Secure Acknowledgment Aggregation and Multisignatures with Limited Robustness*

Claude Castelluccia[1,2], Stanisław Jarecki[1], Jihye Kim[1], and Gene Tsudik[1]

[1] University of California, Irvine
Computer Science Department, Irvine, CA 92697-3425
{stasio,jihyek,gts}@ics.uci.edu
[2] INRIA Rhône-Alpes, 38334 Saint Ismier CEDEX, France
claude.castelluccia@inrialpes.fr

Abstract. In certain reliable group-oriented and multicast applications, a source needs to securely verify whether all (and if not all, which) intended receivers have received a message. However, secure verification of individual acknowledgments from all receivers can impose a significant computation and communication burden. Such cost can be significantly reduced if intermediate nodes along the distribution tree aggregate the acknowledgments produced by the multicast receivers into a single *multisignature*.

The approach explored in prior work on acknowledgment aggregation [11] is based on a multisignature scheme of [1]. However, this multisignature scheme requires a relatively new cryptographic assumption of "Gap Diffie-Hellman". In contrast, we propose a solution using multisignature schemes secure under more standard and long-standing security assumptions. In particular, we show how to extend previously known non-robust multisignature scheme [9] based on the discrete logarithm assumption to achieve limited robustness. Our extension – which also generalizes to certain other multisignature schemes – allows for efficient multisignature generation in the presence of (possibly malicious) node and communication failures, as long as the number of such faults does not exceed certain threshold.

1 Introduction

Multicast (one-to-many) and group communication is widespread in a variety of settings. Popular examples include IP Multicast, p2p content sharing, digital cable TV transmission, mobile ad hoc networks (MANETs) and application-layer replication protocols. Multicast security has been the subject of much attention in the research literature. Most of the relevant work has been in the context of key management, multicast/broadcast

* An early version of this work was presented at the 2004 Security in Communication Networks (SCN'04) conference [4].

encryption and efficient content authentication. One of the related issues that has not been sufficiently considered is the problem of secure (authenticated) acknowledgments. In certain reliable multicast settings, after sending out a multicast message, the source is interested in establishing whether all (and if not all, which) group members have received the message.

In this paper we propose several new techniques for efficient authentication of acknowledgments generated in response to a multicast message. We are interested in schemes which are efficient, scalable, robust with respect to failures and malicious participants, as well as provably secure under long-standing (standard) cryptographic assumptions such as the difficulty of computing discrete logarithms.

The paper is organized as follows: the rest of this section sets the stage for our work, reviews prior work and summarizes our contribution. The next section describes our main result, a new multisignature scheme based on Discrete Logarithm Assumption which offers a limited robustness and can be used for acknowledgment aggregation. Section 3 contains a security proof for this scheme, together with the discussion about optimality of our robustness bound and an extension that allows for larger exponent sizes. Section 4 considers an optimized variant of the scheme applicable in the symmetric key setting. Section 5 discusses performance and presents an optimization that reduces the number of communication stages to only one. In Appendix A we show how our robust multisignature scheme based on the discrete logarithm problem generalizes to other multisignature schemes, using as an example a scheme based on Guillou-Quisquater (GQ) signatures [6].

1.1 Problem Statement

We assume that packets are sent from the source to group members (intended receivers) along a delivery tree. This tree is rooted at the source and members are represented as leaves and, possibly, also as intermediate nodes. The delivery tree is not necessarily binary, i.e., a node can have more than two children. However, for the sake of simplicity in the presentation, we assume that the group members are leaves of a binary multicast tree rooted at the source.

After multicasting a message M to the group, the source needs to make sure that all members have received it. One simple solution is to ask each member to send an authenticated acknowledgment back to the source. However, this solution is not scalable as it results in the acknowledgment implosion problem, i.e. the individual acknowledgments take up too much bandwidth, which is often a scarce resource. While the computational cost of verifying the individual acknowledgments can be sped up via various batch signature verification techniques, such techniques do not address the need to save the communication resources.

1.2 Prior Work

Nicolosi and Mazieres [11] recently proposed to reduce the computation and the communication costs associated with acknowledgment verification by aggregating multiple acknowledgments using a multisignature scheme of Boldyreva [1]. A multisignature scheme is a generalization of the standard notion of a signature to messages signed by *groups* of users. In a multisignature scheme [10], s is called a multisignature on message M issued by a group of players G if (s, M) passes certain verification equation involving the set of all public keys in group G. If the multisignature scheme is secure, this happens only (except for negligible probability) if *all* players in group G indeed signed M.[3] Efficiency-wise, we would like a multisignature scheme to ensure that both the size of the resulting multisignature as well as the time required to verify it are similar to sizes and times incurred by standard signatures, irrespective of the group size.

It is easy to illustrate multisignatures using the scheme of [1], which is a generalization of a plain signature scheme (BLS) proposed by Boneh et al. [3]. Assuming that an element g is a generator of such a group, in a BLS signature the user's private key is x, the public key is a group element $y = g^x$, the signature on a (hashed) message M is $s = M^x$, and the signature verification consists of checking that (g, y, M, s) is a Diffie-Hellman (DDH) tuple. Boldyreva's multisignature scheme generalizes BLS signatures by defining string s as a multisignature on M issued by a *group* of players G if (g, y, M, s) is a DDH tuple for $y = \prod_{i \in G} y_i$. Note that if each s_i is a BLS signature issued by player i on M, then $s = \prod_{i \in G} s_i$ is a multisignature on M issued by players in G. Both schemes are secure in the Random Oracle Model under the so-called "Gap Diffie-Hellman" (GDH) group assumption, which requires that even if it is easy to *decide* whether a tuple of four group elements (g, y, z, w) is a Diffie-Hellman tuple, i.e. whether $DL_g(y) = DL_z(w)$, still *computing* a DH function $F_{g,y}(z) = (z)^x$ on a random group element z is intractable without the knowledge of $x = Dlog_g y$. The GDH assumption is hypothesized to hold in certain

[3] Thus, multisignatures, referred to as "Accountable Subgroup Multisignatures" by Micali, et al. are a special case of so-called "aggregate signatures" [2] which enable aggregation of signatures by multiple signers on possibly *different* messages.

elliptic curve groups with Weil pairings, where decisional Diffie-Hellman can be efficiently computed via the pairing, but where computational Diffie-Hellman still appears to be hard [7, 5].

Since the aggregation of BLS signatures into a multisignature does not require participation of the signers, this multisignature scheme enables robust aggregation of acknowledgments by the intermediate nodes along a multicast delivery tree: Each intermediate node can verify – given the (combined) public keys of the nodes below it – whether the (aggregated) acknowledgments it receives are correct, and then aggregate them further for the node above. Together with an aggregation of the valid multisignatures he receives, each node also passes up identities of members involved in this multisignature. In this way, the source receives the final multisignature and the identities of members whose signatures it contains. Note that this scheme uses constant bandwidth on every link (modulo the size of the list of identities that increases as the message approaches the root), and that the cost of multisignature verification is the same as the verification of a standard BLS signature. Furthermore, this solution implicitly provides *traceability* by allowing the source to eventually identify any malicious participants who might send bogus acknowledgments.

