+w_j) \end{aligned} \tag{10}$$

Now, for $F'_{b,2} = 0$, regardless of whether we substitute x for y, we see that $b_0 = 0$, otherwise the term xv_iyw_j (or $x^2v_iw_j$) cannot be canceled. Similarly, $b_{1,i} = 0$ because of the unique summand xv_iy (or x^2v_i), which makes $b_{2,i} = 0$ because of the summand $v_i^2w_j$. Then, $b_{3,j} = 0$ because of the summand xyw_j (or x^2w_j), which makes $b_{4,j} = 0$ because of the summand $v_iw_j^2$. We are left with the constant zero.

Group $\mathbb{G}_T$: The polynomials corresponding to elements in $\mathbb{G}_T$ that the adversary can compute as a linear combination of elements in its view are:

$$F_{T,c} = \sum F_{1,a} \cdot F_{2,b}. \tag{11}$$

Now, a simple expansion of $F_{T,c}$ has thirty terms. Suppose we clear the denominators in $F_{T,c} = 0$ by multiplying out by $(x + v_i)(v_i + m_i)(y + w_j)(w_j + m_j)$, then we have

$$F'_{T,c} = \sum F_{1,a} \cdot F'_{2,b}. \tag{12}$$

Now, each of the terms in $F_{1,a}$ is unique and $F'_{2,b}$ contains the following unique summands $(xv_iyw_j, xv_iy, v_i^2w_j, xyw_j, v_iw_j^2)$. (Here, the summands $v_i^2w_j$ and $v_iw_j^2$ are actually not unique, but since they also do not contain x or y, it will not matter.) Multiplying these key components out and dropping the subscript for clarity, we obtain:

$$\begin{array}{rlllllllll}
F''_{T,c} = & c_0(vwxy) & + & c_1(vxy) & + & c_2(v^2w) & + & c_3(wxy) & + & c_4(vw^2) \\
+ & c_5(vwx^2y) & + & c_6(vx^2y) & + & c_7(v^2wx) & + & c_8(wx^2y) & + & c_9(vw^2x) \\
+ & c_{10}(v^2wxy) & + & c_{11}(v^2x^2y) & + & c_{12}(v^3w) & + & c_{13}(vwxy) & + & c_{14}(v^2w^2) \\
+ & c_{15}(vwxyz) & + & c_{16}(vxyz) & + & c_{17}(v^2wz) & + & c_{18}(wxyz) & + & c_{19}(vw^2z) \\
+ & c_{20}(vwxy^2z) & + & c_{21}(vxy^2z) & + & c_{22}(v^2wyz) & + & c_{23}(wxy^2z) & + & c_{24}(vw^2yz) \\
+ & c_{25}(vw^2xyz) & + & c_{26}(vwxyz) & + & c_{27}(v^2w^2z) & + & c_{28}(w^2xyz) & + & c_{29}(vw^3z)
\end{array} \tag{13}$$

Now, we are only interested in differences in the polynomial $F''_{T,c}$ when y is replaced by x or not. For clarity, we drop all terms containing neither x nor y, resulting in $c_2 = c_4 = c_{12} = c_{14} = c_{17} = c_{19} = c_{27} = c_{29} = 0$. We substitute $x = y$ to obtain.

$$\begin{array}{rlllllllll}
F'''_{T,c} = & c_0(vwx^2) & + & c_1(vx^2) & + & & + & c_3(wx^2) & + & \\
+ & c_5(vwx^3) & + & c_6(vx^3) & + & c_7(v^2wx) & + & c_8(wx^3) & + & c_9(vw^2x) \\
+ & c_{10}(v^2wx^2) & + & c_{11}(v^2x^3) & + & & + & c_{13}(vwx^2) & + & \\
+ & c_{15}(vwx^2z) & + & c_{16}(vx^2z) & + & & + & c_{18}(wx^2z) & + & \\
+ & c_{20}(vwx^3z) & + & c_{21}(vx^3z) & + & c_{22}(v^2wxz) & + & c_{23}(wx^3z) & + & c_{24}(vw^2xz) \\
+ & c_{25}(vw^2x^2z) & + & c_{26}(vwx^2z) & + & & + & c_{28}(w^2x^2z) & &
\end{array} \tag{14}$$

We want to know if there are any two terms that are symbolically equal when $x = y$ and not otherwise. Scanning the above, we see that the only non-unique terms are in positions 0 and 13, and in positions 15 and 26. Looking back to equation (13), we see that *both* positions 0 and 13 correspond to term $vwxy$, and that *both* positions 15 and 26 correspond to term $vwxyz$. Obviously, these terms will be the same regardless of the substitution of x for y. Since all other terms are unique, we conclude that the adversary's only chance of distinguishing comes from a lucky instantiation of these variables.

