Practical Group Signatures without Random Oracles

Giuseppe Ateniese* Jan Camenisch† Susan Hohenberger† Breno de Medeiros‡

Abstract

We provide a construction for a group signature scheme that is provably secure in a universally composable framework, within the standard model with trusted parameters. Our proposed scheme is fairly simple and its efficiency falls within small factors of the most efficient group signature schemes with provable security in any model (including random oracles). Security of our constructions require new cryptographic assumptions, namely the Strong LRSW, EDH, and Strong SXDH assumptions. Evidence for any assumption we introduce is provided by proving hardness in the generic group model.

Our second contribution is the first definition of security for group signatures based on the simulatability of real protocol executions in an ideal setting that captures the basic properties of unforgeability, anonymity, unlinkability, and exculpability for group signature schemes.

1 Introduction

Group signature schemes, introduced by Chaum and van Heyst [23], allow a member of a user group to sign anonymously on behalf of the group, where a user's anonymity may be revoked by a designated group manager, in case of disputes.

The motivation of this paper is twofold. Our first motivation is to build a practical group signature scheme provably secure under standard assumptions, in particular without resorting to the random oracle model. Prior to this work, there was no group signature scheme known achieving this (with the exception of the recent scheme by Boyen-Waters [16] which, however, fails to meet all required properties–we will later discuss their scheme in more detail).

In this paper, we present a full group signature scheme provably secure under new number-theoretic assumptions. Now, one might say that we trade the "assumption" that the Fiat-Shamir heuristic works with proof of knowledge protocols for discrete logarithms (e.g., such as the Schnorr signature scheme) with other possibly false assumptions. However, while one will probably never be able to prove that the Fiat-Shamir heuristic is reasonable for some cases (on the contrary, many cases are known for which it is unreasonable [33]), our new assumptions are algebraic and hence naturally much easier to analyze. Indeed, as a first step towards their justification, we prove that they hold in the generic group model [40, 46]. We hasten to point out that one can of course prove assumptions hard in the generic group model that are not hard against an algebraically unrestricted adversary, and that the generic group model has some of the same faults as random oracles [27]. Rather, a proof in the generic model should be considered a sanity check for an complexity assumption.

In the light of the quest for a group signature scheme provably secure in the standard model, one can view both schemes, the Boyen-Waters one and ours, as first steps in this direction. The Boyen-Waters scheme uses

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

†IBM Research; Zurich Research Laboratory; Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland.

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

somewhat less complicated assumptions, but sacrifies some desirable properties while our scheme captures these properties but makes more involved assumptions.

The second motivation of this paper is the definition of security in the reactive security or universally composablility model [8, 42, 22, 43]. Let us expand here. The security of early schemes was defined in terms of whether they satisfied a number of independent properties, and lacked a comprehensive view of adversarial behavior. Indeed, as initially the set of features considered was insufficient, some proposed schemes were subsequently broken. An important realization was the requirement that membership certificates be the equivalent of group manager signatures, in order to prevent against attacks by arbitrary group member coalitions [5]. Later, Bellare, Micciancio, and Warinschi [9] (BMW) introduced a security formalism based on adversarial games that combined the several requirements of previous works into fewer ones (namely, traceability and anonymity). However, the BMW formalization relies on the existence of a (trusted) key-issuing entity that generates all keys in the system and distributes them to the group manager and group members. As argued by Kiayias and Yung [36], BMW models a *weaker* primitive than the group signatures proposed by Chaum and van Heyst, since assuming a "tamper-proof" key setup conflicts with the goals of many practical schemes. The BMW model has been extensively adapted (via incorporating exculpability, 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, e.g., works by Boneh et al. [13] and Camenisch and Lysyanskaya [19].

The first formal model to apply to the case of dynamic groups was introduced by Kiayias et al. [36, 35]. The formalization works in the random oracle model, not the standard model, but it captures closely the security requirements of practical group signatures, and it can be readily applied to formally prove the security of practical schemes. Indeed, [36] includes a security proof of a variant of the Ateniese et al. scheme [4]. More recently, Bellare, Shi, and Zhang [11] have proposed a standard-model formalism for dynamic groups, and constructed theoretical schemes based on black-box zero-knowledge primitives.

Simultaneously with this evolution of understanding in group signatures was the development of the universally composable (UC)/reactive framework [8, 42, 22, 43]. The UC/reactive framework enables proofs of security of protocols that are valid under concurrent execution and in composition with other arbitrary protocols. It has been shown that this framework is more powerful than a property based definitional approach as it captures all properties at the same time. Indeed, examples are known of schemes that satisfy a property based defintion (i.e., each property individually) but not a UC/reactive framework definition that requires the fullfilment of all the properties at the same time [22].

To date, it remained an open problem to introduce a UC/reactive security model for group signatures. We introduce this definition in Section 2. It was carefully constructed to incorporate the original vision of Chaum and van Heyst and its subsequent developments. Our definition implies many of the guarantees of prior property-based definitions (e.g., [9, 11, 36, 35]). Two properties that we do not require are: (1) membership revocation, and (2) anonymity even after exposure of a user's secret key (forward anonymity), as in BMW [9]. Regarding point (2), we mean that we provide anonymity only to honest users, i.e., users of which the adversary does not know the secret key of which we believe is sufficient in practise. We emphasize that our model does not require any trusted key-issuing entity and group member secrets are known only to group members.

Note that our results are not in contradiction with the work of Datta et al. [26] on the impossibility of realizing an ideal functionality for group signatures with *perfect anonymity* (i.e., the probability of guessing the identity of one of two signers is *exactly* 1/2) and *perfect traceability* (i.e., the probability of a dishonest group manager violating exculpability is *exactly* zero). Datta et al. admit that they only consider a strong form of group signature, and indeed, their formulation of group signatures has a single entity generating all

signing keys. They also require that the scheme remain secure even when all public parameters are chosen by a potentially malicious group manager, i.e., they consider UC security in the *plain model*. In contrast, our definition (and scheme) does not have a key issuing entity and will allow some global parameters to be given to the group manager—e.g., bilinear map parameters. This is sometimes referred to as a UC model with *trusted parameters*.

We now summarize the contributions of our paper.

1.1 Our Contribution and Comparison with Previous Work

Strong security model: We provide the first definition of security for group signature schemes as an ideal functionality, and provide a construction of a practical and provably secure scheme within the new framework. Static-membership versions of our group signature schemes are secure in the context of composition and concurrent executions. The dynamic-membership version of our group signature scheme is similarly secure, if the join protocol is restricted to sequential execution.

Better Efficiency: Our construction is the first in the standard model to provide *constant* time and space efficiency with respect to the security parameter. Concurrently, and independently from us, Boyen and Waters [16] proposed a scheme provably secure the standard model (under a different definition of security). However, in their scheme the bit length of the signatures and the complexity of the signature and verification algorithms are $O((\log n) \cdot k)$, where n is the number of group members and k is the security parameter. In our scheme, all of these complexities will be $O((\log n) + k)$, where the $(\log n)$ term is there to guarantee that there are enough unique secret keys for each user in the algebraic group. In practice, $\log n$ is smaller than k, so our scheme is an $O(k)$ scheme (i.e., constant in the security parameter).

CCA-anonymity: Our scheme achieves CCA anonymity, i.e., anonymity holds even when the adversary continually has access to an open oracle (which, when queried on a signature, returns the signer's identity). In constrast, the Boyen-Waters scheme achieves only CPA anonymity, where the adversary is not allowed access to the oracle after receiving the challenge signature. Similarly as CCA security is the de facto requirement for encryption schemes, we believe this is a vital property for group signatures.

Strong exculpability: We achieve exculpability, an important property originally proposed by Chaum and van Heyst. In our scheme, a user's secret key is chosen by the user, and we prove that group managers cannot sign on behalf of honest users. The Boyen-Waters scheme lacks (strong) exculpability of users: In their scheme, a trusted key-issuing entity generates and distributes users' secret keys. Thus, this entity can sign messages on behalf of the user, and thus holding users *accountable* for any misbehavior is difficult.

Forward security: Our scheme does not achieve forward anonymity as defined in BMW [9] under the full anonymity property where members remain untraceable even if their secret keys are exposed, but only ordinary anonymity. The Boyen and Waters scheme does achieve forward anonymity.

General setting for bilinear mappings: We use curves with isomorphism-free paired groups, arguably the most efficient, secure, and versatile setting for pairings-based cryptography (e.g., see Galbraith et. al. [31]). In contrast, the Boyen-Waters scheme is restricted to symmetric bilinear mapping settings (supersingular curves). Even more significantly, their scheme uses elliptic curve groups of composite order, and they require that this order be hard to factor, implying it must be large (1024 bits or larger). Consequently, the representation of every elliptic curve point is similarly large, which heavily impacts the performance of cryptographic operations in these curves, as well as their bandwidth requirements.