1.3 Our Contribution

While efficient and robust, the above scheme is based on relatively new GDH cryptographic assumption. In this paper we show that a robust multisignature scheme – and thus a robust acknowledgment aggregation – can be obtained based under more standard cryptographic assumption of the hardness of the discrete logarithm (DL) problem. Our solution is an improvement on the non-robust DL-based multisignature scheme proposed in [9]. Like the scheme in [9], our scheme is a variant of the Schnorr's signature scheme [13], provably secure (in the Random Oracle Model) under the discrete logarithm assumption. However, by tying together the individual players' commitments into Schnorr signatures with the aid of a Merkle hash tree [8], our multisignature scheme has a novel property of *robustness*: Using the new scheme, a group of signers can efficiently generate a multisignature even in the presence of (some number of) communication failures between participating players and/or malicious behavior on the part of (some of) the players. In contrast, the multisignature scheme of [9] would have to be restarted from scratch in the case of a single communication or node fault during a multisignature generation protocol.

Our scheme achieves only *limited robustness* as the number of the communication and/or malicious node faults it can tolerate is bounded by a threshold, which is linear in the bit length of the modulus involved in the underlying discrete logarithm problem. The approximate relationship between the size n of the signing group, the maximal threshold t of faults, and the size $|q|$ of the modulus that determines the size of the exponents in the underlying discrete logarithm problem, is $|q| = \Omega(t * \ln n)$. The exact constraint we impose on tuple (t, n, q) is that $S_{t,n}/q$ is negligible, where $S_{t,n} = \binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{t}$. Since $S_{t,n} \leq t(ne/t)^t$, this constraint holds if, for example, $q \geq t(ne/t)^t * 2^{80}$.

This constraint enables us to achieve full robustness, i.e. $t \leq n$, for small groups of signers, e.g. $n = 256$, using 344-bit exponents. Assuming that standard 160-bit exponentiation modulo a 1024-bit modulus can be done in about $1 - 2$ milliseconds on a modern PC, our signing and verification would then take only $2-4$ milliseconds. For larger groups our scheme can tolerate smaller thresholds at this cost, e.g. $n \approx 10^6$ and $t = 32$ require 597-bit q, but increasing the tolerated threshold size to $t = 10^3$ results in $11K$-bit exponentiation, which makes the scheme significantly less efficient than standard signature schemes, but the signatures can still be computed and verified in about 1/10-th of a second.

Interestingly, the above bound on the robustness threshold of our scheme is *tight*: If the n, t, q parameters do not satisfy it the multisignature scheme becomes efficiently forgeable. This does not endanger our scheme since the check that the parameters are picked correctly is effectively a part of the (multi)signature verification procedure.

1.4 Discussion of Requirements on Multisignature Schemes

We note that our scheme suffers from the same limitation as the multisignature schemes proposed before, including the BLS-signature based multisignature of Boldyreva [1] and the Schnorr-signature based multisignature of Micali et al. [9]. Namely, all these schemes make special requirements on the generation of the participants' public keys. (Identification of these requirements was one of the contributions of [9].)[4]

One possible requirement is that *all* certification authorities (CA-s) that certify the participants' public keys must be honest. In particular, as part of each certification process, each participant must provide a zero-knowledge proof of knowledge of its private key and this proof must be verified by an honest CA. As pointed out in [9], this requirement makes

[4] In contrast, the *aggregate* signature scheme of [2] does not impose these requirements on the public keys, but that scheme only aggregates the bandwidth taken by the aggregate signature, leaving the verification time proportional to the size of the signing group.

delegation problematic, precludes self-delegation completely, and is appropriate only if all certificates are signed by very few completely trusted entities.

An alternative requirement under which all these schemes, including ours, are secure is that all participants generate and certify their public keys as part of a special distributed protocol. While this requirement avoids trusted third parties (i.e., CA-s) completely, it is applicable only to small groups and is thus unsuitable for a general public key infrastructure.

However, while these limitations remain a serious problem for general applications of multisignatures, they do not influence the application of multisignatures to multicast acknowledgment aggregation. In this context, we can safely assume that all participants' keys are certified by a single trusted CA. Moreover, in some applications we can even assume that the CA might have access to everyone's private key. Therefore in the subsequent sections we choose to present our multisignature scheme assuming a single trusted certification authority.

We note that similarly to the BLS-based scheme of [1], but unlike the Schnorr-based scheme of [9], our scheme does not require that all players involved in the multisignature generation protocol take as input the set G of players that participate in this protocol. We believe that the closer analysis of the security proof given for the original scheme of Micali et al. shows that this requirement is not needed in that scheme as well. However, similarly to the scheme of [9] our security analysis requires that the players in our scheme participate in only one instance of this protocol at any given time. However, this requirement can be relaxed and the opening and closing messages from two consecutive instances of this protocol (e.g. for acknowledging two consecutive multicasts) can be piggybacked on one another. We describe this optimization of our protocol in Section 5.

2 DL-based Multisignature Scheme

In this section we construct a multisignature scheme based on a prior extension of the Schnorr signature scheme [13]. Before describing the scheme, we briefly mention certain environmental features and parameters.

2.1 Computational Setting and Initialization

As expected in the original Schnorr signature scheme [13], we assume common parameters (p, q, g) where p, q are large primes (q is a large divisor of $p-1$) and g is an element of order q in $\mathbb{Z}_p^*$. As in the Schnorr signature scheme we assume a hash function $h : \{0,1\}^* \rightarrow \mathbb{Z}_q$, which we model as a random oracle. All equations involving multiplication or exponentiation are meant modulo p.

As mentioned in the introduction, we assume a single trusted CA that signs all participants' public keys. We describe our multisignature scheme using the specific application of acknowledgment aggregation as the underlying context. Namely, we assume that the group of players who are potential participants in the multisignature generation are multicast group members and that they are logically organized in a binary tree, with group members represented as leaves. The intermediate tree nodes are represented by the multicast delivery network and the source S is represented as the root. For simplicity, we will assume that the tree is complete, i.e. that the number of leaves is $n = 2^d$ for some d. We note, however, that the scheme is equally applicable to non-binary and incomplete trees as well as trees where intermediate nodes are themselves receivers or group members.[5]

We denote the left and right children of S as N_0 and N_1. More generally, the left and right children of N_i are defined as N_{i0} and N_{i1} (see Figure 1 for example). Each member N_i randomly selects its secret key $x_i \in [0, q-1]$ and sets its public key $y_i = g^{x_i}$. As discussed in the introduction, under the assumption of a single trusted CA, the proof of security requires that during the registration of the public key y_i a player must pass a zero knowledge proof of possession of the discrete logarithm $x_i = DL_g(y_i)$.[6] When our scheme is used for efficient acknowledgment aggregation, the trusted source can either check each player's ZK proof, or, to support the "optimistic" mode of the protocol operation, the source simply picks N_i's secret x_i himself and shares it with the player. (We describe this optimization in Section 4.)

We assume that each node N_i knows the public keys of all members (tree leaves) in the subtree rooted at N_i. Each node can also aggregate the keys of all the members in his subtree. The aggregated public key y_i is computed as $y_i = y_{i0} * y_{i1}$, where y_{i0}, y_{i1} are (possibly aggregated) public keys of N_i's children. Using this notation, $y = y_\epsilon$ is a product of the public keys of all group members $y = \prod_{i \in G} y_i$.

[5] In a more general case, the tree structure needs to be computed jointly by the participating players, and both the intermediate nodes as well as the source would involve special functions executed by the players with appropriate roles.

