Some Open Issues and New Directions in Group Signatures

Giuseppe Ateniese and Gene Tsudik

USC Information Sciences Institute, 4676 Admiralty Way, Marina Del Rey, CA 90292, USA

Abstract. Group signatures allow any member of a potentially large group to sign on behalf of the group. Group signatures are anonymous and unlinkable for everyone with the exception of a designated group manager who can co-relate signatures and reveal the identity of the actual signer. At the same time, no one (including a group manager) can misattribute a valid group signature. Group signatures are claimed to have many practical applications in e-commerce as well as in military and legal fields.
Despite some interesting and eclectic results, group signatures remain confined to academic literature. The focus of this paper is two-fold. First, it discusses certain issues that stand in the way of practical applications of group signatures and uses the example of on recent group signature scheme to illustrate certain problems. Second, this paper (informally) introduces some practical security services that can be constructed using any group signature scheme. Sample realizations of these services are provided.

Keywords: Public-key Cryptography, Digital Signatures, Group Signatures, Multi-Group Signatures, Sub-Group Signatures, Coalition Attacks, Revocation.

1 Introduction

Digital signatures are rapidly becoming ubiquitous in many aspects of *electronic* life. They are used to obtain security services such as authentication, data integrity and non-repudiation. **Group** digital signatures are a relatively new concept introduced by Chaum and Van Heyst in 1991 [6]. A group signature – similar to its traditional counterpart – allows a signer to demonstrate knowledge of a secret with respect to a specific document. It is also publicly verifiable: anyone in possession of a group public key can validate a group signature. However, group signatures are anonymous in that, no one, with the sole exception of a designated group manager, can discover the identity of the signer. Furthermore, group signatures are unlinkable which makes it computationally hard to establish whether or not multiple signatures are produced by the same group member. In exceptional cases (such as a legal dispute) any group signature can be "opened" by the group manager to reveal unambiguously the identity of the actual signer.

At the same time, no one (including the group manager) can misattribute a valid group signature.

The salient features of group signatures make them attractive for many specialized applications such as voting and bidding. More generally, group signatures can be used to conceal organizational structures, e.g,, when a company or a government agency issues a signed statement. They can also be integrated with electronic currency mechanisms (as blind group signatures) [8] to provide both anonymity and untraceability of currency issuers.

Despite appealing features and interesting research papers, group signatures remain in the domain of theoretical results. To this end, the purpose of this paper is twofold. First, we briefly review the state of the field and try to examine certain issues that hinder wider acceptance and use of group signatures. Second, we discuss and demonstrate some practical services that can be built on top of group signature schemes.

The paper is organized as follows. Next two sections summarize basic properties and features of group signatures and provide a brief overview of previous work. Then, Section 4 discusses some open issues and, to illustrate the point, Section 6 presents a weakness in a recently proposed scheme. Sections 7-9 introduce two new services built upon secure group signatures and sketch out example schemes.

2 Preliminaries

We begin with a definition of group signatures.

Definition 1. *A* `group signature scheme` *is a digital signature scheme comprised of the following procedures:*

- `SETUP` *an algorithm for generating the initial group public key $\mathcal{Y}$ and a group secret key $\mathcal{S}$.*
- `JOIN` *a protocol between the group manager and a user that results in the user becoming a new group member.*
- `SIGN` *a protocol between a group member and a user whereby a group signature on a user-supplied message is computed by the group member.*
- `VERIFY` *an algorithm for establishing the validity of a group signature given a group public key and a signed message.*
- `OPEN` *an algorithm that, given a signed message and a group secret key, determines the identity of the signer.*

A group signature scheme must satisfy the following **security properties**:

- **Unforgeability.** Only group members are able to sign messages on behalf of the group.
- **Anonymity.** Given a signature, identifying the actual signer is computationally hard for everyone but the group manager.
- **Unlinkability.** Deciding whether two different signatures were computed by the same group member is computationally hard.

- **Exculpability.** Neither a group member nor the group manager can sign on behalf of other group members.[1]
- **Traceability.** The group manager is able to open a signature and identify the actual signer; moreover, a signer cannot prevent the opening of a valid signature.
- **Coalition-Resistance.** A colluding subset of group members cannot generate valid group signatures that cannot be traced.

The **efficiency** of a group signature scheme is typically based on the following parameters:

- The size of the group public key $\mathcal{Y}$.
- The size of a group signature.
- The efficiency of `SIGN` and `VERIFY`.
- The efficiency of `SETUP`, `OPEN` and `JOIN`.

3 Previous Work

The concept of group signatures was introduced and realized by Chaum and Van Heist in 1991 [6]. They proposed four schemes of which three provide computational anonymity whereas the fourth provides information-theoretic anonymity. Some of schemes do not allow a group manager to add group members after the initial setup. Others require the group manager to contact each member in order to open a signature, i.e., to reveal the identity of the signer.