1.2 Overview of the Construction

We now provide intuition for understanding our construction, described in detail in Section 5. Our group signature scheme has the standard protocols: Setup, Join, GroupSign, GroupVerify, and Open. Let $S = (Gen, Sign, Verify)$ be an efficient signature scheme secure in the standard model. Let the group manager have keypair (GPK, GSK) and a user have keypair (pk, sk), generated according to Gen. Consider the following scheme. During the Join protocol, the group manager gives the user a certificate on her public key: $Sign_{GSK}(pk)$. To sign message m as a group member, the user prepends her certificate and public key to her signature: ($Sign_{GSK}(pk)$, pk, $Sign_{sk}(m)$). This scheme is clearly unforgeable, but not anonymous. In order to achieve anonymity, the certificate and user's public key must be made unlinkably randomizable, while the signature element must be instantiated using a key-private signature scheme. We actually have the manager and user employ different signature schemes, S_1 and S_2.

S_1 is based on the pairing-based signature scheme of Camenisch and Lysyanskaya [19] (CL); Specifically it uses the extension of this scheme (CL$^+$) by Ateniese et al. [3]. Consider a bilinear map (pairing) $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, defined on groups of prime order p, with generators g, $\tilde{g}$, $\hat{g}$, respectively. 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 $m \in \mathbb{Z}_p^*$, choose 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) $e(AE, \tilde{g}^s) = e(C, \tilde{g})$.

The user's certificate is nothing more than a *blind* CL$^+$ signature from the group manager on the user's secret key sk, i.e., the group member signs sk without learning its value, a necessary condition for our scheme to provide exculpability. Fortunately, the CL$^+$ signature inherits an efficient blind-signature protocol from the original CL scheme. In the above, we instantiate the elements $(Sign_{GSK}(pk), pk)$ 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 in [3], these signatures can be unlinkably re-randomized by choosing a random $r \in \mathbb{Z}_p$ and computing $(a^r,\ a^{tr},\ a^{\{s+st(sk)\}r},\ a^{skr},\ a^{t(sk)r})$, assuming DDH is hard in $\mathbb{G}_1$. The user may therefore release a random-looking copy of her certificate with each group signature.

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 weak message attack secure, where the adversary must submit all challenge messages in advance of learning the public key. The scheme works in a similar pairing setting as the CL^+ signature. 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 $e(g^m pk, A) = e(g, \tilde{g})$.

As a thought experiment, consider our group signature structure using weak BB signatures to implement S_2. The construction is $(CL^+_{GSK}(sk);\ BB_{sk}(m)) = (a,\ a^t,\ a^{s+st(sk)},\ a^{sk},\ a^{t(sk)};\ \tilde{g}^{1/(sk+m)}) = (A,\ B,\ C,\ D,\ E,\ F)$, verifiable by checking the CL^+ signature first and then testing if $e(DA^m, F) = e(A, \tilde{g})$. Unfortunately, as the BB signatures are deterministic, it will be obvious when the same user signs the same message a second time. This violates our privacy definition, so we modify this basic scheme to provide more privacy and 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 triple $(A,\ B,\ C)$ by checking that $e(A\, pk, B) = e(g, \tilde{g})$ and $e(A\, g^m, C) = e(g, \tilde{g})$. We arrive at the construction $(CL^+_{GSK}(sk);\ BB^+_{sk}(m))$, or more exactly $(a,\ a^t,\ a^{s+st(sk)},\ a^{sk},\ a^{t(sk)};\ a^v,\ \tilde{g}^{1/(sk+v)},\ \tilde{g}^{1/(v+m)})$ for 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.

At this point, we have described the entire construction, except for how the Open algorithm works. The

simplest 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. Obviously, this algorithm runs linearly in the number of group members. We improve on this result in Section 6, providing two alternative Open algorithms. The first has $O(\sqrt{n} \cdot k)$ complexity, for membership groups of size n and security parameter k, under the same cryptographic assumptions as the basic scheme. The second reduces to $((\log n) + k)$ under an additional assumption.

Remark 1.1 (Security under concurrent executions.) The join protocol is the only protocol in our scheme that requires sequential composition for security, because it involves a zero-knowledge proof of knowledge. In Appendix B, we discuss some techniques for securely achieving limited forms of concurrency.

Remark 1.2 (Revocation.) Finding an elegant revocation mechanism is a pervasive problem among group signature schemes. To revoke a user in our scheme, the group manager could publish that user's *tracing information* obtained during the join protocol. We explore more efficient alternatives in the full version.

Length of Signatures. The signatures produced by this scheme are short. In the following comparisons we are going use the NIST suggested equivalence of 80-bit symmetric security with RSA-1024 and 128-bit symmetric security with RSA-3072. [1] For our schemes, we estimate the key sizes from our generic-model security reductions to be 1/3 of the equivalent symmetric key size, i.e., 240 bits for 80-bit symmetric security and 384 bits for 128-bit symmetric security. This 3:1 ratio should be also used to compare schemes based on the q-Strong Diffie Hellman assumption, due to recent results by Cheon [24].

Assuming that the bitlength of elements in $\mathbb{G}_1$ is 241 (an extra bit is needed to indicate the y-coordinate among two options), and that the curves implemented in the MIRACL library are used [44], 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 $\mathbb{G}_1$ (more precisely 721 bits), and our scheme would take approximately 2888 bits to represent a group signature, comprised of six elements in $\mathbb{G}_1$ and two elements in $\mathbb{G}_2$. If the newer curves of embedding degree 12 are used [7], one could employ 385-bit groups (for 128-bit generic security) to achieve RSA-3072 security equivalence. These new curves have better ratios, with $\log |\mathbb{G}_2| / \log |\mathbb{G}_1| = 2$. In this case, our signatures would take 3848 bits to be represented, about 25% larger than a plain RSA signature with the same security level. As mentioned before, this efficiency is incomparable with that of Boyen and Waters [16], which (1) grow logarithmically with the number of system members and (2) require elliptic curve group orders over a 1000 bits long.

Our scheme can be compared with the most efficient short group signatures secure in the random oracle setting. For instance, the scheme by Boneh, Boyen, and Shacham [13], which achieves only CPA-anonymity, would require about 2163 bits for the RSA-1024 security level (or about 1442 in the MNT setting). [2] A shorter scheme by Boneh and Shacham [15] requires 1682 bits to achieve RSA-1024 comparable security. [3]

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.

[1] `http://www.nsa.gov/ia/industry/crypto_elliptic_curve.cfm`

[2] Their paper provides the value 1533 bits instead of 2163, but this does not take into account the above mentioned results about the concrete security of the q-Strong Diffie-Hellman assumption [24].

[3] Again, this value is larger than the one provided by the authors to account for the concrete security of q-Strong Diffie-Hellman.

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.

Let $params$ be global parameters generated during a setup phase; ideally $params$ is empty.

- The GroupSetup$(1^k, params)$ algorithm is a key generation algorithm for the group manager $\mathcal{GM}$. It takes as input the security parameter 1^k and outputs the key pair $(pk_{\mathcal{GM}}, sk_{\mathcal{GM}})$. (Assume that $sk_{\mathcal{GM}}$ contains the $params$, so we do not have to give $params$ explicitly to the group manager again.)
- The 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 Join$(\mathcal{U}(pk_{\mathcal{GM}}, sk_{\mathcal{U}}), \mathcal{GM}(pk_{\mathcal{U}}, sk_{\mathcal{GM}}))$ protocol, the user $\mathcal{U}$ joins the signatory group managed by $\mathcal{GM}$. The user's output is a personalized group signing credential $C_{\mathcal{U}}$, or an error message. $\mathcal{GM}$'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 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 GroupVerify$(pk_{\mathcal{GM}}, m, \sigma)$ 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 $pk_{\mathcal{GM}}$.
- The Open$(sk_{\mathcal{GM}}, D, m, \sigma)$ algorithm allows the group manager, with $sk_{\mathcal{GM}}$ and database D, 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 VerifyOpen$(\mathcal{GM}(sk_{\mathcal{GM}}, D, m, \sigma, pk), \mathcal{V}(pk_{\mathcal{GM}}, m, \sigma, pk))$ protocol, $\mathcal{GM}$ 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, $\mathcal{F}_{gs}$

Our security model uses the ideal/real world model as in multiparty computation [20, 21, 22] and reactive systems [42, 43] to capture the security properties of group signatures in a single definition.