[6] If no trusted CA's can be assumed, to assuage the problem of concurrent composition of such proofs, our multisignature scheme would have to generate all public keys simultaneously, in a distributed protocol proposed by [9].

2.2 Scheme Overview

In the Schnorr signature scheme, a signature on message M under key $y = g^x$ is generated by producing a one-time "commitment" $r = g^v$ for a random $v \in [0, q-1]$, computing a "challenge" $c = h(m, r)$, and issuing a "response" $z = v + c * x \bmod q$. The signature is a pair (r, z) s.t. $g^z = r * y^c \bmod p$ and $c = h(m, r)$. Micali, et al. aggregate such signatures, i.e. pairs (r_i, z_i) produced by members of some group G, by running a 3-round protocol: (1) everyone *broadcasts* its commitment $r_i = g^{v_i}$, (2) everyone combines $r = \prod_{i \in G} r_i$ and computes the challenge $c = h(m, r)$, and, (3) everyone broadcasts their responses $z_i = v_i + c * x_i \bmod q$. It's easy to see that (c, z), where $z = \sum_{i \in G} z_i$, is a "Schnorr multisignature" for the group G, with $y = \prod_{i \in G} y_i$ as the verification key.[7]

However, this scheme is not robust against node and link failures during the computation of the multisignature. For example, if a node first sends a commitment r_i but fails to send a valid response z_i, the multisignature has to be recomputed from scratch. To alleviate this problem, instead of hashing a simple product of all r_i's as above, we compute the challenge c via a Merkle-tree [8] aggregation of the r_i values. Namely, the r_i values of each two neighbouring leaf nodes are hashed together by their parents, these hashes are then hashed together by their parents, and so on until the root hashes the values passed to it by its two children into the final c.[8] Since a Merkle Tree is a commitment to all the r_i's, the resulting challenge c is meaningful to all subsets of r_i's that were used to create it, and hence it can be used for a multisignature involving those (and only those) players that respond to c with a proper response z_i's. We note that the Merkle tree we construct is not standard because we fold into the intermediate values r_i accumulated so far (see Figure 1), which allows for a more efficient handling of potential faults occurring later in the protocol.

[7] In the extended version [10], the authors show that the same scheme also works without broadcast, e.g., if players communicate in a ring-like fashion. That version improves communication costs but is also not robust.

[8] We note that Micali, et al. use a Merkle tree in the key generation protocol, but they use it to enable provable security in the absence of a trusted CA, whereas we use it in the multisignature generation protocol to achieve robustness. The two concerns are independent, and in particular the Micali et al. technique of key generation can be adopted to our scheme if trusted CAs cannot be assumed. (See the discussion in section 1.4.)

2.3 Multisignature Generation

The multisignature generation protocol has 4 **stages**. We use the term *stage*, as opposed to *round*, to take into account the structure of the delivery tree. Thus, each stage, whether upward- or downward-bound, takes $\log n$ rounds.

Each player is assumed to store all information passing through it, but can discard this information once the protocol ends. We assume that the players participate in one instance of this protocol at a time. (We will relax this assumption slightly in Section 5.)

Stage 0: This is the stage of the actual message delivery, i.e., source sends a message M along the delivery tree.

Stage 1: Each member N_i that receives M randomly selects $v_i \in [0, q-1]$ and sends to its parent the commitment $r_i = g^{v_i}$ and the *partial* challenge $c_i = h(r_i)$. A node N_j that receives two commitments and partial challenges $\{r_{j0}, c_{j0}\}$ and $\{r_{j1}, c_{j1}\}$ from its two children, N_{j0} and N_{j1}, stores these values, generates its own commitment and partial challenge $r_j = r_{j0} * r_{j1}$ and $c_j = h(r_{j0}, r_{j1}, c_{j0}, c_{j1})$. It then forwards $\{r_j, c_j\}$ to its parent, as illustrated in Figure 1. Each N_i also passes up the identities of nodes in N_i's subtree which participated in the protocol. Alternatively, if all group members participate in the protocol by default, the intermediary nodes only pass to each other identities of nodes that *did not* participate. If some node N_j on the tree does not send correct values to its parent, the parent assigns $r_j = 1$ and $c_j = 0$.

Stage 2: When the source receives the two tuples $\{r_0, c_0\}$ and $\{r_1, c_1\}$ from its two children N_0 and N_1, it computes $r = r_0 * r_1$ and the final challenge $c = h(M, G, r_0, r_1, c_0, c_1)$. The source then sends back the set G and values (c, r_1, c_1) to N_0, and G and (c, r_0, c_0) to N_1. N_0 then sends G and $(c, r_1, c_1, r_{01}, c_{01})$ to N_{00} and G and $(c, r_1, c_1, r_{00}, c_{00})$ to N_{01}, and so on. Figure 2 shows an example of how the challenge c is propagated from the source to the members.

The point of this process is that each node N_i, including the intermediary nodes, receives the challenge c together with the values lying on the so-called "co-path" of node N_i in the Merkle hash tree, which in our case are values (r_j, c_j), which are necessary for node N_i to (re)compute the sequence of hash functions that lead from its own values (r_i, c_i) to the challenge c. More formally, we will denote such set as $\mathsf{MHPath_i} = \{(\mathsf{r_j}, \mathsf{c_j})\}_{\mathsf{j} \in \mathsf{copath_i}}$, where $\mathsf{copath_i} = \{\mathsf{i}\} \cup \{\mathsf{i_{[0,k]}\bar{i}_{k+1}}\}_{\mathsf{k}=0,\ldots,|\mathsf{i}|-1}$, where $i_{[0,k]}$ denotes the k-bit prefix of the bit string i, including the empty string i_0, $\bar{i}_{k+1}$