A number of improvements and enhancements followed the initial work. Some notable results are due to Chen/Pedersen [7], Camenisch [3], Petersen [9] and Camenisch/Stadler [4]. In [7], two schemes were proposed providing information theoretic and computational anonymity, respectively. Unfortunately, the proposed scheme allows the group manager to misattribute a signature, i.e., falsely accuse a group member of having signed a message.

In [3] an efficient group signature scheme was presented, providing computational anonymity, ability to add (or remove) group members after the initial setup, and the possibility of being generalized by allowing authorized set of group members to sign collectively (appearing to verifiers as a single signer) on behalf of the group. This scheme can be extended to allow the functionality of the group manager to be shared among several entities. The drawbacks include the size of the public key and the signature size (both proportional to group size).

As also noted in [4], most previous results exhibit some or all of the following shortcomings:

1. The size of the group public key depends on the size of the group.
2. The length of a group signature depends on the size of the group.
3. New member addition requires restarting the entire system or involves reissuing all members' keys and changing the group public key.

[1] Note that the above does not preclude the group manager from creating fraudulent signers (i.e., nonexistent group members) and then producing group signatures.

4. Exclusion/revocation of group members requires re-issuing all members' keys and changing the group public key.

The state-of-the-art in the field is exemplified by the recent result of Camenisch and Stadler [4]. It addresses the first three of the above issues; albeit, at significant cost. The basic scheme (referred hereafter as CS97), while quite elegant, involves costly computation and reliance on new problems conjectured to be computationally difficult to solve. (One such problem is the difficulty of computing double discrete logarithms in finite groups. Another – the difficulty of computing roots of discrete logarithms.) Nonetheless, CS97 is, in principle, simple and appealing in that the group public key and the group signature are both of constant size. The computational complexity of SIGN and VERIFY are also independent of group size.

4 What Stands in the Way?

Schemes such as CS97 make group signatures more attractive and bring the entire concept closer to practice. While there is still no single *killer app*, a number of applications can be envisaged for group signatures. Examples include:

- Government, military and commercial press releases
- Email messages in anonymous support groups
- Electronic cash (with banks forming a cash-issuing group)
- Issuance of stock, bond and other financial reports
- Voting applications
- Document signing by certified (but anonymous) professionals, e.g., lawyers, notaries public, auditors, etc.

There remain, however, some bothersome issues that stand in the way of real-world applications and deployments of group signatures. In this paper we concentrate on two that we believe to be the most important:

- Security against coalitions
- Member deletion (revocation) complexity

The first problem is straight-forward. If a scheme is vulnerable to coalition attacks, its utility is limited to environments where member collusion is not a threat. Although such environments might exist[2], in general, coalition resistance must be **provable**. This is, unfortunately, not achieved by any existing scheme. In order to better illustrate this point, in Section 6 we present a coalition-related weakness in a recently proposed group signature scheme.

The second problem is more subtle. It is related to the notoriously difficult task of credential revocation in regular (non-group) public key signature schemes. However, as discussed below, revocation in the context of group signatures is a more complicated problem.

[2] For instance, electronic lotteries.

5 Member Deletion

One intuitive and seemingly effective way to delete group members is via Certificate Revocation Lists (CRLs). A group manager periodically issues a new CRL wherein each entry identifies a group member (or a specific group certificate) whose membership has been revoked. There are many ways to generate and distribute CRLs. Unfortunately, the CRL approach is unsuitable for group signatures for the following reasons:

- Since group signatures are anonymous and unlinkable, how can the group manager "identify" a deleted member?
- If the group manager reveals certain (hereto secret) values corresponding to a deleted member, can the anonymity and unlinkability of **past signatures** of the said member be preserved?

The last item is probably the most important issue in member deletion. The preservation of security of past signatures after deletion is reminiscent of Perfect Forward Secrecy (PFS), a property often sought in key distribution protocols. If a member leaves the group voluntarily or a member is excluded for some administrative reason (such as a job promotion) there is nothing to gain (and much to loose) in sacrificing the security of past signatures.

However, in some cases, PFS is not desirable. For instance, suppose that a malicious member of a stock brokerage firm has been detected. It is surely beneficial – in the process of deletion – to inform all potential verifiers (clients) who might hold old signatures by the crooked broker.

A more interesting scenario is the compromise of a member's secrets. Suppose that a member's group certificate and/or group private key are inadvertently divulged. The group manager faces a real problem: to reveal this member's secrets means that all of its past signatures can be linked! Since there is no reason to doubt the validity of past signatures (the deleted member was not malicious) sacrificing unlinkability is clearly undesirable.

Another member deletion tactic is to change the group public key. After generating the new group public key, the group manager simply issues a new set of membership certificates to all remaining members while the rest are automatically excluded. This is an acceptable solution for small and stable groups. In large groups (e.g., lawyers in the state of California) this approach is unworkable due to frequent membership changes (e.g., lawyers being disbar-ed), the computational expense of re-issuing membership certificates and change notifications for all remaining members. Also, we assumed thus far that a member's new certificate can be issued unilaterally (off-line) by the group manager. However, some schemes may require interaction as part of repeated certificate issuance. (Initial certificate issuance always requires interaction). This complicates member deletion even further.