In the real world, there are a number of parties who together execute some cryptographic protocol. A 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 to and receive their outputs from a trusted party $\mathcal{T}$. The specification for how $\mathcal{T}$ behaves is formalized as an ideal functionality.

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 $\mathcal{S}$.

Group Signature Ideal Functionality. We now describe $\mathcal{F}_{gs}$. In addition to the environment $\mathcal{Z}$, we have two types of players: a group manager $\mathcal{GM}$ and users $\mathcal{U}_i$. 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.
- GroupSetup: Upon receiving $(\mathcal{GM}, \text{“group setup”})$ from $\mathcal{GM}$, send to $\mathcal{S}$ tuple $(\mathcal{GM}, \text{“group setup”})$.
- UserKeyGen: Similarly, upon receiving $(\mathcal{U}_i, \text{“keygen”})$ from $\mathcal{U}_i$ inform $\mathcal{S}$.
- Join: Upon receiving $(\mathcal{U}_i, \text{“enroll”})$ from $\mathcal{U}_i$, ask the group manager $\mathcal{GM}$ if $\mathcal{U}_i$ may join the group. The $\mathcal{GM}$ 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-$\mathcal{GM}$.
- GroupSign: Upon receiving $(\mathcal{U}_i, \text{“sign”}, m)$, 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 $\mathcal{GM}$ 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 $(\mathcal{U}_i, \text{“verify”}, m, id)$ from $\mathcal{U}_i$ (or $\mathcal{GM}$), 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 $\mathcal{GM}$ if it will allow $\mathcal{F}_{gs}$ to open id for user $\mathcal{U}_i$. If $\mathcal{GM}$ agrees and $\mathcal{U}_j \neq$ corrupt-$\mathcal{GM}$, then output the identity $\mathcal{U}_j$. Otherwise, output $\perp$.

Let us provide some intuition for understanding this model. Informally, the properties that we capture are unforgeability, anonymity, and exculpability. This definition is general enough to capture unforgeability under adaptive chosen message attack [34] without *requiring* schemes to be *strongly* unforgeable [2]. In a strongly unforgeable scheme, a new signature on a previously signed message is considered a forgery; while in the standard notion, a forgery must be on a new message.

The definition also captures the important exculpability property (i.e., even a rogue group manager cannot frame an honest user). Indeed, 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 $(\text{“sign”}, m)$ 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 $\mathcal{GM}$ will later be able to open it, reveal the identity of the original signer, and prove this to the user.

The above definition does not define membership revocation. However, it is not difficult to extend $\mathcal{F}_{gs}$ to address revocation, and we plan to do so in the full version of the paper.

It is not hard to see that our definition implies prior many of the guarantees of the property-based definitions (e.g., [9, 11, 36, 35]). Two properties that we do not require are: (1) membership revocation, and (2) anonymity even after exposure of a user secret key (forward anonymity), as in BMW [9]. While of course both of these properties could be added easily to the model, our scheme does not statisfy them. Note, however, that our definition (and scheme) does provide anonymity to (honest) users, i.e., users of which the adversary is not privy of their secret keys. Finally, notice that our definition implies CCA anonymity, i.e., an anonymity property based definition where the adversary is allowed to query the open oracle even after having been presented the challenge signature. This is in contrast to BMW model (and to the Boyen-Waters group signature scheme) which provide only CPA anonymity, i.e., where the adversary is no longer allowed access to the open oracle after seeing the challenge signature.

3 Preliminaries and Complexity Assumptions

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

Pairings: Let $\mathsf{BilinearSetup}$ be an algorithm that, on input the security parameter 1^k, outputs the parameters for a pairing 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 [14]. Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ all be (multiplicative) groups of prime order $p = \Theta(2^k)$, where each element of each group has a unique binary representation. Furthermore, let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be an efficient *pairing*, i.e., a mapping with the following properties: (Bilinearity) for all $g \in \mathbb{G}_1$, $\tilde{g} \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$, $e(g^a, \tilde{g}^b) = e(g, \tilde{g})^{ab}$; and (Non-degeneracy) 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 $\mathbb{G}_T$.

3.1 Complexity Assumptions

The security of our construction in Section §5 is based on the following assumptions about pairing groups. In Appendix §D, we provide generic group proofs for EDH and Strong SXDH. A generic group proof for Strong LRSW was previously given in [3].

Two criticisms could be made of these assumptions. The first could be that they are closely related to specific security properties of the scheme. With regards to this, we point out that even if we *were* to assume a specific property, such as unforgeability (which we don't do), security in our model would not follow. Indeed, it is non-trivial to show that our assumptions imply that our scheme realizes our ideal functionality. We also point out that the generic group proofs of these assumptions are highly non-trivial and required new techniques, which may be useful elsewhere.

A second criticism could be that the assumptions are interactive and thus not black-box falsifiable [39]. However, we believe that our provided generic-model hardness proofs show that these assumptions are reasonable: Violating them would result in the design of elliptic curve algorithms with better than generic efficiency, a major cryptographic breakthrough with likely wider ramifications. In addition, our proofs provide estimates for the key sizes required for particular security levels, making our security assumptions indeed very concrete: The resulting $\Omega(p^{1/3})$-generic security of our interactive assumptions (for elliptic curve subgroups of order p) puts them on a similar footing with related falsifiable assumptions, such as the q-Strong Diffie-Hellman assumption [24].

Let $\mathsf{BilinearSetup}(1^k) \to (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g})$, where $\mathbb{G}_1 = \langle g \rangle$ and $\mathbb{G}_2 = \langle \tilde{g} \rangle$, be public.

Assumption 1 (Symmetric External Diffie-Hellman (SXDH) [6, 3, 30]) *The 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 $\psi : \mathbb{G}_1 \to \mathbb{G}_2$ or $\psi' : \mathbb{G}_2 \to \mathbb{G}_1$.*

Note that SXDH also subsumes a traditional Co-Gap assumption, i.e., the Co-CDH problem [14] is hard in the pairing groups: Given $(g, \tilde{g}, g^x, \tilde{g}^y)$, it is hard to compute $\tilde{g}^{xy}$ or g^{xy}.

Good candidates for pairing groups where SXDH is hard are certain MNT curve implementations where no efficient isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ are known [6, 3, 47, 32, 30]. The asymmetric version of this assumption, simply called XDH, only requires that DDH be hard in $\mathbb{G}_1$ [29, 45, 38, 13, 6].

The LRSW assumption is a discrete-logarithm assumption introduced in 1999 by Lysyanskaya et al. [37] and used in many subsequent works. Recently, a stronger form of the LRSW assumption which *implies the SXDH assumption*, called *Strong LRSW*, was introduced by Ateniese et al. [3].

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 $m \in \mathbb{Z}_p^*$,*

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

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 $c \in \mathbb{Z}_p^*$:

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

We make an interactive version of this assumption. As we mentioned in the introduction, our efficiency analysis takes into account Cheon's recent results [24] on q-SDH.

Assumption 3 (Extended Diffie-Hellman (EDH)) *Let $x \in \mathbb{Z}_p^*$. Let oracle $O_x(\cdot)$ take input $c_i \in \mathbb{Z}_p^*$ and produce output $(g^{v_i}, \tilde{g}^{1/(x+v_i)}, \tilde{g}^{1/(v_i+c_i)})$, for a random $v_i \in \mathbb{Z}_p^*$. For all probabilistic polynomial-time adversaries $\mathcal{A}$, all $v, c \in \mathbb{Z}_p^*$, and all $a \in \mathbb{G}_1$ such that $a \neq 1$,*

$$\Pr[x \xleftarrow{R} \mathbb{Z}_p : \mathcal{A}^{O_x}(g, g^x, \tilde{g}, \tilde{g}^x) = (c, a, a^x, a^v, \tilde{g}^{1/(x+v)}, \tilde{g}^{1/(v+c)}) \wedge c \notin Q] < 1/\mathrm{poly}(k)$$

where Q is the set of queries $\mathcal{A}$ makes to oracle $O_x(\cdot)$.

The assumptions discussed so far are underlying the unforgeability of our group signature scheme. Its anonymity is based on a single assumption: that SXDH holds even when the adversary is given oracle access to additional information about the DDH instance.

Assumption 4 (Strong SXDH) *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 $x, y \in \mathbb{Z}_p$,*

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

In Theorem D.3, we show that Strong SXDH also has the same complexity as q-SDH for generic groups.

4 Key Building Blocks: CL$^+$ and BB$^+$ Signatures

As mentioned in Section §1, our group signature scheme is built out of two standard signature schemes secure without random oracles. We review the important details now.