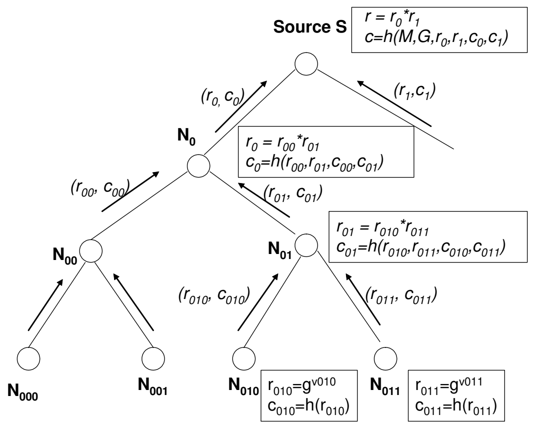

Fig. 1. Computation of the Merkle Tree

denotes the complement of the $k+1$-st bit of i, and "$i_{[0,k]}\bar{i}_{k+1}$" denotes their concatenation. For example, $\mathsf{copath}_{011} = \{011, 010, 00, 1\}$. Using the $\mathsf{MHPath_i}$ information, each N_i can recompute $c_j = h(r_{j0}, r_{j1}, c_{j0}, c_{j1})$ and $r_j = r_{j0} * r_{j1}$ consecutively, for each proper non-empty prefix j of i, and finally verify that $c = h(M, G, r_0, r_1, c_0, c_1)$. We denote this operation as checking if $c = h_{\mathsf{MHT}}\ (M, G, \mathsf{MHPath_i})$. For example, for N_{011} the set of co-path values is $\mathsf{MHPath}_{011} = \{(\mathsf{r}_{011}, \mathsf{c}_{011}), (\mathsf{r}_{010}, \mathsf{c}_{010}), (\mathsf{r}_{00}, \mathsf{c}_{00}), (\mathsf{r}_1, \mathsf{c}_1)\}$, and the verification if $c = h_{\mathsf{MHT}}(M, G, \mathsf{MHPath}_{011})$ consists of recomputing $r_{01} = r_{010} * r_{011}$ and $c_{01} = h(r_{010}, r_{011}, c_{010}, c_{011})$, $r_0 = r_{00} * r_{01}$ and $c_0 = h(r_{00}, r_{01}, c_{00}, c_{01})$, and checking if $c = h(M, G, r_0, r_1, c_0, c_1)$. In fact, since the intermediary nodes N_i know the (r_{i0}, c_{i0}) and (r_{i1}, c_{i1}) values of their children, the intermediary nodes know sets $\mathsf{MHPath_{i0}}$ and $\mathsf{MHPath_{i1}}$ in addition to $\mathsf{MHPath_i}$. They will use this knowledge in the next stage in case a child sends a faulty response then.

Stage 3: If the challenge c verifies, each signer N_i sends back its response $z_i = v_i + c * x_i \bmod q$. Importantly, N_i needs to then either erase the private value v_i used here or mark as used and never re-use it on a different c. An intermediate node N_j that receives values z_{j0} and z_{j1} from its two children verifies each of them by checking that $g^{z_{j0}} = r_{j0} * (y_{j0})^c$ and $g^{z_{j1}} = r_{j1} * (y_{j1})^c$. If the equations verify, N_j forwards to its parent the aggregated value $z_j = z_{j0} + z_{j1} \bmod q$, and so on until the aggregated

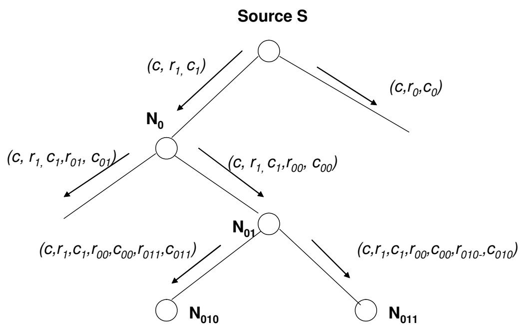

Fig. 2. Transmission of the challenge value c

$z = z_0 + z_1 \bmod q$ value reaches the source. Note that in the absence of faults we have $g^z = ry^c$ because $g^{z_i} = r_i y_i^c$ for every $i \in G$, as illustrated in Figure 3.

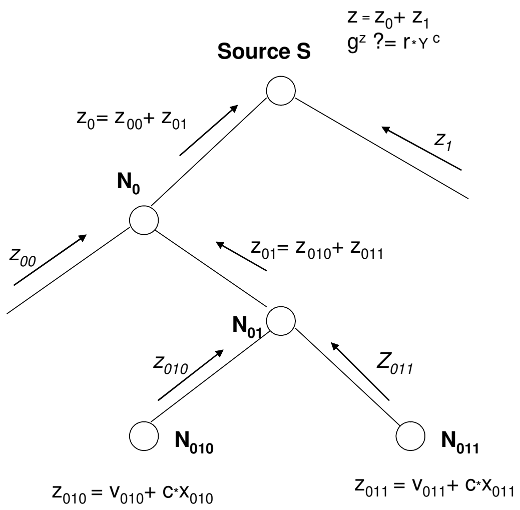

Fig. 3. Default propagation of responses z_i

If one of the signatures is incorrect (let's say z_{j1}), N_j sets z_j to z_{j0} instead of $z_{j0} + z_{j1} \bmod q$, and sends to its parent the value z_j together with two sets, $\mathcal{M}_j = \{j1\}$ and $\mathsf{MHSet_j} = \{\mathsf{MHPath_{j1}}\}$, containing the information about the faulty node N_{j1} in N_j's subtree. Note that N_j's parent knows some part of the $\mathsf{MHPath_{j1}}$ information, so in practice not all this information needs to be sent, but for simplicity of description we will denote it as $\mathsf{MHPath_{j1}}$. The parent, let's say N_k such that $j = k1$, performs two checks: (1) N_k checks if $g^{z_j} = r_j/r_{j1} * (y_j/y_{j1})^c$; and (2) N_k checks if the information about the faulty node is correct by checking that $c = h_{\mathsf{MHT}}(M, G, \mathsf{MHPath_{j1}})$.

In general, each intermediate node N_j passes up a response z_j along with a set $\mathcal{M}_j$ of indices of faulty nodes in N_j's subtree, i.e. nodes whose z_i values were not delivered (possibly in accumulated form) to N_j, as well as a set $\mathsf{MHSet_j} = \{\mathsf{MHPath_i}\}_{\mathsf{i} \in \mathcal{M}_\mathsf{j}}$ of values containing the r_i commitments that correspond to the missing z_i challenges together with the information needed to verify that these supplied r_i commitments are correct. Each node N_k upon receiving such messages from its children first performs the following tests, for its two branches $b = 0$ and $b = 1$:

1. N_k sets $r'_{kb} = r_{kb}/(\prod_{i \in \mathcal{M}_{kb}} r_i)$ and $y'_{kb} = y_{kb}/(\prod_{i \in \mathcal{M}_{kb}} y_i)$ and checks if $g^{z_{kb}} = r'_{kb} * (y'_{kb})^c$
2. N_k checks if $c = h_{\mathsf{MHT}}(M, G, \mathsf{MHPath_i})$ for each $i \in \mathcal{M}_{kb}$

If everything verifies, N_k passes up $z_k = z_{k0} + z_{k1} \bmod q$, $\mathcal{M}_k = \mathcal{M}_{k0} \cup \mathcal{M}_{k1}$, and $\mathsf{MHSet_k} = \mathsf{MHSet_{k0}} \cup \mathsf{MHSet_{k1}}$. In case of a failure in branch b, N_k passes up only the correct values, i.e. $z_k = z_{k\bar{b}}$, and passes up the new set of the missing values as $\mathcal{M}_k = \mathcal{M}_{k\bar{b}} \cup \{kb\}$ and $\mathsf{MHSet_k} = \mathsf{MHSet_{k\bar{b}}} \cup \{\mathsf{MHPath_{kb}}\}$. If both branches fail, N_k passes up just $\mathcal{M}_k = \{k\}$, and implicitly $\mathsf{MHSet_k}$ is set as $\mathsf{MHSet_k} = \{\mathsf{MHPath_k}\}$ by N_k's parent.

Figure 4 illustrates this step when one of the member's signature is incorrect. For lack of space on the picture, we denote there a simplified version of the verification of aggregated signatures performed by each intermediary node. In our example, N_{01} detects that the signature generated by N_{011} is incorrect because $g^{z_{011}} \neq r_{011} * y^c_{011}$. N_{01} then sets z_{01} to z_{010} and forwards the message $\mathcal{M}_{01} = \{011\}$ and $\mathsf{MHSet_{01}} = \{\mathsf{MHPath_{011}}\}$ to its parent N_0. N_0 then checks if $c = h_{\mathsf{MHT}}(G, M, \mathsf{MHPath_{011}})$ and verifies the signature aggregated so far by checking whether $g^{z_{01}} = r_{01}/r_{011} * (y_{01}/y_{011})^c$ and $g^{z_{00}} = r_{00} * (y_{00})^c$, which on figure 4 we summarily denote as verifying if $g^{z_0} = r_0/r_{011} * (y_0/y_{011})^c$. The N_0 node then forwards $z_0 = z_{00} + z_{01} \bmod q$, $\mathcal{M}_0 = \mathcal{M}_{01}$ and $\mathsf{MHSet_0} = \mathsf{MHSet_{01}}$ to the root, who does the final check and aggregation.