In summary, we have only *scratched the surface* of the member deletion problem. Since it appears that no proposed group signature scheme adequately addresses it, efficient and secure member deletion remains a pressing and interesting open problem.

6 Coalition Resistance: Quasi-Attacks on CS97

Many ground-breaking cryptographic protocols are based on new number theoretic assumptions. There are two ways to invalidate these assumptions. One is by proving that they are false, the other – by proving that they are unsuitable for a specific protocol. The latter is the case with one of the assumptions underlying the Camenisch and Stadler's CS97 group signature scheme [4]. We briefly present some attacks on CS97 and then propose some simple ways of preventing them.

Remark 1. The following text describes two attack scenarios. It is necessary to stress that these are not all-out attacks that really *break* the scheme. Instead, we refer to them as *quasi-attacks* (for lack of better term) that exploit or break only one specific assumption.

A *coalition* attack occurs when a subset of group members pool together their secrets and generate perfectly valid group signatures. Such signatures have the property that even the group manager is unable to open them. The coalition resistance of Camenisch-Stadler protocol [4] is based on the following number-theoretic assumption:

> Let n and e be RSA-like parameters and $a \in \mathbb{Z}_n^*$ an element of large multiplicative order both the factors of n such that computing the discrete logarithms to the base a is infeasible. It is hard to compute a pair (x, v) of integers such that
>
> $$v^e \equiv 1 + a^x \bmod n$$
>
> holds, if factorization of n is unknown. Furthermore, this is true even when other pairs (x', v'), satisfying the above equation, are known.

Loosely speaking, given one or more group certificate of the form $(a^x + 1)^d$, where $d \cdot \ e \equiv 1 \bmod \phi(n)$, it should be hard to generate a new valid group certificate without the aid of the group manager.

However, consider the following special cases. Let $(a^x + 1)^d$ be a valid group certificate. It can be re-written as $(a^x + 1)^d = a^{xd}(1 + a^{-x})^d$. Now, suppose that $x = ke$ for some integer k, so that $a^{xd}(1 + a^{-x})^d = a^k(1 + a^{-x})^d$. Obviously, multiplying it by the inverse of $a^k \bmod n$ we get the new certificate $(1 + a^{-x})^d$. This attack allows a *single* group member to produce a valid and untraceable signature.

This attack can be trivially prevented, for example, by requiring the group member to prove that x is relatively prime with e (i.e. gcd(x,e)=1) without revealing x itself. However, as we now demonstrate, it is possible to generate a valid group certificate even when x is a generic value, e.g., if gcd(x,e)=1.

1. The first member registers and obtains a certificate A=$(a^x + 1)^d = a^{xd}(1 + a^{-x})^d$.

2. The second member then registers a^{-x} and obtains $B = (a^{-x} + 1)^d$ as its certificate.
3. They compute the inverse of B modulo n (using Euler's algorithm) and multiply it by A. The end-result is that two colluding members obtain the value $a^{xd} = A \cdot B^{-1} = a^{xd}(1 + a^{-x})^d \cdot (1 + a^{-x})^{-d} \bmod n$.

   Remark 2. Possession of this value is dangerous: colluding members can take different strategies to generate a new group certificate. For instance, the value a^{xd} can be exponentiated with a random c, obtaining a^{cxd} and its inverse $a^{-cxd} \bmod n$.

4. Then, a third group member registers a^{cx} and obtains $(a^{cx} + 1)^d$ as its certificate. Multiplying $(a^{cx} + 1)^d$ by a^{-cxd} yields a new and untraceable certificate $(a^{-cx} + 1)^d$ (in fact, $(a^{cx} + 1)^d \cdot a^{-cxd} = (a^{cx-cx} + a^{-cx})^d)$. It is also possible to generalize this strategy.

A simple way to prevent this kind of attack is to substitute 1 in the certificate structure $(a^x + 1)^d$ with a random value t resulting in $(a^x + t)^d$ as the new certificate form.[3] The scheme must be modified slightly in order to work with this new certificate form but the extra cost is a single exponentiation. Another simple fix, not mutually exclusive with the above, is to randomize (or "salt") the new member's membership key (and certificate) as part of JOIN.

However, we cannot prove that the proposed fixes result in a secure form of membership certificates, i.e., that coalition attacks are, in fact, infeasible. We only showed that 1, as a redundancy factor, is a wrong choice for the CS97 scheme. However, we would like to propose a new challenge with this new certificate's form, i.e. $(a^x + t)^d$ where t is a random value and x is the group member's secret exponent randomized by the group manager. In the rest of the paper we will make use of this modified CS97 scheme.