4.1 Camenisch-Lysyanskaya Signatures

Recall the basic Pedersen commitment scheme [41], in which the public parameters are a group G of prime order p, and two generators g and h of G. To commit to the value $m \in \mathbb{Z}_p$, pick a random $r \in \mathbb{Z}_p$ and set $C = \mathrm{PedCom}(m; r) = g^m h^r$.

The Camenisch-Lysyanskaya (CL) signature scheme is secure without random oracles under the LRSW assumption [19]. CL signatures are also useful, because they support an efficient two-party protocol for obtaining a CL signature on the value (message) committed to in a Pedersen commitment. The common inputs are $C = \mathrm{PedCom}(m; r)$ and the verification key of the signer pk. The signer additionally knows the corresponding signing key sk, while the receiver additionally knows m and r. As a result of this protocol, the receiver obtains the signature $\sigma_{sk}(m)$, while the signer does not learn anything about m. For our current purposes, it will not matter *how* this protocol actually works. Fortunately, a recent extension of the CL signatures by Ateniese et al. [3], denoted CL$^+$, inherits this protocol.

CL$^+$ Signatures. Let the security parameter be 1^k. The global parameters are the description of a pairing $params = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g})$, where $\mathbb{G}_1 = \langle g \rangle$ and $\mathbb{G}_2 = \langle \tilde{g} \rangle$, obtained by running $\mathsf{BilinearSetup}(1^k)$. Keypairs are of the form $pk = (\tilde{g}^s, \tilde{g}^t)$ and $sk = (s, t) \in \mathbb{Z}_p^2$.

- **Signing:** Choose random $a \in \mathbb{G}_1$, output $(a, a^t, a^{s+stm}, a^m, a^{mt})$ as the signature on *hidden* message $m \in \mathbb{Z}_p^*$.
- **Verification:** On input a purported signature (A, B, C, D, E) accept that σ authenticates the message hidden as $\log_A(D)$ if and only if: (1) $e(B, \tilde{g}) = e(A, \tilde{g}^t)$, (2) $e(D, \tilde{g}^t) = e(E, \tilde{g})$, and (3) $e(C, \tilde{g}) = e(A, \tilde{g}^s)e(E, \tilde{g}^s)$.
- **Re-Randomization:** On input a signature (A, B, C, D, E), choose a random $r \in \mathbb{Z}_p^*$ and output $(A^r, B^r, C^r, D^r, E^r)$.

CL$^+$ signatures are secure assuming SXDH and Strong LRSW. As previously observed in [3], when CL$^+$ signatures are set in pairing groups where SXDH is hard, this re-randomization is *unlinkable*. We formally argue this second point in Lemma A.2.

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 pairing $params = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ obtained by running $\mathsf{BilinearSetup}(1^k)$ (here, we ignore the generators output by $\mathsf{BilinearSetup}$). Keypairs are of the form $pk = (g, g^{sk}, \tilde{g})$ and $sk \in \mathbb{Z}_p^*$, for random generators $g \in \mathbb{G}_1$ and $\tilde{g} \in \mathbb{G}_2$. To sign a message $m \in \mathbb{Z}_p^*$, output the signature $\tilde{g}^{1/(sk+m)}$. To verify signature σ, accept if and only if $e(\sigma, g^{sk} g^m) = e(g, \tilde{g})$. Note that in this work we reverse the roles of $\mathbb{G}_1$ and $\mathbb{G}_2$ from the original description in [12]. As in other implementation of pairing-based schemes where distortion maps are not available, one chooses the role of each pairing group to maximize the efficiency of one's protocol.

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. Boneh and Boyen gave one method of modifying this weak scheme into an adaptively-secure one [12]. We provide a second method, which is more suited to our purposes.

BB$^+$ Signatures. The intuition here is that to issue a signature on a message m, a weak BB signature under sk is issued on a one-time signing key v, and then another weak BB signature under v is issued on message m. The additional randomness v allows to prove adaptive security.

- **Key generation:** Same as before. (Although, now the same bases may be used for all keys.)
- **Signing:** On input a secret key sk and a message $m \in \mathbb{Z}_p^*$, select a random $r \in \mathbb{Z}_p^*$, and output the signature $(g^r, \tilde{g}^{1/(sk+r)}, \tilde{g}^{1/(r+m)})$.
- **Verification:** On input a public key $(g, g^{sk}, \tilde{g})$, a message m, and a purported signature (A, B, C), accept if and only if: (1) $e(g^{sk}A, B) = e(g, \tilde{g})$ and (2) $e(Ag^m, C) = e(g, \tilde{g})$.

Lemma 4.1 *The BB$^+$ signature scheme is existentially unforgeable under adaptive chosen-message attack under the EDH assumption.*

5 Our Basic Group Signature Construction

Notation: BB$^+$ and CL$^+$, respectively, denote our Section §4 modifications of the Boneh-Boyen [12] and Camenisch-Lysyanskaya [19] signature schemes. When we write $A = Sign_{GSK}^{CL+}(m; a)$, we mean that A is a CL$^+$ signature under key GSK on message m using base a; that is, $A = (a, a^t, a^{s+stm}, a^m, a^{mt})$ for $GSK = (s, t)$. Similarly, when we write $A = Sign_{sk}^{BB+}(m; g, \tilde{g})$, we mean that A is a BB$^+$ signature under key sk on message m using bases $(g, \tilde{g})$; that is, $A = (g^v, \tilde{g}^{1/(sk+v)}, \tilde{g}^{1/(v+m)})$ for some $v \in \mathbb{Z}_p^*$.

Let $\mathsf{BilinearSetup}(1^k) \rightarrow params = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g})$, where $\mathbb{G}_1 = \langle g \rangle$ and $\mathbb{G}_2 = \langle \tilde{g} \rangle$.

GroupSetup$(1^k, params)$**:** The group manager establishes the public parameters for the Pedersen commitment scheme [41] and adds those to $params$. Then, the group manager executes $Gen^{CL+}(1^k, params)$ to obtain $GPK = (params, \mathcal{S} = \tilde{g}^s, \mathcal{T} = \tilde{g}^t)$ and $GSK = (s, t)$.

UserKeyGen$(1^k, params)$**:** Each user $\mathcal{U}$ selects random $sk \in \mathbb{Z}_p^*$ and random $h \in \mathbb{G}_1$, and outputs a 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 $pk_i = (p_1, p_2)$ and tracing information $Q_i = \tilde{g}^{sk_i}$ to $\mathcal{GM}$. If $e(p_1, Q_i) \neq p_2$ or sk_i was already in D, $\mathcal{GM}$ aborts. Else, $\mathcal{GM}$ enters Q_i in database D.
2. The user sends a commitment $A = \mathrm{PedCom}(sk_i)$ to $\mathcal{GM}$. The user and $\mathcal{GM}$ run the CL protocol (see Section 4.1) 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 $Sign_{GSK}^{CL+}(sk_i; f_1) = (f_2, f_3)$ 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.
3. The user provides a zero-knowledge proof that the committed value sk_i contained in commitment A is consistent with the public key pk_i and a zero-knowledge proof of knowledge of sk_i using any proof technique that is *extractable*. (For more on such proofs, see Appendix B.)
4. The group manager provides an extractable zero-knowledge proof of $GSK = (s, t)$.
5. Next, the user locally computes the values $f_4 = f_1^{sk_i}$ and $f_5 = f_2^{sk_i}$.

6. At the end of this protocol, the user obtains the following membership certificate:

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

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.

1. Re-randomize C_i using a random $r \in \mathbb{Z}_p$, i.e., compute $(a_1, \ldots, a_5) = (f_1^r, \ldots, f_5^r)$.
2. Compute $Sign_{sk_i}^{BB^+}(m; a_5, \tilde{g}) = (a_6, a_7, a_8)$.
3. Output the signature $(a_1, \ldots, a_8)$ of the form $(b, b^t, b^{s+st(sk_i)}, b^{sk_i}, b^{(sk_i)t}, b^v, \tilde{g}^{1/(sk_i+v)}, \tilde{g}^{1/(v+m)})$.

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

1. Check that $(a_1, a_2, a_3, a_4, a_5)$ is a valid CL$^+$ signature for public key GPK where the *hidden* message is the exponent of a_4 (base a_1). Specifically, verify that: (1) $e(a_1, \mathcal{T}) = e(a_2, \tilde{g})$, (2) $e(a_4, \mathcal{T}) = e(a_5, \tilde{g})$, and (3) $e(a_1 a_5, \mathcal{S}) = e(a_3, \tilde{g})$.
2. Check that (a_6, a_7, a_8) is a valid BB$^+$ signature for public key $(a_1, a_4, \tilde{g})$ on message m. Specifically, verify that: (1) $e(a_4 a_6, a_7) = e(a_1, \tilde{g})$ and (2) $e(a_6 a_1^m, a_8) = e(a_1, \tilde{g})$.
3. If both 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.