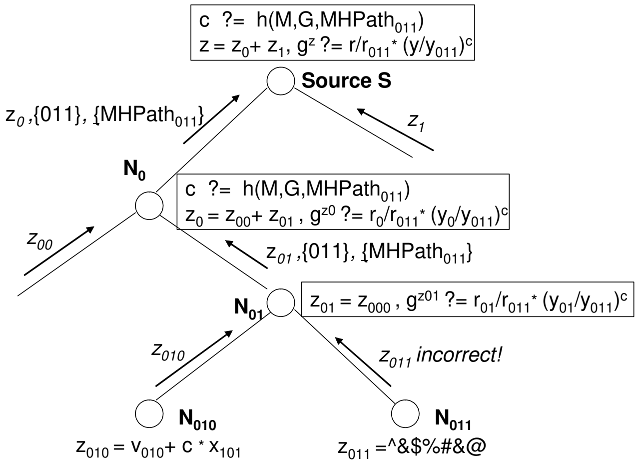

Fig. 4. Propagation of responses z_i in case of a fault at N_{011}

2.4 Multisignature Verification

The final aggregation and verification performed by the root can be seen as a verification of the following *multisignature*:

$$\sigma = [z, (r_0, r_1, c_0, c_1), \mathcal{M}, \{\mathsf{MHPath_i}\}_{i \in \mathcal{M}}]$$

which is defined as a correct multisignature issued on message M by players in $G \setminus \mathcal{M}$ as long as:

$$g^z = \left(\frac{r}{\prod_{i \in \mathcal{M}} r_i} \right) * \left(\prod_{i \in G \setminus \mathcal{M}} y_i \right)^c$$

where

$$c = h(M, G, r_0, r_1, c_0, c_1) \;\; \text{and} \;\; r = r_0 * r_1$$

and moreover:

1. $c = h_{\mathsf{MHT}}(M, G, \mathsf{MHPath_i})$ for each $i \in \mathcal{M}$

2. Values $n = |G|$, the number t of *individual participants* (implicitly) specified by the missing set $\mathcal{M}$, and the modulus q defining the size of the exponents, satisfy the constraint that $S_{t,n}/q$ is negligible, e.g. less than 2^{-80}, where $S_{t,n} = \binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{t}$.

Importantly, the criterion in point 2 above limits the number of the missing *individual participants* represented by the $\mathcal{M}$ set, and not just the *size* of that set, i.e. the number of indices in set $\mathcal{M}$, because set $\mathcal{M}$ in effect represents *subtrees* of missing participants.

3 Security Analysis

We briefly recall the definition of security for a multisignature scheme given by Micali, et al. [9]. The adversary A can corrupt any set of group members, and he conducts a *chosen message and subgroup* attack, i.e. he specifies the message M and the subgroup of players G which participate in the multisignature generation protocol, and then participates, on behalf of the corrupted group members, in the multisignature generation protocol involving the specified group and message.

Definition 1. ([9]) *We say that a multisignature scheme is secure if every efficient adversary A which stages a chosen message and subgroup attack against the multisignature scheme has at best negligible chance of outputting triple (M, G, σ) s.t. (1) σ is a valid multisignature on M issued by the group G, and (2) there exists an uncorrupted player $N_{i^*} \in G$ who has never been asked by A to participate in a multisignature protocol involving group G on message M.*

Theorem 1. *The multisignature scheme described in Section 2 is secure in the Random Oracle Model under the Discrete Logarithm assumption.*

Proof. The proof goes by exhibiting a simulator S which, with sizable probability, converts a successful attack algorithm A against our new multisignature scheme into an algorithm that computes discrete logarithms. The simulation of this scheme is very similar to the simulation of the Schnorr signature scheme, although it is less efficient, and hence the exact security of our scheme is not optimal. However, a similar degradation, although for a different reason, is suffered by the exact security of the multisignature scheme of [9]. The simulator's goal is to compute, on input a random y in $\mathbb{Z}_p^*$ a discrete logarithm $x = DL_g(y)$. Without loss of generality we can assume that the adversary forges a multisignature issued by players $G = \{1, \ldots, n\}$, all of whose members are corrupted except of player N_n, on some message M which N_n is never ask to sign. (This assumption does not hold if the adversary is adaptive, but the same proof holds there too, except that the simulator has to guess the identity of an uncorrupted player against whom the forgery claim is made.) The simulator assigns $y_n = y$ as the public key of N_n, while it picks the private keys x_i of *all the other players* at random.

Since S knows the private data of all uncorrupted players except for N_n, the only thing that the simulator needs to simulate are N_n's responses. This is done similarly as in the Pointcheval and Stern's proof [12] of security of Schnorr signatures, except that as in the security proof [10] of the Schnorr-based multisignature scheme of Micali, et al., the simulator needs to rewind the adversary in this simulation. Namely, when N_n is asked to participate in the multisignature generation on message M, S picks c and z_n at random in $\mathbb{Z}_q$, outputs value $r_n = g^{z_n} * y_n^c$, and then embeds c in the answer to *one* of the A's queries $(M, G, r_0, r_1, c_0, c_1)$ to the h oracle made after S issued its commitment r_n. (Note that thanks to the Merkle hash tree the adversary has only a negligible chance of embedding r_n into any values queried to h before r_n is issued.) If this is not the c that comes down to N_n in the second stage of the protocol together with some correct co-path values $\mathsf{MHPath_n}$ such that $c = h_{\mathsf{MHT}}(M, G, \mathsf{MHPath_n})$, then S cannot proceed and the simulation has to wind back to right after N_n outputs his commitment r_n. (Note that the Merkle Tree hashing does not help us here in any obvious way because the adversary can still try any number of values $r_1, \ldots, r_{n-1}$ he likes, form them together with r_n into many different Merkle Tree hash constructions, and pick any of the resulting c values. For the same reason, however, it does not change anything if the adversary makes the choice of group G, s.t. $N_n \in G$ during this hash query, and that's why player N_n can be told the identity of group G in state 2 instead of stage 0.) If q_h is the maximal number of hash queries made by A, this trial and error procedure eventually succeeds in expected number of at most q_h repeats, which slows the Schnorr-like simulation of this signature process by only a polynomial factor. (We crucially use here the assumption that the players do not participate in two multisignature protocol instances at the same time.) When S is finally lucky and the right c comes down to N_n, the simulator outputs its prepared answer z_n.