7 Multi-Group Signatures

Thus far, we discussed some outstanding issues that block wider acceptance and use of group signatures. Lest the overall theme of this paper appear downbeat, we now switch gears and turn to "new and exciting" uses and extensions of group signatures.

One novel and practical service that can be built using group signatures as a primitive is **multi-group signatures**. (Not to be confused with so-called multi-signatures [1].)

Informally, we define a multi-group signature (MGS) as a set of regular group signatures generated by the same entity (signer) who is a member of multiple groups. For example, a user belonging to two groups A and B may have a reason to sign something as a member of both groups. The need for such signatures may arise in real-life situations as illustrated in the following (admittedly somewhat contrived) examples:

[3] This has been also suggested by J. Camenisch (via private communication).

1. A financial (e.g., loan) contract may need to be signed by an authorized loan officer of a bank and by a notary public. If the same person is a member of both groups (loan officers and notaries public) s/he can sign the document with a multi-group signature.
2. A republican senator (anonymously) releasing a statement to the media needs to assure the public that s/he is, in fact, a member of the senate and a republican party member. This can be achieved with a double-group signature.
3. An expert witness testifying in court (or in congress) who is afraid of retribution may need, nonetheless, to convince the public that s/he is, say, a medical doctor and an officer (of a certain rank) in the armed forces. Once again, a double-group signature is appropriate.
4. A person submitting a formal peer complaint to a state judicial review panel might have to sign the complaint text simultaneously as a: i) US citizen, ii) resident of that state and, iii) member of the bar. This requires a triple-group signature.

An interesting property required of MGS is the ability to **link** the individual group signatures comprising an MGS. In other words, it must be evident that the same signer produced each single-group signature which is part of an MGS.

At first, this might seem both counter-intuitive and counter-productive. Recall that one of the basic tenets of group signatures is **unlinkability**, i.e., the computational difficulty of establishing whether two distinct group signatures were produced by the same group member. However, we observe that the property we seek is **linkability** of group signatures **across different groups** rather than linkability of different signatures within a group. Thus, the desired property is not inherently in conflict with traditional group signatures.

We note that, in all other respects, an MGS scheme must have the same properties as a regular group signature scheme.

Now, whether an MGS scheme can be built out of existing group signature schemes is an entirely separate issue. We have considered several group signature schemes from the literature. Some are not easily converted, e.g., [3]. However, at least one scheme, CS97 [4], can be used as building block for a secure MGS scheme.

8 MGS Example

We begin with a short overview of the CS97 basic scheme, modified as shown in section 6.

The scheme is premised on two important building blocks: proofs (signatures) of knowledge of double discrete logarithms (SKLOGLOG) and signatures of knowledge of roots of discrete logarithms (SKROOTLOG). SKLOGLOG entails, given d, demonstrating knowledge of $log_b(log_a\ d)$. SKROOTLOG entails, given c, and e demonstrating knowledge of $(log_a c)^{1/e}$ where $(e, d = 1/e)$ is an RSA key-pair.

As part of setup, *GRMGR* selects the following values:

- an RSA public-key parameter $n = pq$ for two large primes p, q and an RSA key-pair (e, d) $e \cdot d \equiv 1 \bmod n$.
- a cyclic group $G =< g >$ of order n where computing discrete logarithms is infeasible.
- an element $a \in \mathbb{Z}_n^*$ of large multiplicative order modulo both p and q.
- some other parameters not relevant to the ensuing discussion.

A prospective group member can join the group by selecting a random value x_1 randomized by *GRMGR* resulting in the secret x (known only by the group member), computing $y = a^x \bmod n$ and $z = g^y$. Upon sending y to *GRMGR* and proving knowledge of x, the member is issued a certificate: $(y + t)^d$.

A group signature over a message m is defined in CS97 as a tuple $< m, \bar{g}, \bar{z}, V_1, V_2 >$ where:

- m is the message to be signed
- $\bar{g} := g^r$ for $r \in_R \mathbb{Z}_n^*$
- $\bar{z} := \bar{g}^y$
- $V_1 :=$ SKLOGLOG$[x \ : \ \bar{z} = \bar{g}^{a^x}](m)$
- $V_2 :=$ SKROOTLOG$[v \ : \ \bar{z}\bar{g}^t = \bar{g}^{v^e}](m)$ for $v = (y + t)^d$

To verify a group signature, it is sufficient to check V_1 and V_2. We do not elaborate on other details/features of the scheme (such as the OPEN procedure) or its security properties as they are not germane to our discussion.

To further illustrate the MGS concept, we consider the specific case of double-group signatures; the technique can be trivially extended to n-group signatures. Suppose that the same group manager is used for both groups G_1 and G_2. (This is not an inherent requirement but it simplifies the example.) The setup procedure is almost the same as in CS97, i.e., *GRMGR* picks n, $G =< g >$, a and other parameters as above. These values are common for both groups. However, *GRMGR* selects different public key-pairs: (e_1, e_1^{-1}) and (e_2, e_2^{-1}) for G_1 and G_2, respectively.