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, 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. We call this step *anonymity revocation*. Here, the $\mathcal{GM}$ provides a zero-knowledge proof of knowledge of a value $\alpha \in \mathbb{G}_2$ (i.e., the tracing information for $\mathcal{U}_i$) such that $e(p_1, \alpha) = p_2$ and $e(a_1, \alpha) = e(a_4, \tilde{g})$ [1].

The revocation described above revokes the anonymity of a particular signature, however, the $\mathcal{GM}$ could instead revoke the anonymity of *all* signatures belonging to user $\mathcal{U}_i$ by publishing the tracing information Q_i. Then anyone can verify that the user with public key $pk = (p_1, p_2)$ must be responsible by checking that: (1) $e(p_1, Q_i) = p_2$, and (2) $e(a_1, Q_i) = e(a_4, \tilde{g})$.

Theorem 5.1 *In the plain model, the above group signature scheme realizes $\mathcal{F}_{gs}$ from Section §2 under the Strong LRSW, the EDH, and the Strong SXDH assumptions.*

Proof of Theorem 5.1 appears in Appendix A.

6 Opening Signatures in Sublinear Time

The basic Open algorithm described in Section §5 takes $O(n \cdot k)$ for a signing group of n members and security parameter k. Practically, this precludes this scheme from being used for many applications with *large* groups. We provide several options to remedy this situation in Appendix C.

First, we present an Open algorithm with complexity $O(\sqrt{n} \cdot k)$ which can be extended to one with complexity $O((\log n) \cdot k)$ at the cost of group signatures becoming of size $O((\log n) \cdot k)$. This improvement requires no additional assumptions, but does add two elements in $\mathbb{G}_1$ (resp. $\log n$ elements) to the signature length. Next, we present a $O((\log n) + k)$ Open algorithm. This increases the basic signature by three elements in $\mathbb{G}_1$ and requires a slightly different anonymity assumption.

References

[1] B. Adida, S. Hohenberger, and R. L. Rivest. Ad-Hoc-Group Signatures from Hijacked Keypairs, 2005. At `http://theory.lcs.mit.edu/~rivest/publications`.

[2] J. H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In *EUROCRYPT*, volume 2332 of LNCS, pages 83–107, 2002.

[3] G. Ateniese, J. Camenisch, and B. de Medeiros. Untraceable RFID tags via insubvertible encryption. In *ACM CCS*, pages 92–101, 2005.

[4] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO*, volume 1880 of LNCS, pages 255–270, 2000.

[5] G. Ateniese and G. Tsudik. Some open issues and new directions in group signatures. In *Financial Cryptography*, volume 1648 of LNCS, pages 196–211, 1999.

[6] L. Ballard, M. Green, B. de Medeiros, and F. Monrose. Correlation-resistant storage. Technical Report TR-SP-BGMM-050705, Johns Hopkins University, CS Dept, 2005. `http://spar.isi.jhu.edu/~mgreen/correlation.pdf`.

[7] P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order, 2005. Cryptology ePrint Archive: 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] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definition, simplified requirements and a construction based on general assumptions. In *EUROCRYPT*, volume 2656 of LNCS, pages 614–629, 2003.

[10] M. Bellare and A. Palacio. GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. In *CRYPTO*, volume 2442 of LNCS, pages 162–177, 2002.

[11] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA*, volume 3376 of LNCS, pages 136–153, 2005.

[12] D. Boneh and X. Boyen. Short signatures without random oracles. In *EUROCRYPT 2004*, volume 3027 of LNCS, pages 56–73, 2004.

[13] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *CRYPTO*, volume 3152 of *LNCS*, pages 41–55, 2004.

[14] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *ASIACRYPT*, volume 2248 of LNCS, pages 514–532, 2001.

[15] D. Boneh and H. Shacham. Group signatures with verifier-local revocation. In *Proc. of the ACM Conf. on Computer and Communications Security (ACM CSS 2004)*, pages 168–177. ACM Press, 2004.

[16] X. Boyen and B. Waters. Compact Group Signatures Without Random Oracles. In *EUROCRYPT '06*, volume 4004 of LNCS, pages 427–444, 2006.

[17] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *ACM CCS*, pages 132–145, 2004.

[18] J. Camenisch and I. Damgård. Verifiable encryption, group encryption, and their applications to group signatures and signature sharing schemes. In *ASIACRYPT*, volume 1976 of LNCS, pages 331–345, 2000.

[19] J. Camenisch and A. Lysyanskaya. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *CRYPTO*, volume 3152 of LNCS, pages 56–72, 2004.

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

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

[22] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[23] D. Chaum and E. van Heyst. Group Signatures. In *EUROCRYPT*, volume 547 of LNCS, pages 257–265, 1991.

[24] J. H. Cheon. Security Analysis of the Strong Diffie-Hellman Problem. In *EUROCRYPT '06*, volume 4004 of LNCS, pages 1–11, 2006.

[25] I. Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT*, volume 1807 of LNCS, pages 418–430, 2000.

[26] A. Datta, A. Derek, J. C. Mitchell, A. Ramanathan, and A. Scedrov. Games and the impossibility of realizable ideal functionality. In *TCC*, volume 3876 of LNCS, pages 360–379, 2006.
[27] A. W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In *ASIACRYPT '02*, volume 2501 of LNCS, pages 100–109, 2002.
[28] M. Fischlin. Communication-Efficient Non-Interactive Proofs of Knowledge with Online Extractors. In *CRYPTO*, pages 152–168, 2005.
[29] S. D. Galbraith. Supersingular curves in cryptography. In *ASIACRYPT*, volume 2248 of LNCS, pages 495–513, 2001.
[30] S. D. Galbraith. Personal communication, August, 2005.
[31] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. Technical Report 2006/165, International Association for Cryptological Research, 2006.
[32] S. D. Galbraith and V. Rotger. Easy decision Diffie-Hellman groups. *Journal of Computation and Mathematics*, 7:201–218, 2004.
[33] S. Goldwasser and Y. T. Kalai. On the (In)security of the Fiat-Shamir Paradigm. In *FOCS*, pages 102–115, 2003.
[34] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. of Computing*, 17(2):281–308, 1988.
[35] A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In *EUROCRYPT*, volume 3027 of *LNCS*, pages 571–589, 2004.
[36] A. Kiayias and M. Yung. Group signatures: Provable security, efficient constructions and anonymity from trapdoor-holders, 2004. Cryptology ePrint Archive: 2004/076.
[37] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *SAC*, volume 1758 of *LNCS*, pages 184–199, 1999.
[38] N. McCullagh and P. S. L. M. Barreto. A new two-party identity-based authenticated key agreement. In *CT-RSA*, volume 3376 of LNCS, pages 262–274, 2004.
[39] M. Naor. Cryptographic assumptions and challenges. In *Proc. Adv. in Cryptology (CRYPTO 2003)*, volume 2729 of *LNCS*, pages 96–109. Springer, 2003.
[40] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55:165–172, 1994.
[41] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, volume 576 of LNCS, pages 129–140, 1991.
[42] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *ACM CCS*, pages 245–254, 2000.
[43] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE S&P*, pages 184–200, 2001.
[44] M. Scott. MIRACL library. Indigo Software. `http://indigo.ie/~mscott`.
[45] M. Scott. Authenticated ID-based key exchange and remote log-in with simple token and PIN number, 2002. Cryptology ePrint Archive: 2002/164.
[46] V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, LNCS, pages 256–266, 1997. Update: `http://www.shoup.net/papers/`.
[47] E. R. Verheul. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. In *EUROCRYPT*, volume 2045 of LNCS, pages 195–210, 2001.

A Security Proof of Basic Construction

We now prove Theorem 5.1 on the security of our basic construction.

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 of towards $\mathcal{A}$.