Thus the simulation proceeds slowly but surely, and A eventually creates a valid multisignature involving N_n with non-negligible probability $\epsilon = \Omega(1/poly(k))$ for k a security parameter. We will show that an argument similar to the "forking lemma" argument of Pointcheval-Stern [12] applies in this case, and that in polynomial time, with constant probability, such attack algorithm A can be used to produce a polynomial number of related multisignature forgeries. The original proof of Pointcheval-Stern given for security of Schnorr signatures requires extraction of only two related forgeries from A. However, extraction of polynomial number of such forgeries is not much different. Pointcheval-Stern argument applies to any signature scheme whose form is (σ_1, c, σ_2) where σ_1 is a commitment, $c = h(M, \sigma_1)$ is the challenge, σ_2 is some deterministic function of message M and (σ_1, c), and σ_1 is chosen from a sufficiently large space. In the case of our multisignature, thanks to the properties of the Merkle hash tree in the random oracle model, when c is computed as a top of the Merkle hash tree of all the r_i values, all these r_i values are committed. Therefore when $(M, G, r_0, r_1, c_0, c_1)$ is hashed into c, there is a fixed set of values $\{r_i\}_{i \in G}$ which are committed by this hash. Therefore, in our case $\sigma_1 = (G, \{r_i\}_{i \in G})$, c can be thought of as a hash of (M, σ_1), and $\sigma_2 = (z, \mathcal{M}, \{\mathsf{MHPath_i}\}_{i \in \mathcal{M}})$, but we can think of σ_2 as only $(z, \mathcal{M})$, since the $\mathsf{MHPath_i}$ values are just the appropriate elements of the Merkle hash tree supplied for each $i \in \mathcal{M}$.

As in the Pointcheval-Stern argument, see esp. Lemma 8 in [12], it remains true here that, with a constant probability, a polynomial number of re-runs of A produces a pair $(\hat{M}, \hat{\sigma}_1)$ (at some point this pair must be queried, in our case via a Merkle tree of hashes, into the hash function h), s.t. a *random* execution of A on the same fixed set of random coins but on a random oracle h which acts the same until the $(\hat{M}, \hat{\sigma}_1)$ "query", but uses fresh randomness from then on including for its answer $c = h(\hat{M}, G, r_0, r_1, c_0, c_1)$ for (G, r_0, r_1, c_0, c_1) specified by $\hat{\sigma}_1$, produces a *related* forgery $(\hat{M}, \hat{\sigma}_1, c, \sigma_2)$, involving the same pair $(\hat{M}, \hat{\sigma}_1)$, with probability $\epsilon' = \Theta(\epsilon / q_h)$ where q_h is the number of A's queries to h, and hence polynomial in k.

Therefore, since, as we will see below, our simulator will need $n + 2$ such related forgeries to compute its discrete logarithm challenge, if S independently re-runs (in parallel) $n + 2$ instances of the adversary A for $\Theta(\log n / \epsilon')$ times each, where each instance of A executes on the same random string and interacts with hash function h which acts the same until A produces the (G, r_0, r_1, c_0, c_1) query, but which then uses independent randomness in this and subsequent answers, then, with a constant probability, all of these $n + 2$ parallel experiments produce related forgeries $(\hat{M}, \hat{\sigma}_1, c^{(j)}, \sigma_2^{(j)})$, for $j = 1, ..., n + 2$. Note that the probability that since the probability that one instance of an interaction of A with such h produces the above forgery is ϵ', the probability that t such experiments fail to produce a forgery is $e^{-t\epsilon'}$ for small values of ϵ'. Therefore the probability that all $\Theta(n)$ such experiments succeed when executed in parallel t times each is $(1-e^{-t\epsilon'})^n$, which is constant for $t = \Theta(\log n/\epsilon')$. Moreover, since n is polynomial in k, and each execution runs an independent copy of h, the probability that any of these related forgeries share the same challenge c, i.e. that $c^{(i)} = c^{(j)}$ for some $i \neq j$, is negligible.

Thus, with constant probability, in polynomial time S has $n+2$ related forgeries $(\hat{M}, \hat{\sigma}_1, c^{(j)}, \sigma_2^{(j)})$, s.t. each $\sigma_2^{(j)}$ specifies value $z^{(j)}$ and sets $\mathcal{M}^{(j)}$, and $\{MHPath_i\}_{i\in\mathcal{M}^{(j)}}$ s.t.:

1. The number of individual participants implicitly specified by each set $\mathcal{M}^{(j)}$ is smaller than t s.t. $S_{(t,n)}/q < 2^{-80}$. (For simplicity, we use $\mathcal{M}^{(j)}$ here to describe this set of participants; Note that then $\mathcal{M}^{(j)} \subseteq G$ and $n \notin \mathcal{M}^{(j)}$.)
2. As we argued before, in the random oracle model, the Merkle hash tree construction guarantees that all the values r_i specified by sets $\{MHPath_i\}_{i\in\mathcal{M}^{(j)}}$ for every j must, except for negligible probability, belong to the set of n values $r_1, ..., r_n$. A violation in this condition would imply that the adversary A finds a collision in the Merkle hash tree. (Note that each instances of A executes against a hash function which answers all queries in the same way until the $(M, G, r_0, r_1, c_0, c_1)$ query.)
3. Each $z^{(j)}$ satisfies $g^{z^{(j)}} = r/r_{\mathcal{M}^{(j)}} * (y_n\bar{y}/y_{\mathcal{M}^{(j)}})^{c^{(j)}}$, where $r = r_0 * r_1$, $r_{\mathcal{M}^{(j)}} = \prod_{i\in\mathcal{M}^{(j)}} r_i$, $\bar{y} = \prod_{i\in G\setminus\{n\}} y_i$, and $y_{\mathcal{M}^{(j)}} = \prod_{i\in\mathcal{M}^{(j)}} y_i$

happen only with negligible probability in ROM.

Let's denote $v = DL(r)$ where $r = r_0 * r_1$, $v_i = DL(r_i)$ for $i = 1, .., n$, $x_n = DL(y_n)$, and $\bar{x} = \sum_{i\in G\setminus\{n\}} x_i$. Then the condition in item (3) translates into a linear equation on $n+2$ unknowns $v, v_1, \ldots, v_n, x_n$, for each $j = 1, ..., n+2$:

$$z^{(j)} = v - \sum_{i\in\mathcal{M}^{(j)}} v_i + c^{(j)}(x_n + \bar{x} - \sum_{i\in\mathcal{M}^{(j)}} x_i) \bmod q \qquad (1)$$

It remains for us to argue that, except for negligible probability, the $n+2$ equations of type (1) created by this interaction of A and S are linearly independent, and hence, except for negligible probability, we can solve them for x_n and thus answer the DLP challenge. This is because for every choice of membership in the set $\mathcal{M}^{(j)}$, there is only at most one value $c^{(j)}$ which makes the j-th equation linearly dependent on the previous $j-1$ equations. Therefore, the number of $c^{(j)}$ values which can possibly make the new equation dependent on the previous ones is at most $S_{t,n}$, the number of possible t-element subsets $\mathcal{M}^{(j)}$ of G. Since $c^{(j)}$

is always chosen at random, if $S_{t,n} \ll q$ and n is polynomial in the security parameter then the probability that any of the $n+2$ equations is linearly dependent on the previous ones is negligible.

The necessity of running A for $O(q_H n \log n/\epsilon)$ times creates a polynomial factor blow-up in the running time of the simulation. However, it is worse only by the $O(n \log n)$ factor then the blow-up encountered in the security argument for the regular Schnorr signature scheme using the Pointcheval-Stern argument.