The JOIN procedure is largely the same as in CS97 except that a member who joins both groups (and is interested in computing MGSs) must use the same secret value x in both registrations[4]: $y_1 = y_2 = a^x \bmod n$. The two respective membership certificates are: $v_1 = (y + t)^{e_1^{-1}}$ and $v_2 = (y + t)^{e_2^{-1}}$.

To produce an MGS (a double-group signature, in this case) a member of both groups constructs a tuple $< m, SIG_1, SIG_2, V_3 >$ where:

- m is the message being signed.
- $SIG_1 =< \bar{g_1}, \bar{z_1}, V_{1,1}, V_{2,1} >$ is a single-group signature of message m where:
- $SIG_2 =< \bar{g_2}, \bar{z_2}, V_{1,2}, V_{2,2} >$ is also a single-group signature of m where:
- $V_3 =$ SKEQLOG$(\bar{z_1}, \bar{g_1}, \bar{z_2}, \bar{g_2})$ is a signature of knowledge of equality of two discrete logarithms (of $\bar{z_1}$ and $\bar{z_2}$) with two different bases $\bar{g_1}$ and $\bar{g_2}$. More generally, SKEQLOG denotes a non-interactive proof of $log_{\bar{g_1}} \bar{z_1} = log_{\bar{g_2}} \bar{z_2}$.

[4] There does not appear to be any security risk in doing this.

One efficient technique for constructing SKEQLOG is due to Chaum and Pedersen [5][5].

As in CS97, $V_{1,1}$ and $V_{1,2}$ each demonstrate knowledge of the secret x, i.e., double-discrete logarithm of $\bar{z_1}$ and $\bar{z_2}$, respectively (which is x in both cases). $V_{2,1}$ and $V_{2,2}$ demonstrate knowledge of e_1-th (and e_2-th) roots of discrete logarithm of b_1 (and b_2), i.e., possession of membership certificates of groups G_1 and G_2. Finally, V_3 demonstrates equality of two discrete logarithms (z_1 and z_2) with two different bases (g_1 and g_2). The last step is crucial to "linking" the two signatures since it asserts that the same y is used in both signatures. Consequently, the same secret x is used in both signatures (since $y = a^x$.)

It appears that no reduction in security is incurred by using this MGS construct. (Assuming, of course, that the underlying CS97 basic scheme is itself secure.) In particular, the OPEN procedure remains the same and MGSs remain unlinkable much like their single-group counterparts.

A potential concern with the above extension is its relative inefficiency. since a generalized i-group MGS involves computing i regular single-group signatures and $(i-1)$ SKEQLOG-s. However, most expensive constructs are SKROOTLOGs and SKLOGLOGS required by CS97.

To conclude the MGS discussion, we need to point out that there is no compulsion to use the same group manager for MGS. In the CS97-based example above, the group public key is:

$$\mathcal{Y} = [n, < g >, a, e, \text{other parameters}]$$

If two groups are managed separately, the two managers can independently compute their respective group public and private keys. A member registers with the same secret x in both groups and obtains two corresponding certificates. However, the problem arises when the member tries to prove that s/he uses the same x in generating SIG_1 and SIG_2. Since each group has a distinct base a, the member has to prove **equality of two double discrete logarithms**.

9 Sub-Group Signatures

Another interesting outgrowth of conventional group signatures is the notion of **sub-group signatures**. Informally, we define a sub-group signature (SGS) as a function of a member group $GR = \{M_1, ..., M_t\}$, a subset $S \subset GR$ and a message m as:

$$\mathcal{T} = \{SIG_j(m) | M_j \in S\} \;\; \#\mathcal{T} = i$$

Each $SIG_j(m)$ is a normal group signature by a group member M_j; for a given M_j, at most one SIG_j is in $\mathcal{T}$.

[5] More formally, a pair (c, s) satisfying $c = H(m||y_1||y_2||a||b||a^s y_1^c||b^s y_2^c)$ is a Schnorr signature of the discrete logarithm of both $y_1 = a^x$ w.r.t. the base a and $y_2 = b^x$ w.r.t. the base b, on the message m. Only the party in possession of the secret x is able to compute the signature, if $x = \log_a(y_1) = \log_b(y_2)$, by choosing a random t and then computing c and s as $c = H(m||y_1||y_2||a||b||a^t||b^t)$, $s = t - cx$ (in $\mathbb{Z}$).

We should stress that the notion of SGS is very different from that of coalition group signatures as defined by Camenisch in [3]. Coalition group signatures are indistinguishable from single-member group signatures by verifiers. Only the group manager can (via OPEN) determine the coalition size and structure. In contrast, in SGS the size of the signing sub-group (or coalition) must be explicit and verifiable by anyone.