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 $\mathcal{GM}$ 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 [17] 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 on behalf of unknown users and might later be asked to open them. 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 $\mathsf{BilinearSetup}(1^k) \rightarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g}) = params$, 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 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 ($\mathcal{U}_i$, "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): The group manager being corrupt, $\mathcal{S}$ records (corrupt-$\mathcal{GM}$, σ, m, id) in L, chooses an identity $\mathcal{U}_i$ at random among the corrupt users, and sends ($\mathcal{U}_i$, "sign", m, id) on behalf of corrupt-$\mathcal{GM}$ 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 real-world corrupted user, $\mathcal{S}$ first runs its ideal-world 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-$\mathcal{GM}$, 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 $(\mathcal{U}_i, m, \sigma)$. 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 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 EDH, and the Strong SXDH 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 A.1 *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 EDH, and the Strong SXDH 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 honest party ideal-world signature into a real-world signature, or a corrupted party real-world signature into

an ideal-world one. 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 A.2, 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 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 A.1. 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 VerifyOpen protocol with $\mathcal{S}$ on common input $(\mathcal{U}_i, m, \sigma)$ where $\mathcal{U}_i$ is not the 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 3.1 that Co-CDH is implied by the Strong SXDH 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 $C = \mathrm{PedCom}(0^{|p|})$.
  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 A.2. 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^*.* $\mathcal{A}$ produces signature $\sigma = (a_1, \ldots, a_8)$ and message m s.t. 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 EDH assumption, regardless of whether the group manager is corrupted.

  Recall that EDH takes as input $(g, g^x, \tilde{g}, \tilde{g}^x)$ together with access to oracle $O_x(\cdot)$ that takes input $c \in \mathbb{Z}_p^*$ and produces output $(g^x, \tilde{g}^{1/(x+v)}, \tilde{g}^{1/(v+c)})$ for a random $v \in \mathbb{Z}_p^*$. The goal is to produce a tuple $(c, a, a^v, \tilde{g}^{\frac{1}{x+v}}, \tilde{g}^{\frac{1}{v+c}})$ for any $a \in \mathbb{G}_1$ and any $v, c \in \mathbb{Z}_p^*$ such that c was not queried to the oracle.

  Let τ be the number of honest users in the system. When $\mathcal{A}$ succeeds with probability ε, then $\mathcal{B}$ solves the EDH problem with probability ε/τ. $\mathcal{B}$ proceeds as follows:

  1. *Setup:* $\mathcal{B}$ must establish the global parameters and key generation.
     (a) 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.
     (b) 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 $sk^* = x$.)
     (c) 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.
     (d) If the group manager is corrupt, extract the group key $GSK = (s, t)$ during the proof of knowledge.
  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 $sk^* = x$, do:
     (a) Query oracle $O_x(m)$ to get output (f_1, f_2, f_3).
     (b) Select a random $r \in \mathbb{Z}_p$. Use $GSK = (s, t)$, to output the group signature on m as
     $$(g^r, g^{tr}, g^{sr}(g^x)^{str}, (g^x)^r, (g^x)^{tr}, f_1^r, f_2, f_3).$$
  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 $(m', a_1, a_4, a_6, a_7, a_8)$ to solve the 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 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 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 $a \in \mathbb{G}_1$. (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 A.1 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 A.2 (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 SXDH assumptions, for all probabilistic polynomial-time adversaries $\mathcal{A}$, the following value is negligible in k:*

$$\begin{aligned}&\Pr\left[\mathcal{A}^{O_{u_1,u_2,\ldots,u_\tau}}(params, GPK, \{pk_i\}_{i\in[1,\tau]}) = 1\right] - \\ &\quad \Pr[\mathcal{A}^{O_{u_1,u_1,\ldots,u_1}}(params, GPK, \{pk_i\}_{i\in[1,\tau]}) = 1]\,.\end{aligned}$$

Proof. 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 SXDH assumption. Overall, if $\mathcal{A}$ succeeds with probability ε, then we can break Strong SXDH 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 SXDH. On Strong SXDH 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 SXDH with probability at least ε/τ, where τ is the number of honest users in the system. □

B Towards a Concurrent Join Protocol

In Section 5, we 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*. 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. There are, however, 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. This solution would not give a group signature scheme that can be concurrently composed with other schemes.

Solutions that would allow for concurrent executions come from applying one of the various transformations of 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. [25]). 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 [18]). 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 [28]. 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).

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.

C Open Algorithm with $O(\sqrt{n} \cdot k)$ Complexity from Trees.

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.

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 correct branch and then the correct leaf (user) for that branch. For a balanced tree, this results in a search time of $2k\sqrt{n}$. Now we present the details. During the Join protocol, the group manager secretly assigns each user to one of $\sqrt{n}k$ 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 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 C.1 *In the plain model, the above extension to the Section §5 scheme realizes $\mathcal{F}_{gs}$ from Section §2 under the the Strong LRSW, the EDH, and the Strong SXDH assumptions.*

In practice, one can achieve a "constant time" open algorithm by having less branches per node but more levels. Assume we want to be able to handle 2^{40} members. Then we could have 2^{10} branches and a tree depth of 4. This would result is a scheme where signature would have an additional 8 elements (i.e., this would double the length of the signature) the group manager would need to do at most 3072 exponentations (to walk through the first three levels) and 1024 pairings (to find the group member) to open a signature. Furthermore, the 3072 exponentations could be considerably sped-up by giving all branches of the same node the same (but random) ID except the last 10 bits. Given that opening signatures is an exceptional event, we believe such a scheme would be practical.

Open Algorithm with $O((\log n) + k)$ 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 publicly-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 a version of Elgamal encryption of their identity as the last three items:

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

To verify the signature $\sigma = (a_1, \ldots, a_{11})$, in addition to the usual CL$^+$ and BB$^+$ checks, a verifier must be sure that the ciphertext is correctly formed by checking that: (1) $e(a_1, a_{10}) = e(a_4, \tilde{g})e(a_9, \tilde{g})$ and (2) $e(a_9, eGPK) = e(a_1, a_{11})$.

Now, to the key point: 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 C.2 *In the plain model, the above extension to the Section §5 scheme realizes $\mathcal{F}_{gs}$ from Section §2 under the Strong LRSW, the EDH, and (an extension of) the Strong SXDH assumptions.*

The extension of the Strong SXDH assumption mentioned above requires changes to oracles O and Q, from Definition 4 in Section §3. 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 D.3 that Strong SXDH is hard in generic groups can be modified to cover this extended version as well.

D Generic Security of the New Assumptions

To provide more confidence in our scheme, we prove lower bounds on the complexity of our assumptions for generic groups [40, 46].

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 achieve the same asymptotic complexity bound for EDH as was shown for q-SDH.

Theorem D.1 (EDH is Hard in Generic Groups) *Let $\mathcal{A}$ be an algorithm that solves the EDH problem in the generic group model, making a total of q_G queries to the oracles computing the group action in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, the oracle computing the bilinear pairing e, and oracle $O_x(\cdot)$. If $x \in \mathbb{Z}_p^*$ and ξ_1, ξ_2, ξ_T are chosen at random, then the probability ε that $\mathcal{A}^{O_x}(p, \xi_1(1), \xi_1(x), \xi_2(1), \xi_2(x))$ outputs $(c, \xi_1(r), \xi_1(r \cdot x), \xi_1(r \cdot v), \xi_2(\frac{1}{x+v}), \xi_2(\frac{1}{v+c}))$ with $c \in \mathbb{Z}_p^*$ not previously queried to O_x, is bounded by*

$$\varepsilon \leq \frac{(q_G + 4)^2(8q + 8)}{p} = O\left(\frac{q_G^3}{p}\right).$$

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 $\tau_1 + \tau_2 + \tau_T = \tau + 4$. The only twist between our setup and that of Boneh and Boyen 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 $\mathbb{Z}_p[x, \dots]$ where additional variables will be dynamically added. 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 EDH game at step $\tau = 0$ with $\tau_1 = 2$, $\tau_2 = 2$, and $\tau_T = 0$. These correspond to the polynomials $F_{1,0} = F_{2,0} = 1$ and $F_{1,1} = F_{2,1} = x$, and the random strings $\xi_{1,0}, \xi_{1,1}, \xi_{2,0}, \xi_{2,1}$.

$\mathcal{B}$ begins the game with $\mathcal{A}$ by providing it with the 4 strings $\xi_{1,0}, \xi_{1,1}, \xi_{2,0}, \xi_{2,1}$. 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 $0 \leq i, j < \tau_1$, 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 $u \in \{0, \dots, \tau_1 - 1\}$, then $\mathcal{B}$ sets $\xi_{1,\tau_1} = \xi_{1,u}$; otherwise, it sets ξ_{1,τ_1} to a random string in $\{0,1\}^* \setminus \{\xi_{1,0}, \dots, \xi_{1,\tau_1-1}\}$. 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 $0 \leq i < \tau_1$ and $0 \leq j < \tau_2$. $\mathcal{B}$ sets $F_{T,\tau_T} \leftarrow F_{1,i} \cdot F_{2,j}$. If $F_{T,\tau_T} = F_{T,u}$ for some $u \in \{0, \dots, \tau_T - 1\}$, then $\mathcal{B}$ sets $\xi_{T,\tau_T} = \xi_{T,u}$; otherwise, it sets ξ_{T,τ_T} to a random string in $\{0,1\}^* \setminus \{\xi_{T,0}, \dots, \xi_{T,\tau_T-1}\}$. 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.