3.1 Optimality of the Robustness Bound

Interestingly, the robustness bound on t given by the constraint that $S_{t,n} \ll q$, is essential, and our multisignature scheme can be efficiently forged if $t = n$ and $S_{t,n} = 2^n > 2q$. First, note that the adversary can foil the above proof if he can force any $n+2$ equations (1) to be linearly dependent, which is in fact easy if $|q| \leq n-1$, and the choices $b_i \in \{0,1\}$, for $i = 1,..,n-1$, of whether or not to include index i in set $\mathcal{M}$, form a binary representation of the challenge c. Moreover, this strategy of foiling the reduction extends to an efficient forgery of a multisignature that can be performed by $n-1$ corrupted players against the honest n-th player. If the r_i values for $i = 1,..,n$ are chosen as $r_i = y_n^{\gamma_i} g^{\beta_i}$ s.t. each β_i is random in $\mathbb{Z}_q$ but $\gamma_i = 2^{i-1}$ for $i = 1,..,n-1$ and $\gamma_n = 0$, then by setting bits b_i, for $i = 1,..,n_1$ so that $[b_{n-1}..b_1]$ forms a binary representation of $c < 2^{n-1}$ (from MSB to LSB), and bit $b_n = 1$, the adversary can hash these r_i values in a Merkle hash tree and respond to any resulting challenge $c \in \mathbb{Z}_q$ with the set $\mathcal{M}$ of all players i s.t. i-th bit of c $(= b_i)$ is 0, and z is computed as

$$z = \sum_{i=1}^{n-1} x_i b_i + \sum_{i=1}^{n} \beta_i b_i \bmod q$$

The resulting tuple $(\mathcal{M}, z)$ always satisfies equation (1), and the forgery always succeeds as long as the set t of the missing players can be as large as n and $n \geq |q| + 1$.

3.2 Accommodating Larger Moduli q

The constraint $S_{t,n} \ll q$ can be satisfied for larger values of n and t only by growing the size of q, $|q| \approx (t * \ln n)$. These values grow quite large for larger n, t, for example for $n = 10^6$ and $t = 10^3$ we need $|q| > 11,000$. If p is a prime then p must be larger than q which means that all multiplications and exponentiations are done modulo a 11K-bit modulus, which would make the scheme impractical. However, in a trusted CA setting we adopt in this paper, a CA can pick a 1024-bit RSA modulus n, an element g which is a generator of a large-order subgroup (e.g. the squares modulo n) of $\mathbb{Z}_n^*$, and a parameter $q = 2^k$ for any integer $k > 160$. Our scheme would act in the same way in this setting, except that all multiplications and exponentiations would be done modulo n, secret keys x_i could be chosen in $\{0,1\}^{k'}$ for $k' \geq 512$, hash function h which produces challenge c would be a k-bit string, the v_i's would be chosen in $\{0,1\}^{k''}$ where $k'' = k + k' + 80$, and all operations on these exponents, i.e. computing responses $z_i = v_i + c * x_i$ and adding these z_i's during the aggregation, would be done *over the integers*. Since no one, including the combined powers of the adversary and the simulator, can compute the modulus $\phi(n)$ that acts on these exponents, all equations (1) would remain invertible as long as the constraint $S_{t,n} \ll 2^k$ holds. Also, the $k'' = k + k' + 80$ size of the exponents v_i allows for statistically indistinguishable simulation of the signatures on behalf of the uncorrupted player N_n. Since the responses z_i will be k''-bit long, their accumulation will lead to z's of at most $k'' + \log n$ size. Therefore the bandwidth in such scheme will still be linear in k, and the signing, verifying, and aggregating costs will involve $(k + 762 + \log n)$-bit exponentiations modulo a 1024-bit long modulus.

4 A Multi-MAC Variant

If acknowledgment non-repudiation is not required, the source can share a unique secret key with each intended receiver. In this case, the aggregation scheme can be more appropriately called "multi-MAC" rather than "multisignature". Moreover, while the basic scheme described above requires four stages, the multi-MAC variant can run in an "optimistic" fashion, which requires only two stages if no intermediate node acts maliciously.

In this variant, each member has a unique key z_i shared with the source. We assume that each such key is agreed upon or distributed whenever the member joins the group. Knowing all such keys, the source can add them all up and obtain the aggregated key for any group G of players, $x_G = \sum_{i \in G} x_i$. When a member N_i receives a message M from the source, it replies by sending the acknowledgment $ack_i = m^{x_i}$, where $m = h(M)$, to its parent N_k, which, in turn, multiplies the acknowledgments of its children and sends the resulting aggregated message $ack_i = ack_{i0} * ack_{i1}$ to its parent. The parent also passes up the identities of players that participated in the acknowledgment in his subtree. If most members usually do participate, the parent can instead attach a vector identifying all subtree members who do not participate. When the source computes the final aggregated acknowledgment $ack = ack_0 * ack_1$ and combines the sets of participating members into one set G, it can verify if all these members indeed acknowledge receipt of M by checking whether $ack = h(M)^{x_G}$.

This protocol is robust only against *non-malicious* communication faults. Also, to save memory, the source can pick all the members' keys as $x_k = h(s, k)$ where s is the source's short secret. In this way, the source would not have to store all secrets keys as it can easily compute them on-the-fly.

The optimization allows the source to verify the aggregated acknowledgment in two stages, however, it is not robust against malicious faults, since, if the aggregated acknowledgment is invalid, the source is unable to identify the malicious member(s). We therefore suggest to combine the two schemes by piggybacking the commitment of the basic scheme with the authenticators of the second scheme. As a result, the source can verify the aggregated acknowledgment in two stages. If the verification fails, *Stage 2* and *Stage 3* of the basic scheme can be executed to trace the malicious nodes and robustly compute the desired multisignature.

5 Communication Costs

The exact communication overhead introduced by our scheme depends on the underlying communication medium. If players communicate over a ring as in [10], the communication cost grows from $O(n)$ to $O(tn \log n)$, where t is the number of faults. If players communicate via reliable broadcast, as in [9], then the communication cost does not change. Whereas, if they communicate via a multicast tree, as in the case of multicast acknowledgment aggregation, the total communication cost is $O(n + t(\log n)^2)$.

In the context of multicast acknowledgment aggregation, the comparison of our scheme to that of Nicolosi and Mazieres [11] is as follows: Assuming that the source shares symmetric keys with the receivers, if no *malicious* node faults occur then our scheme can run in an "optimistic" mode which provides an all-or-nothing verification of aggregated acknowledgments and matches the communication cost of the scheme of Nicolosi and Mazieres, i.e. it takes two communication stages and $O(n)$ total bandwidth, where n is the size of the multicast group. Moreover, our scheme has a smaller *computational* overhead since we avoid expensive pairing operations used in [11].

In case of malicious node faults, our robustness mechanisms kicks in. This increases the cost to four stages and total bandwidth grows to $O(n+t(\log n)^2)$, t being the number of faults. In comparison, [11] takes only two stages and total bandwidth remains $O(n)$. Our scheme is therefore applicable when the number of malicious faults and link failures is low, which we believe to be the case for many applications.

However, we suggest a trivial optimization that, while still requiring $O(n + t(logn)^2)$ bandwidth, allows our scheme to run in two stages thus matching the round complexity of [11]. This optimization is based on the assumption that, over time, the source sends multiple messages and the delivery tree is relatively stable. The main idea is to combine *Stage 3* of one protocol run with *Stage 1* of the next protocol run. This is possible since a commitment generated by a receiver node does not need to depend on the message sent by the source. Therefore, we can let each receiver generate a commitment (for the next message) and send it along the tree to the source, anticipating the subsequent protocol run. As a result, Stage 2 of the protocol can be combined with, or piggy-backed over, Stage 0 (the actual message delivery stage).