(As a sidenote, the scheme in [3] includes a method for obtaining coalition signatures; however, the coalition structures are fixed a priori by the group manager. This limits the utility of the whole concept.)

Consider the following example applications of SGS:

- A petition is circulated among members of a certain group. A number of members (i) sign the petition and then publicly announce that i members stand behind it. Any insider or outsider is able to verify that i distinct (and anonymous) members have indeed signed the petition.
- A military order must be signed by a least t ranked officers. Without exposing any internal command structure, t officers sign the order thus allowing anyone to verify its validity.
- A prospective club member must be vouched for by at least n existing members. These members must be distinct and, while n might be pre-set, there is no upper limit on the number of vouching members. The more members vouch, the faster the new member is admitted.
- A vote on a proposition is conducted by members of the board. Each member is given a single vote and vote tallies must be publicly verifiable.

It is also important to point out the differences between SGS and threshold schemes. Threshold schemes aim to partition a secret (e.g., a signature key) in a way that allows at least a pre-determined number (threshold) of shareholders to re-create a secret. In contrast, SGS does not involve a pre-determined threshold; any number of group members may sign a given message. The central goal of SGS is to demonstrate that a subset of a certain (unknown a priori) size has signed a given message.

In addition to the security properties of regular group signatures, a SGS scheme must feature:

- Compositional integrity: assure the verifier that all signatures comprising a SGS have been generated by distinct signers (group members).
- SGS unlinkability: multiple SGS-s generated by the same sub-group must be unlinkable. We note that this property does not necessarily follow from the unlinkability of the underlying group signature scheme (the latter only assures unlinkability of single-member signatures.)

SGS unlinkability is a natural property. Clearly, if single-member signatures are unlinkable, so should SGS-s. Compositional integrity, on the other hand, seems to contradict the unlinkability of regular group signatures. Recall that in Section 2 we defined **unlinkability** (in the context of regular group signatures) as the difficulty of deciding whether two distinct signatures were generated by the same group member. Indeed, we may need to weaken the unlinkability property in order to achieve compositional integrity. However, we emphasize that this should be done **only** for SGS; i.e., the structure of other types of group signatures (regular, multi-group) must remain unchanged.

The feasibility of SGS schemes is illustrated by a concrete example in the next section. Once again, we use the CS97 [4] group signature scheme as our main building block.

10 SGS Example

The actual SGS extension is quite trivial. The SETUP, JOIN and OPEN procedures are the same as in CS97. Only SIGN and VERIFY are modified slightly.

The modified SIGN procedure involves a new initial stage whereby the i multiple signers (sub-group members) agree on a common random, one-time base:

$$\bar{g} := g^r \text{ for } r \in_R \mathbb{Z}_n^*$$

In CS97, r (and, hence, $\bar{g}$) is generated at random by one signer. In our case, r needs to be generated collectively by the sub-group of signers. There are many methods for doing this; one possibility is to run a group Diffie-Hellman key agreement protocol (e.g., [2] or [10]) and, as a result, generate a common:

$$\bar{g} := g^r \text{ where } r = a^{r_1 \cdots r_i} (mod n)$$

and i is the size of the sub-group and each r_j is generated at random by the j-th signer.

Having agreed on a one-time base $\bar{g}$, each signer (M_j) computes a regular CS97 group signature: $SIG_j(m) =< m, \bar{g}, \bar{z}, V_{1,j}, V_{2,j} >$. Pooled together, the signatures form an SGS, $\mathcal{T} = \{SIG_j | 0 < j \leq i\}$. Of course, redundant m and $\bar{g}$ fields can be eliminated from $\mathcal{T}$.

The VERIFY procedure is also modified in accordance to the parameter i, the size of the (claimed) sub-group. The same $\bar{g}$ must be used as a base in verifying every signature in $\mathcal{T}$. In addition, for each $SIG_j(m)$ the verifier must check the uniqueness of $\bar{z}$ contained therein. (An occurrence of a duplicate $\bar{z}$ within a $\mathcal{T}$ indicates a redundant signature by a given signer.)

As a more concrete example, consider a SGS with a sub-group of size two:

$$< m,\ \bar{g},\ SIG_1,\ SIG_2 > \text{ where:}$$

$$SIG_i =< \bar{z_i}, V_{1,i}, V_{2,i} > \text{ and } \bar{z_i} \neq \bar{z_j} \text{ for } 0 < i \neq j \leq 2$$

10.1 Security Considerations

Assuming, once again, that the underlying CS97 scheme is secure, we claim the following:

Lemma 1. *If $\mathcal{T}$ is a valid SGS by sub-group $S = \{M_1, ..., M_i\}$ on message m, then $\mathcal{T}$ has compositional integrity, i.e., anyone can verify that $\mathcal{T}$ is composed of i group signatures (of message m) each generated by a distinct group member.*

Proof outline: Since $\mathcal{T}$ is a valid SGS, for each constituent SIG_j, $\bar{z_j}$ is unique and all SKROOTLOG-s and SKLOGOLOG-s hold with a common base, $\bar{g}$. Suppose, by contradiction, that there exist two signatures $SIG_u, SIG_w \in \mathcal{T}$ that are actually generated by the same group (and also sub-group) member, M_k. Each group member has only one certificate of the form $(a^x + t)^d$.

