A Robust Multisignature Scheme with Applications to Acknowledgement Aggregation

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, 655 Avenue de l'Europe, 38334 Saint Ismier CEDEX, France
claude.castelluccia@inrialpes.fr

Abstract. A multicast communication source often needs to securely verify which multicast group members have received a message, but verification of individually signed acknowledgments from each member would impose a significant computation and communication cost. As pointed out by Nicolosi and Mazieres [NM04], such cost is minimized if the intermediate nodes along the multicast distribution tree aggregate the individual signatures generated by the multicast receivers into a single *multisignature*.
While the solution of [NM04], based on a multisignature scheme of Boldyreva [Bol03], relied on so-called "Gap Diffie-Hellman" groups, we propose a solution using a multisignature scheme which is secure under just the discrete logarithm assumption. However, unlike the previously known discrete-log based multisignature scheme of Micali et al. [MOR01a], our multisignature scheme is robust, which allows for an efficient multisignature generation even in the presence of (possibly malicious) node and communication failures.

1 Introduction

Multicast (or one-to-many) communication is widespread in a variety of settings. Popular examples include IP Multicast, p2p content sharing, digital cable TV transmission, mobile ad hoc networks 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 encryption and efficient content authentication. One of the related issues that has not been sufficiently considered is the problem of secure (authenticated) acknowledgments. After sending out a multicast message, the source is often interested in establishing 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, and provably secure under long-standing cryptographic assumptions like the hardness of computing discrete logarithms.

C. Blundo and S. Cimato (Eds.): SCN 2004, LNCS 3352, pp. 193–207, 2005.

Importance of Multicast Acknowledgment Aggregation. We assume that the packets are sent from the source to the members along a delivery tree. This tree is rooted at the source and the members are represented as leaves and, possibly, also as intermediate nodes. The delivery tree is no necessarily binary, i.e., a node can have more than two children. This model is general enough to cover the standard client-server and peer-to-peer multicast flavors. In the former, the group members are the leaves, whereas, in the latter, intermediate nodes can also be group members. However, for the sake of simplicity in the presentation, we will 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 in many application will be a scarce resource. While the computational cost of verifying the individual acknowledgments can be sped up by various batch signature verification techniques, such techniques do not address the need to save the communication resources as well.

Prior Art: Acknowledgment Aggregation Using Multisignatures Based on GDH Groups. Nicolosi and Mazieres [NM04] recently proposed to reduce the computation and the communication costs associated with acknowledgment verification by aggregating the acknowledgments using a multisignaturescheme of Boldyreva [Bol03]. A multisignature scheme is a generalization of the standard notion of a signature to messages signed by *groups* of users. It was formally defined only recently by Micali et al. in [MOR01a],[1] a long time after the (less formal) introduction of this concept by Itakura and Nakamura [IN83], and after several such schemes were proposed and a few were shown to have serious security vulnerabilities. In a multisignature scheme 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.[2]

It is easy to illustrate multisignatures using the multisignature scheme of Boldyreva [Bol03], which is a generalization of a regular signature scheme proposed by Boneh et al. [BLS01]. 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 signature verification consists of checking that (g, y, M, s) is a Diffie-Hellman tuple. Boldyreva's multisignature scheme generalizes the BLS signatures by defining

[1] A full version of this paper is available as [MOR01b].

[2] Thus multisignatures, referred to as "accountable subgroup multisignatures" by Micali et al., are a special case of so-called "aggregate signatures" [BGLS03] which enable aggregation of signatures by multiple signers on possibly *different* messages.

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 $F_x(z) = (z)^x$ on a random group element z is intractable without the knowledge of x. GDH is assumed to hold for certain 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 [Jou02, Gag02].

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 on the multicast delivery tree: Each intermediate node can verify, given the (combined) public keys of the nodes below him, whether the (aggregated) acknowledgments he 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 are aggregated in it. Note that the scheme uses constant bandwidth on every link, and that the cost of the 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 the malicious participants who send bogus acknowledgments.

Our Contribution: A Robust DL-Based Multisignature and Acknowledgment Aggregation. While efficient and robust, the above scheme is based on relatively new cryptographic assumption of GDH. In this paper we show that a robust multisignature scheme, and thus a robust acknowledgment aggregation, can be done securely based (in ROM) on a more standard cryptographic assumption of hardness of discrete logarithm. Our solution is an improvement on the DL-based multisignature scheme proposed by Micali et al. [MOR01a]. Just as the multisignature of [MOR01a], our scheme is a variant of the Schnorr's signature scheme [Sch89], provably secure (in ROM) under the discrete logarithm assumption. However, by tying together the individual players' commitments in Schnorr signatures with Merkle tree hashes [Mer89], our multisignature scheme has a novel property of *robustness*, because it enables the group of signers to efficiently generate a multisignature even in the presence of (some number of) communication failures between the participating players and/or malicious behavior on the part of (some of) the players. By contrast, the multisignature scheme of [MOR01a] would have to be restarted from scratch in the case of a single communication or node fault during a multisignature generation protocol.