In more detail, when the source sends the very first message m_1, the protocol executes in full, as described in Section 2, except that, in *Stage 3*, each receiver i prepares for the next protocol execution (for a future m_2) by appending to the message (containing z_i^1) a tuple: $[r_i^2,\ c_i^2]$. (Note that the superscript represents the sequence number of the protocol run, i.e., its index.) Then, in the next protocol run, Stage 0 and Stage 2 are combined and Stage 1 is skipped altogether. In other words, the protocol starts with the source propagating the message m_2 together with the aggregated challenges. Similarly, in *Stage 3*, when each member replies with its signature z_i^2, it appends a tuple: $[r_i^3,\ c_i^3]$ for the subsequent protocol run. Although this optimization does not lower overall bandwidth overhead of our scheme, it significantly reduces the number of rounds (and thus the latency) since the optimized protocol only has two stages: message delivery and acknowledgment gathering.

This piggybacking does not violate the requirement that no player engages in two instances of this protocol at once: That requirement is used to enable efficient rewinding in the simulation, which would be problematic if the protocol asked player N_i to start the second instance of the protocol by issuing commitment r_i^2 before that player "consumed" the challenge c^1 and issued its response z_i^1 in the previous instance of the protocol. The optimized protocol we propose here remains secure because it maintains this requirement: Player N_i issues his commitment r_i^2 after issuing response z_i^1 (and only if he does issue this response), and sends the two together.

6 Conclusions

In this paper we focused on the problem of secure and efficient aggregation of acknowledgments in reliable multicast settings. We constructed a modification of a prior multisignature scheme that is scalable and able to tolerate failure of a small number of participants, and examined its application to acknowledgment aggregation. To reduce the round complexity of our proposal, we suggested an optimization that allows us to match the round complexity of prior state-of-the-art represented by the scheme of [1, 11], while ensuring security of the resulting solution under weaker and more standard cryptographic assumption of the hardness of discrete logarithm.

References

1. A. Boldyreva. Efficient threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography 2003*, 2003.
2. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiable encrypted signatures from bilinear maps. In *Advances in Cryptology - EUROCRYPT 2003*, 2003.
3. Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001*, pages 514–532, 2001.
4. Claude Castelluccia, Stanisław Jarecki, Jihye Kim, and Gene Tsudik. A robust multisignature scheme with applications to multicast acknowledgement aggregation. In *Security of Communication Networks: 4th International Conference, SCN'04*, pages 193 – 208, 2004.
5. Martin Gagne. Applications of bilinear maps in cryptography. Master's thesis, University of Waterloo, 2002.
6. L.C. Guillou and J.-J. Quisquater. A practical zero-knolwedge protocol fitted to security microprocessor minimizing both transmission and memory. In *Advances in Cryptology - EUROCRYPT 1988*, 1988.
7. A. Joux. The weil and tate pairings as building blocks for public key cryptosystems. In *Proceedings of the 5th International Symposium on Algorithmic Number Theory*, 2002.
8. Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology - CRYPTO 1989*, 1989.
9. S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures. In *ACM Conference on Computer and Communications Security*, October 2001.
10. S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures. available from *www.cs.bu.edu/∼reyzin/research.html*, 2001.

11. A. Nicolosi and D. Mazieres. Secure acknowledgement of multicast messages in open peer-to-peer networks. In *3rd International Workshop on Peer-to-Peer Systems (IPTPS '04)*, San Diego, CA, February 2004.
12. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361 – 396, 2000.
13. C. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology - CRYPT0 1989*, Santa Barbara, CA, August 1989.

A Robust Multisignature based on GQ Signatures

The robustness mechanism described in Section 2 can be applied to other multisignature schemes which, similarly to the multisignature scheme of Micali et al. [9], are based on signature schemes built from three-round zero-knowledge proofs where the first round is a one-time commitment. For example, our technique can be used with the Guillou-Quisquater (GQ) signature scheme [6] which is secure under the RSA assumption in the random oracle model [12]. As shown by Pointcheval and Stern [12], the security argument for the GQ signature is identical to the security argument for the Schnorr signature, because both signature schemes are built using a three-round zero-knowledge proof. The only difference is that they use different one-way functions: Schnorr signatures use exponentiation of a known base to a secret exponent modulo a prime, while GQ signatures use exponentiation of a secret element to a known 80-bit exponent modulo an RSA modulus. Since the security proof of our Schnorr-based multisignature scheme presented in section 3 adopts the Pointcheval-Stern argument, and the same argument holds for the GQ signature scheme, an extension of this argument to the case of the multisignature scheme based on GQ signatures, is virtually the same, except that the argument relies on the hardness of taking roots modulo an RSA modulus instead of on the hardness of discrete logarithm.

In the GQ scheme, each signer A gets a public key (n, e, J_A) and a private key a, such that $J_A * a^e = 1 \bmod n$, where J_A is a unique identifier of the signer, n is the product of two large primes p and q, and $e \in \{1, .., n-1\}$ such that $gcd(e, (p-1)*(q-1)) = 1$. A signs a message m by producing a one-time commitment $r = k^e \bmod n$ for a random k, computing a challenge $c = h(m||r)$ and issuing a response $z = k * a^c \bmod n$. The signature is then the pair (c, z) s.t. $c = h(m||u)$ for $u = z^e * J_A^c \bmod n$.

If we assume that a common third trusted party (TTP) computes n and provides to each member i a unique public key (n, e, J_i) and a corresponding private key a_i, several GQ signatures (c_i, z_i), produced by members of G, can be aggregated into a single multisignature as follows:

– First, each member broadcasts its own $r_i = k_i^e \bmod n$ to the group.
– Next, each player computes $r = \prod_{i \in G} r_i$, $c = h(m||r)$, $z_i = k_i * a_i^c$ and broadcasts z_i.
– The pair (c, z), where $z = \prod_{i \in G} z_i$, is a GQ multisignature for group G, with $(n, e, \prod_{i \in G} J_i)$ as a combined verification (public) key.

As with [9], this multisignature is not robust against node or link failures during the computation of the multisignature. Fortunately, this can be addressed with the same robustness technique described in Section 2, i.e. by computing the challenge c via a Merkle hash tree aggregation of the values c_i, instead of simply multiplying them.

One advantage of the Schnorr-based scheme is that it only requires a single on-line modular multiplication by the members (the signers), provided exponentiation may be done as a precomputation. It is therefore better suited for CPU-constrained receivers. In contrast, the GQ-based scheme requires one on-line modular multiplication and one on-line modular exponentiation (with an 80-bit exponent). If precomputation is not possible, the Schnorr-based scheme requires one additional 160-bit modular exponentiation, whereas, the GQ-based scheme requires one 80-bit modular exponentiation. The GQ-based scheme thus becomes more efficient.

Multisignature verification in the Schnorr-based scheme is more expensive than in its GQ counterpart. The former requires one multiplication, one exponentiation with an 80-bit value and one – with a 160-bit value. GQ-based scheme only requires one multiplication and one 80-bit exponentiation.