For SIG_u, $V_{1,u}$ demonstrates knowledge of x_u where $\bar{z_u} = \bar{g}^{a^{x_u}}$ and $V_{2,u}$ demonstrates knowledge of the corresponding certificate $(a^{x_u} + t)^d$. However, SIG_w is also a valid group signature and $V_{1,w}$, $V_{1,w}$ demonstrate knowledge of x_w ($\bar{z_w} = \bar{g}^{a^{x_w}}$) and $(a^{x_w} + t)^d$, respectively. As M_k only has one certificate, $(a^{x_w} + t)^d = (a^{x_u} + t)^d$ and $x_w = x_u$. This means:
$\bar{g}^{a^{x_w}} = \bar{z_w} = \bar{g}^{a^{x_u}} = \bar{z_u}$ which contradicts our assumption. ∎

Lemma 2. *It is computationally difficult to decide i) whether two different and valid SGS-s $\mathcal{T}_\infty, \mathcal{T}_\in$ are generated by the same sub-group or ii) whether sub-groups that produced $\mathcal{T}_\infty, \mathcal{T}_\in$ have any members in common.*

Proof sketch: Unlinkability of SGS-s, follows directly from the security of CS97. Although the same random base $\bar{g}$ is used in the generation of all single-member group signatures that form an SGS, each signer M_j uses a distinct membership key a^{x_j} in computing its $\bar{z_j} = \bar{g}^{a^{x_j}}$. The sequence:

$$\bar{g} = g^r,\ \bar{z_1} = g^{r \cdot a^{x_1}}, \ldots, \bar{z_i} = g^{r \cdot a^{x_i}}$$

can be viewed as a sequence of ElGamal encryptions of a fixed message "1" where:

- (r, g^r) represent the one-time private and public keys of the signing sub-group
- every $g^{a^{x_j}}$ $(0 < j \leq i)$ is the "public key" of a given signer, M_j and a^{x_j} is the corresponding "private key".
- $\bar{z_j} = g^{r \cdot a^{x_j}}$ is the actual *encryption*.

Since r is unique and random for every SGS, unlinkability is achieved, however, only **outside the sub-group** as discussed in the next section.

10.2 Unlinkability Reconsidered

Having established SGS as a practical construct, we now observe that unlinkability, in the context of SGS, has acquired a new dimension:

> Unlinkability outside the signers' sub-group as opposed to universal unlinkability

The former refers to unlinkability of multiple signatures by **outsiders**, i.e., non-members of the signing sub-group. Whereas, the latter means that SGS-s are unlinkable by anyone with the sole exception of the group manager. To be more specific, consider the following two scenarios:

1. A sub-group $\mathcal{M}$ of like-minded group members (fully aware of each other's identities) decide to sign two documents. No one outside the sub-group (whether group member or not) can link the two SGS-s. However, since members of $\mathcal{M}$ "know" each other, they can implicitly link multiple SGS-s that they produce.
2. All members of a large group vote on two different issues. We consider only the YES votes. The YES-voters on the first issue, form a *de facto* sub-group $\mathcal{M}_1$ and produce SGS_1. Likewise, YES-voters on the second issue, form $\mathcal{M}_2$ and produce SGS_2. Since each member votes independently, the composition of $\mathcal{M}_1$ (and $\mathcal{M}_2$) must be secret even to its own members. A member of $\mathcal{M}_1$ ($\mathcal{M}_2$) needs to know only that s/he signed and the total number of signers. Furthermore, if we let $\mathcal{M}_3 = \mathcal{M}_1 \cap \mathcal{M}_2$, then, clearly it is evident that $0 \leq \#\mathcal{M}_3 \leq min(\#\mathcal{M}_1, \#\mathcal{M}_2)$. However, a group member $M_j \notin \mathcal{M}_1 \cup \mathcal{M}_2$ cannot narrow $\#\mathcal{M}_3$ any further. Also, a group member $M_j \in \mathcal{M}_1 \cup \mathcal{M}_2$ can at most narrow $\#\mathcal{M}_3$ to:

   $$1 \leq \#\mathcal{M}_3 \leq min(\#\mathcal{M}_1, \#\mathcal{M}_2)$$

   Put another way, **a signer cannot be linked across multiple SGS-s even by fellow co-signers.**

It is notable that the sample SGS construct of the previous section only provides unlikability of the first kind, i.e., multiple SGS-s can be linked but only by common co-signers. Recall that in the initial stage of our SGS scheme, signers agree on a common r (generated perhaps as a result of a Diffie-Hellman key exchange). Because r is known to all signers, given:

$$z_j = \bar{g}^{a^{x_j}} = (g^r)^{a^{x_j}}$$

produced by some signer M_j, any other signer M_i can compute: $(z_j)^{r^{-1}} = (g)^{a^{x_j}}$. The latter is constant for a given M_j and can thus be linked with other SGS-s where both M_i and M_j are co-signers.