Oracle $O_x(\cdot)$**:** Let τ_v be a counter initialized to 1. $\mathcal{A}$ inputs c in $\mathbb{Z}_p^*$, followed by $\mathcal{B}$ choosing a new *variable* v_{τ_v} and setting $F_{1,\tau_1} \leftarrow v_{\tau_v}$. If $F_{1,\tau_1} = F_{1,u}$ for some $u \in \{0, \dots, \tau_1 - 1\}$, then $\mathcal{B}$ sets $\xi_{1,\tau_1} = \xi_{1,u}$; otherwise, it sets ξ_{1,τ_1} to a random string in $\{0,1\}^* \setminus \{\xi_{1,0}, \dots, \xi_{1,\tau_1-1}\}$. $\mathcal{B}$ sends ξ_{1,τ_1} to $\mathcal{A}$, adding $(F_{1,\tau_1}, \xi_{1,\tau_1})$ 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 $u \in \{0, \dots, \tau_2 - 1 + j\}$, 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 $\{0,1\}^* \setminus \{\xi_{2,0}, \dots, \xi_{2,\tau_2-1+j}\}$. $\mathcal{B}$ sends $(\xi_{2,\tau_2}, \xi_{2,\tau_2+1})$ to $\mathcal{A}$, adding $(F_{2,\tau_2}, \xi_{2,\tau_2})$ 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.

We assume SXDH holds in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ and therefore no ismorphism oracles exist.

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

$$F_{T,*} = \left(\frac{F_{1,k}}{F_{1,a}} + x\right) \cdot F_{2,d} - 1. \quad (1)$$

$$F_{T,\circ} = \left(\frac{F_{1,k}}{F_{1,a}} + c\right) \cdot F_{2,f} - 1. \quad (2)$$

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 $\xi_{1,k}$, $\tilde{g}^{1/(x+v)}$ 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 it must be the case that $F_{T,*}(x) = F_{T,\circ}(x) = 0$ for any value of x. We now argue that it is *impossible* for $\mathcal{A}$ to achieve this. Each output polynomial must be some linear combination of polynomials corresponding to elements available to $\mathcal{A}$ in

the respective groups:

$$F_{1,a} = a_0 + a_1 x + \sum_{i=1}^{q} a_{2,i} v_i \tag{3}$$

$$F_{1,b} = b_0 + b_1 x + \sum_{i=1}^{q} b_{2,i} v_i \tag{4}$$

$$F_{1,k} = k_0 + k_1 x + \sum_{i=1}^{q} k_{2,i} v_i \tag{5}$$

$$F_{2,d} = d_0 + d_1 x + \sum_{i=1}^{q} \frac{d_{2,i}}{x + v_i} + \sum_{i=1}^{q} \frac{d_{3,i}}{v_i + c_i} \tag{6}$$

$$F_{2,f} = f_0 + f_1 x + \sum_{i=1}^{q} \frac{f_{2,i}}{x + v_i} + \sum_{i=1}^{q} \frac{f_{3,i}}{v_i + c_i} \tag{7}$$

where q is the number of queries to oracle Q_x. Now, we know, by definition, that $F_{1,b}/F_{1,a} = x$, thus one can verify, using equations (3) and (4), that $a_1 = a_{2,i} = 0$ and $F_{1,a} = a_0$ is a constant.

Notation: For readability of our later analysis, we denote the following values, where *constant* values $y_0 = b_0/a_0, y_1 = b_1/a_0 + 1, y_{2,i} = b_{2,i}/a_0$:

$$Y = (F_{1,k}/a_0 + x) = y_0 + y_1 x + \sum_{i=1}^{q} y_{2,i} v_i \tag{8}$$

$$Z = (F_{1,k}/a_0 + c) = c + y_0 + (y_1 - 1)x + \sum_{i=1}^{q} y_{2,i} v_i \tag{9}$$

We also give names to the following frequently-used products:

$$P = \prod_{j=1}^{q} (x + v_j) \quad \text{and} \quad P_{i \neq j} = \prod_{i \neq j}^{q} (x + v_j) \tag{10}$$

$$Q = \prod_{j=1}^{q} (v_j + c_j) \quad \text{and} \quad Q_{i \neq j} = \prod_{i \neq j}^{q} (v_j + c_j) \tag{11}$$

$$\tag{12}$$

Using our above notation, consider the polynomials $F_{2,*}$ and $F_{T,\circ}$ from equations (1) and (2) when both sides are multiplied by PQ and we substitute in equations (6), (7), (8), and (9). For some *constants* d_i and f_i, the new polynomials:

$$PQF_{T,*} = 0 = d_0 YPQ + d_1 xYPQ + \sum_{i=1}^{q} d_{2,i} YP_{i \neq j} Q + \sum_{i=1}^{q} d_{3,i} YPQ_{i \neq j} - PQ \tag{13}$$

$$PQF_{T,\circ} = 0 = f_0 ZPQ + f_1 xZPQ + \sum_{i=1}^{q} f_{2,i} ZP_{i \neq j} Q + \sum_{i=1}^{q} f_{3,i} ZPQ_{i \neq j} - PQ \tag{14}$$

Now, we inspect equations (13) and (14). We consider two cases.

Case 1: $y_1 = 0$. Then we have $Z = c + y_0 - x + \sum_{i=1}^{q} y_{2,i} v_i$. Now, we inspect the terms of equation (14). We deduce that $f_1 = 0$, because it is the only term containing $x^{q+2} \prod_{i=1}^{q} v_i$. Then, $f_0 = 0$, because it is the only term containing $x^{q+1} \prod_{i=1}^{q} v_i$. Next, each $f_{3,i} = 0$, because they have unique terms $x^{q+1} \prod_{i \neq j} v_j$. We are left with $0 = \sum_{i=1}^{q} f_{2,i} Z P_{i \neq j} Q - PQ$. We divide by Q, and the result is $0 = \sum_{i=1}^{q} f_{2,i} Z P_{i \neq j} - P$. It follows then that at least one $f_{2,i}$ must be non-zero for the equation to be solvable; denote the first non-zero $f_{2,i}$ as $f_{2,\beta}$.

Now, suppose some constant $y_{2,i} \neq 0$ in Z, meaning that Z contains a v_i term. If $\beta \neq i$, then $y_{2,i} Z P_{i \neq j}$ contains a unique term $v_i^2 \prod_{i \neq j \neq \alpha} v_j$ that cannot be canceled. Thus, $y_{2,i} = 0$, and furthermore $f_{2,i}$, for all $i \neq \beta$. Now we are left with the equation $0 = f_{2,\beta}(c + y_0 - x + y_{2,\beta} v_\beta) P_{\beta \neq j} - P$, we divide out $P_{\beta \neq j}$ and observe that $f_{2,\beta} = -1$, to get $0 = -(c + y_0 - x + y_{2,\beta} v_\beta) - (x + v_\beta)$.

Now, since $c, y_0, y_{2,\beta}$ are all constants and v_β is a variable, we conclude that $y_{2,\beta} = 1$ and $c = -y_0$. That means that $Y = y_0 + v_\beta$, where $y_0 \neq 0$. Plugging back into equation (13), we have $d_1 = 0$ due to unique x^{q+1} term, then it must be the case that $d_0 = 0$ due to $x^q v_\beta \prod_{i=1}^{q} v_i$. Next, it must be that $d_{3,i} = 0$ for all $i \neq \beta$ due to unique $x^q v_\beta^2 \prod_{j \neq i \neq \beta} v_j$. Next, we see that terms corresponding to $d_{3,\beta} Y P Q_{\beta \neq j} = d_{3,\beta}(y_0 + v_\beta) P Q_{\beta \neq j}$ and PQ are the only two left with a x^q term; thus, $d_{3,\beta} \neq 0$. Further, cancelling the term $x^q \prod_{i=1}^{q} v_i$ from PQ requires that $d_{3,1} = 1$. Thus, we find that to cancel all related x^q terms, it must be that $c = c_\beta$. Since c, which represents the message corresponding to $\mathcal{A}$'s signature, is an old value, this is not a valid forgery.