Analysis of $\mathcal{B}$'s Simulation. At this point $\mathcal{B}$ chooses random values $x^*, y^*, \{v_d^*\}_{d\in[1,\tau_v]}$, $\{w_d^*\}_{d\in[1,\tau_w]}, \{r_d^*\}_{d\in[1,\tau_w]} \in \mathbb{Z}_p^*$. $\mathcal{B}$'s simulation is perfect, and therefore reveals nothing to $\mathcal{A}$ about b, provided that none of the following non-trivial equality relations hold:

$$F_{1,i}(x^*, y^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) - F_{1,j}(x^*, y^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) = 0, \text{ for some } i, j \text{ s.t. } F_{1,i} \neq F_{1,j}, \tag{15}$$

$$F_{1,i}(x^*, x^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) - F_{1,j}(x^*, x^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) = 0 \text{ for some } i, j \text{ s.t. } F_{1,i} \neq F_{1,j}, \tag{16}$$

$$F_{2,i}(x^*, y^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) - F_{2,j}(x^*, y^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) = 0, \text{ for some } i, j \text{ s.t. } F_{2,i} \neq F_{2,j}, \tag{17}$$

$$F_{2,i}(x^*, x^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) - F_{2,j}(x^*, x^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) = 0, \text{ for some } i, j \text{ s.t. } F_{2,i} \neq F_{2,j}, \tag{18}$$

$$F_{T,i}(x^*, y^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) - F_{T,j}(x^*, y^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) = 0, \text{ for some } i, j \text{ s.t. } F_{T,i} \neq F_{T,j}, \tag{19}$$

$$F_{T,i}(x^*, x^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) - F_{T,j}(x^*, x^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) = 0, \text{ for some } i, j \text{ s.t. } F_{T,i} \neq F_{T,j}. \tag{20}$$

For fixed i and j, the probability of the first and second cases occurring are no more than $2/p$, where this results from the maximum degree of equation (8). For fixed i and j, the probability of the third and fourth cases occurring are no more than τ_2/p, where this results from the maximum degree of equation (10). Finally, for the fifth and sixth cases, the probability is at most $2\tau_2/p$, where this results from the maximum degree of equation (12).

Therefore, by summing over all (i, j) pairs in each case, we bound $\mathcal{A}$'s overall success probability $\varepsilon \leq 2\binom{\tau_1}{2}\frac{2}{p}+2\binom{\tau_2}{2}\frac{\tau_2}{p}+2\binom{\tau_T}{2}\frac{2\tau_2}{p}$. Since $\tau_1+\tau_2+\tau_T \leq q_G+3$, we end with $\varepsilon \leq (q_G+3)^2(2+q_G+2q_G)/p = O(q_G^3/p)$. □

The following corollary is immediate. Here $\gamma = \varepsilon - \frac{1}{2}$; that is, γ is the adversary's advantage beyond guessing.

Corollary 7.4 *Any adversary that breaks the Strong SXDDH assumption with constant probability* $\gamma > 0$ *in generic groups of order* p *requires* $\Omega(\sqrt[3]{\gamma p})$ *generic group operations.*

References

[1] Ben Adida, Susan Hohenberger, and Ronald L. Rivest. Ad-Hoc-Group Signatures from Hijacked Keypairs, 2005. Preliminary version in the DIMACS Workshop on Theft in E-Commerce. Available at `http://theory.lcs.mit.edu/~rivest/publications`.

[2] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars Knudsen, editor, *Advances in Cryptology — EUROCRYPT '02*, volume 2332 of LNCS, pages 83–107, 2002.

[3] Giuseppe Ateniese, Jan Camenisch, and Breno de Medeiros. Untraceable RFID tags via insubvertible encryption. In *ACM Conference on Computer and Communications Security (to appear)*, 2005.

[4] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *LNCS*, pages 255–270, 2000.

[5] Giuseppe Ateniese and Gene Tsudik. Some open issues and new directions in group signatures. In Matthew K. Franklin, editor, *Financial Cryptography, Third International Conference (FC'99)*, volume 1648 of LNCS, pages 196–211, 1999.

[6] Lucas Ballard, Matthew Green, Breno de Medeiros, and Fabian Monrose. Correlation-resistant storage. Technical Report TR-SP-BGMM-050705, Johns Hopkins University, Computer Science Department, 2005. `http://spar.isi.jhu.edu/~mgreen/correlation.pdf`.

[7] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. Technical Report 2005/133, International Association for Cryptologic Research, 2005. `http://eprint.iacr.org/2005/133`.

[8] D. Beaver. Secure multi-party protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4:75–122, 1991.