The above may not be a problem in some practical settings since co-signers might choose to identify themselves to one another prior to signing (for other reasons). On the other hand, it turns out to be easy to change our example SGS scheme to provide universal SGS unlinkability.

We first note that there is no requirement in CS97 for a signer to know r used to compute the one-time random base $\bar{g} = g^r$. This allows us to amend the initial stage of SGS generation so that signers compute a common:

$$\bar{g} = g^r \text{ where } r = r_1 \cdots r_i \text{ and } r_j \in_R \mathbb{Z}_n^* \text{ by } M_j$$

The differences with respect to the first SGS scheme are superficial. The same group Difie-Hellman key agreement can be used, except that, instead of computing a common r, the signers compute a common g^r. (And, r itself is unknown.) However, in the resulting scheme, SGS-s are unlinkable even by common so-signers.

10.3 Synchronous *vs* Asynchronous Operation

Our sample CS97-based SGS scheme assumes synchronous or on-line operation. This is due to the nature of the initial stage: members of the signing sub-group (co-signers) agree on a secret r which is contributed to by each co-signer. Consequently, the scheme fails to address certain SGS scenarios, such as:

> Suppose, once again, that members of a large and distributed group are asked to vote. Instead of voting on-line, they are given a time window (say, a week) to cast their votes.

In this case it is clearly impractical to conduct the initial stage as described above. Ideally, each signer should be able to sign asynchronously, at its own leisure. At the same time, the crucial properties of SGS must be maintained; especially, compositional integrity.

One possible solution is to ask a trusted entity to generate a common random base $\bar{g} = g^r$ for all members who choose to sign a given message. This entity can be the group manager who is already trusted with issuing membership certificates and generating group public and group private keys. As noted in the previous section, signers do not need to know r; they must be assured that it is random and fresh. It seems reasonable to trust the group manager with this task.

More concretely, the group manager generates a one-time random base $\bar{g}$ and commits to it by signing (it) with its own private key which is unrelated to the group private key. Furthermore, the signed $\bar{g}$ must be associated with the specific message to be signed. In the voting example, the group manager would generate two signed statements (where SK_{gm} is the group manager's private key):

1. { YES, m=[text], $\bar{g_1}$, GROUP-ID } SK_{gm}
2. { NO, m=[text], $\bar{g_2}$, GROUP-ID } SK_{gm}

The advantage of this approach is the apparent lack of additional overhead imposed by SGS: the total cost is exactly that of computing multiple CS97 group signatures. The main drawback is, of course, the involvement of a trusted third party; whether the group manager or another trusted entity.

11 Summary

In conclusion, this paper identified some open issues in group signatures. In particular, we concentrated on two issues that, in our opinion, prevent the reduction of group signatures to practice: lack of solid foundation (and proofs) for coalition-resistance and the difficulty of deleting group members and, consequently, the difficulty of re-issuing group and individual member's keys.

In the second part of the paper we presented two new and practical services that can be easily built on top of any group signature scheme: multi-group signatures (MGS) and sub-group signatures (SGS). After analyzing their respective security requirements, we presented sample constructs for both MGS and SGS.

12 Acknowledgements

The authors would like to thank Jan Camenisch as well as the anonymous and unlinkable referees for their helpful comments.

References

1. C. Boyd. Digital Multi-signatures. Cryptography and Coding (H.J.Beker and F.C.Piper Eds.), Oxford University Press, 1989, pp 241-246..
2. M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology – EUROCRYPT*, 1994.
3. J. Camenisch. Efficient and generalized group signatures. In *Advances in Cryptology – EUROCRYPT*, 1997.
4. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology – CRYPTO*, 1997.
5. D. Chaum and T. Pedersen. Wallet databases with observers. In *Advances in Cryptology – CRYPTO*, 1992.
6. D. Chaum and E. van Heyst. Group signatures. In *Advances in Cryptology – EUROCRYPT*, 1991.
7. L. Chen and T. Pedersen. On the efficiency of group signatures providing information-theoretic anonymity. In *Advances in Cryptology – EUROCRYPT*, 1995.
8. A. Lysyanskaya and Z. Ramzan. Group blind digital signatures. In *Financial Cryptography Conference*, 1998.
9. H. Petersen. How to convert any digital signature scheme into a group signature scheme. In *Security Protocols Workshop*, 1997.
10. M. Steiner, G. Tsudik, and M. Waidner. Diffie-hellman key distribution extended to groups. In *ACM Conference on Computer and Communication Security*, pages 31–37, March 1996.