Case 2: $y_1 \neq 0$. Now Y contains an x term, and we inspect equation (13). We deduce that $d_1 = 0$, because it is the only term containing $x^{q+2} \prod_{i=1}^{q} v_i$. Then, $d_0 = 0$, because it is the only term containing $x^{q+1} \prod_{i=1}^{q} v_i$. Next, each $d_{3,i} = 0$, because they have unique terms $x^{q+1} \prod_{i \neq j} v_j$. We are left with $0 = \sum_{i=1}^{q} d_{2,i} Y P_{i \neq j} Q - PQ$. We divide by Q, and the result is $0 = \sum_{i=1}^{q} d_{2,i} Y P_{i \neq j} - P$. To satisfy this equation, for some $d_{2,i}$, we must have $d_{2,i} \neq 0$. We denote this value $d_{2,\beta}$.

From this point, we proceed in a fashion similar to case 1. By inspecting the above equation, we find that $y_0 = 0$, and for all $i \neq \beta$, $y_{2,i} = 0$, otherwise there exist unique terms: e.g., $v_\beta \prod_{\beta \neq j}^{q} v_j$. Furthermore, $d_{2,\beta} y_1 = 1$, since the x^q term of P has coefficient 1. And, $d_{2,\beta} y_{2,\beta} = 1$, since the $\prod_{i=1}^{q} v_i$ term of P has coefficient 1. So, $y_1 = y_{2,\beta} = 1/d_{2,\beta}$ and we plug into Z as $Z = c + (1/d_{2,\beta} - 1)x + v_\beta / d_{2,\beta}$.

From equation (14), we have that $f_1 = 0$ due to $x^{q+1} \prod_{i=1}^{q} v_i$, $f_0 = 0$ due to $v_\beta x^q \prod_{i=1}^{q} v_i$. For all $i \neq \beta$, $f_{3,i} = 0$, otherwise unique terms $v_\beta^2 x^q \prod_{i \neq j \neq \beta} v_j$ appear. Given that all $f_{3,i} = 0$ except for $f_{3,\beta}$, it follows all $f_{2,i} = 0$ except for $f_{2,\beta}$ due to unique terms containing v_i^3 for $i \neq \beta$. Thus, we now have the equation $0 = f_{2,\beta} Z P_{\beta \neq j} Q + f_{3,\beta} Z P Q_{\beta \neq j} - PQ$. We divide by $P_{\beta \neq j} Q_{\beta \neq j}$ to obtain $0 = f_{2,\beta} Z(v_\beta + c_\beta) + f_{3,\beta} Z(x + v_\beta) - (x + v_\beta)(v_\beta + c_\beta)$.

Now, suppose $d_{2,\beta} = 1$ and thus $Z = (c + v_\beta)$. Then we know that $f_{3,\beta} = 1$ to cancel the term $x v_\beta$; this forces $f_{2,\beta} = 0$ because the v_β^2 term is already canceled in whole by the $f_{3,\beta}$ component. Thus, it is immediate that $c = c_\beta$, which is not a valid forgery.

On the other hand, suppose $d_{2,\beta} \neq 1$ and thus Z contains an x term. Then, we know that $f_{3,\beta} = 0$, because its x^2 term would be unique. This forces $c = 0$ because otherwise the constant term $f_{2,\beta} c c_\beta$ would be unique. However, c must be an element in $\mathbb{Z}_p^*$, and thus this is also not a valid forgery.

Thus, we conclude that $\mathcal{A}$'s success depends *solely* on his luck when the variables are 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 15,16,17) 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 18, 19) 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:

$$F_{1,i}(x^*) - F_{1,j}(x^*) = 0, \quad \text{for some } i,j \text{ such that } F_{1,i} \neq F_{1,j}, \tag{15}$$
$$F_{2,i}(x^*) - F_{2,j}(x^*) = 0, \quad \text{for some } i,j \text{ such that } F_{2,i} \neq F_{2,j}, \tag{16}$$
$$F_{T,i}(x^*) - F_{T,j}(x^*) = 0, \quad \text{for some } i,j \text{ such that } F_{T,i} \neq F_{T,j}, \tag{17}$$
$$F_{T,*}(x^*) = 0, \tag{18}$$
$$F_{T,\circ}(x^*) = 0. \tag{19}$$

We observe that $F_{T,*}$ and $F_{T,\circ}$ are non-trivial polynomials of degree at most $\leq 2q+2$. Each polynomial $F_{1,i}$ and $F_{2,i}$ has degree at most 1 and $2q+1$, respectively.

For fixed i and j, the first case occurs with probability $\leq 1/p$; the second occurs with probability $\leq (2q+1)/p$; and the third occurs with probability $\leq (2q+2)/p$. (We already take into account multiplying out the denominators of any rational polynomials.) Finally, the fourth and fifth cases happen with probability $\leq (2q+2)/p$. Now summing over all (i,j) pairs in each case, we bound $\mathcal{A}$'s overall success probability $\varepsilon \leq \binom{\tau_1}{2}\frac{1}{p} + \binom{\tau_2}{2}\frac{2q+1}{p} + \binom{\tau_T}{2}\frac{2q+2}{p} + \frac{2(2q+2)}{p}$. Since $\tau_1+\tau_2+\tau_T \leq q_G+4$, we end with $\varepsilon \leq (q_G+4)^2(8q+8)/p) = O(q_G^3/p)$. □

The following corollary is immediate.

Corollary D.2 *Any adversary that breaks the 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 SXDH 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 D.3 (Strong SXDH 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 SXDH 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 D.1. (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 D.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 $\mathbb{Z}_p^*$, followed by $\mathcal{B}$ choosing a new *variable* v_{τ_v} and setting $F_{1,\tau_1} \leftarrow v_{\tau_v}$. 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 $\mathcal{A}$, adding $(F_{1,\tau_1}, \xi_{1,\tau_1})$ 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 $\mathcal{A}$, adding $(F_{2,\tau_2}, \xi_{2,\tau_2})$ 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, except that 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:

$$F_{1,a} = a_0 + a_1 \cdot x + a_{2,i} \cdot v_i + a_{3,j} \cdot r_j + a_{4,j} \cdot r_j \cdot y + a_{5,j} \cdot r_j \cdot w_j \tag{20}$$

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:

$$F_{2,b} = b_0 + \frac{b_{1,i}}{x + v_i} + \frac{b_{2,i}}{v_i + m_i} + \frac{b_{3,j}}{y + w_j} + \frac{b_{4,j}}{w_j + m_j} \tag{21}$$

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 (21) to obtain:

$$\begin{aligned} F'_{2,b} \quad = \quad & b_0(x + v_i)(v_i + m_i)(y + w_j)(w_j + m_j) + \\ & b_{1,i}(v_i + m_i)(y + w_j)(w_j + m_j) + b_{2,i}(x + v_i)(y + w_j)(w_j + m_j) + \\ & b_{3,j}(x + v_i)(v_i + m_i)(w_j + m_j) + b_{4,j}(x + v_i)(v_i + m_i)(y + w_j) \end{aligned} \tag{22}$$

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:

$$F_{T,c} = \sum F_{1,a} \cdot F_{2,b}. \tag{23}$$

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

$$F'_{T,c} = \sum F_{1,a} \cdot F'_{2,b}. \tag{24}$$

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:

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

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.

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

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 (25), 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:

$$F_{1,i}(x^*, y^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) - F_{1,j}(x^*, y^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) = 0 : \text{for some } i, j, \text{ such that } F_{1,i} \neq F_{1,j}, \tag{27}$$

$$F_{1,i}(x^*, x^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) - F_{1,j}(x^*, x^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) = 0 : \text{for some } i, j, \text{ such that } F_{1,i} \neq F_{1,j}, \tag{28}$$

$$F_{2,i}(x^*, y^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) - F_{2,j}(x^*, y^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) = 0 : \text{for some } i, j, \text{ such that } F_{2,i} \neq F_{2,j}, \tag{29}$$

$$F_{2,i}(x^*, x^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) - F_{2,j}(x^*, x^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) = 0 : \text{for some } i, j, \text{ such that } F_{2,i} \neq F_{2,j}, \tag{30}$$

$$F_{T,i}(x^*, y^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) - F_{T,j}(x^*, y^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) = 0 : \text{for some } i, j, \text{ such that } F_{T,i} \neq F_{T,j}, \tag{31}$$

$$F_{T,i}(x^*, x^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) - F_{T,j}(x^*, x^*, \{v_d^*\}, \{w_d^*\}, \{r_d^*\}) = 0 : \text{for some } i, j, \text{ such that } F_{T,i} \neq F_{T,j}. \tag{32}$$

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 (20). 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 (22). 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 (24).

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 D.4 *Any adversary that breaks the Strong SXDH assumption with constant probability $\gamma > 0$ in generic groups of order p requires $\Omega(\sqrt[3]{\gamma p})$ generic group operations.*