[9] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definition, simplified requirements and a construction based on general assumptions. In *Advances in Cryptology — EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629, 2003.

[10] Mihir Bellare and Adriana Palacio. GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, volume 2442 of LNCS, pages 162–177. Springer Verlag, 2002.

[11] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In *Topics in Cryptology: Proc. of the RSA Conference, Cryptographer's Track—CT-RSA 2005*, volume 3376 of *LNCS*, pages 136–ff. Springer-Verlag, 2005.

[12] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *Advances in Cryptology — EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73. Springer-Verlag, 2004.

[13] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Advances in Cryptology – CRYPTO '04*, volume 3152 of *LNCS*, pages 41–55, 2004.

[14] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In Joe Kilian, editor, *Theory of Cryptography (TCC)*, volume 3378 of LNCS, pages 325–341, 2005.

[15] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT '01*, volume 2248 of LNCS, pages 514–532, 2001.

[16] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *ACM Conference on Computer and Communications Security*, pages 132–145, 2004.

[17] Jan Camenisch and Ivan Damgård. Verifiable encryption, group encryption, and their applications to group signatures and signature sharing schemes. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT '00*, volume 1976 of LNCS, pages 331–345, 2000.

[18] Jan Camenisch and Anna Lysyanskaya. A Signature Scheme with Efficient Protocols. In *Security in Communication Networks*, volume 2576 of LNCS, pages 268–289, 2002.

[19] Jan Camenisch and Anna Lysyanskaya. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *Advances in Cryptology — CRYPTO 2004*, volume 3152 of LNCS, pages 56–72, 2004.

[20] Ran Canetti. *Studies in Secure Multiparty Computation and Applications.* PhD thesis, Weizmann Institute of Science, Rehovot 76100, Israel, June 1995.

[21] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[22] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. of Foundations of Computer Science (FOCS '01)*, pages 136–145, 2001.

[23] David Chaum and Eugene van Heyst. Group Signatures. In *Advances in Cryptology – EUROCRYPT '91*, volume of LNCS, pages 257–265, 1991.

[24] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT '00*, volume 1807 of LNCS, pages 418–430, 2000.

[25] Yevgeniy Dodis and Aleksandr Yampolskiy. A Verifiable Random Function with Short Proofs an Keys. In *Public Key Cryptography*, volume 3386 of LNCS, pages 416–431, 2005.

[26] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO '86*, volume 263 of LNCS, pages 186–194, 1986.

[27] Marc Fischlin. Communication-Efficient Non-Interactive Proofs of Knowledge with Online Extractors. In Matt Franklin, editor, *To appear in Advances in Cryptology – CRYPTO '05*, 2005.

[28] S. D. Galbraith and V. Rotger. Easy decision Diffie-Hellman groups. *Journal of Computation and Mathematics*, 7:201–218, 2004.

[29] Steven D. Galbraith. Supersingular curves in cryptography. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT '01*, volume 2248 of LNCS, pages 495–513, 2001.

[30] Steven D. Galbraith. Personal communication, August, 2005.

[31] Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, April 1988.

[32] Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. Traceable signatures. In *Advances in Cryptology — EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 571–589, 2004.

[33] Aggelos Kiayias and Moti Yung. Group signatures: Provable security, efficient constructions and anonymity from trapdoor-holders. Technical Report 2004/076, International Association for Cryptologic Research, 2004. `http://eprint.iacr.org/2004/076`.

[34] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In *Proc. of the 6th Annual Intern. Workshop on Selected Areas in Cryptography — SAC 1999*, volume 1758 of *LNCS*, pages 184–199, 1999.

[35] Noel McCullagh and Paulo S. L. M. Barreto. A new two-party identity-based authenticated key agreement. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA*, volume 3376 of LNCS, pages 262–274, 2004.

[36] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55:165–172, 1994.

[37] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of LNCS, pages 129–140, 1991.

[38] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. of Computer and Communications Security (ACM CCS 2000)*, pages 245–254. ACM Press, 2000.

[39] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 184–200. IEEE Computer Society Press, 2001.

[40] Michael Scott. MIRACL library. Indigo Software. http://indigo.ie/∼mscott/#download.

[41] Mike Scott. Authenticated ID-based key exchange and remote log-in with simple token and PIN number. Technical Report 2002/164, International Association for Cryptologic Research, 2002. `http://eprint.iacr.org/2002/164`.

[42] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology – EUROCRYPT '97*, LNCS, pages 256–266, 1997. Revised version: `http://www.shoup.net/papers/`.

[43] Eric R. Verheul. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. In *Advances in Cryptology — EUROCRYPT '01*, volume 2045 of *LNCS*, pages 195–210. Springer-Verlag, 2001.