Our robust multisignature scheme is provably secure only for a limited number of faults t. Specifically, if q is the size of the multiplicative group this discrete-log scheme is instantiated with and n is the maximum number of players allowed

to participate in the multisignature generation protocol, then our scheme is secure as long as quantity $S_{t,n}/q$ is negligible, where $S_{t,n}$ is a sum of consecutive combinations, $S_{t,n} = \binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{t}$. Although we do not see any attack on our scheme for larger values of n and t, our proof of security does not extend beyond these limitations, and an existence of a discrete-log multisignature scheme which is robust (without having to re-start the generation protocol) against any number of faults remains an open problem. However, we note that our scheme works for parameters like $(q, n, t) = (2^{1024}, 2^{10}, 100)$, which should be useful in practice. Furthermore, note that the if the number of faults t crosses the above limit the multisignature protocol fails and needs to be restarted for the remaining players, hence the above bounds really limit only the robustness property, and not security.

The robustness property we introduce to Schnorr-based multisignatures comes either at no extra communication cost, or at a modest communication cost increase, depending on the communication medium connecting the players. In the case when the players communicate in a ring as in [MOR01b], the total communication cost grows from $O(n)$ group elements to $O(n \log n)$. If the players communicate via a reliable broadcast medium, as in [MOR01a], then the communication costs do not change. Finally, if the players communicate via a multicast tree, as is the case in our application of multisignatures to multicast acknowledgement aggregation, the total communication cost is $O(n + t(\log n)^2)$ group elements, where t is the number of faults. This is the communication setting in which we will describe our multisignature scheme, but the scheme is applicable to the other settings as well.

When we apply our multisignatures to multicast acknowledgement aggregation, the comparison of the resulting scheme to that of Nicolosi and Mazieres [NM04] 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 one round of communication and total bandwidth of $O(n)$ group elements, where n is the size of the multicast group. Moreover, our scheme has a smaller *computational* overhead because we avoid the pairing operations used in the GDH-based scheme of [NM04]. In the case of malicious node faults, our robustness mechanisms kick in and the scheme takes three communication rounds, and the total bandwidth grows to $O(n + t(\log n)^2)$ bandwidth where t is the number of faults, whereas the scheme of [NM04] takes only one round and the total bandwidth remains $O(n)$. Our scheme is therefore most applicable when the number of malicious faults and link failures is moderate, which we believe is the case for many applications.

Limitations of Current Multisignature Schemes. We point out that the multisignature scheme we propose continues to suffer from the problem identified by Micali et al., a problem which is shared by both the scheme of Micali et al. and by the scheme of Boldyreva. Namely, none of these schemes, including ours, is known to be secure without special requirements on the generation of the par-

ticipants' public keys. Micali et al. list a number of such possible requirements on the key-generation process (see esp. [MOR01b]), which apply equally to the scheme of Micali et al., Boldyreva, and ours, but we will mention here only two.

The first requirement under which all three schemes are provably secure is the assumption that *all* certificate authorities who certify the participants' public keys are first of all honest, and second, that they verify a zero-knowledge proof of knowledge of the private key when certifying some user's private key. As pointed out in [MOR01a], this requirement makes delegation problematic, disallows self-delegation completely, and is probably sensible only when all certificates are signed by very few completely trusted entities. The second possible requirement is that all participants generate and certify their public keys in a special distributed protocol. While this requirement avoids trusted third parties completely, it is applicable only to small groups, and is unsuitable for general public key infrastructure.

Moreover, unlike in the scheme of Boldyreva but like in the scheme of Micali et al., we will require that the players involved in the multisignature generation protocol take as input the set G of players (potentially) participating in this protocol.

However, while these limitations remain a serious problem for general applications of multisignatures, they do not influence the application of multisignatures to multicast acknowledgement aggregation. In this application we can assume not only that all participants' keys are certified by a single trusted certification authority, but we can in fact simply give everyone's private key to this authority. Therefore in the subsequent sections we choose to present our multisignature scheme assuming a single trusted certification authority. Similarly, in the multicast acknowledgement aggregation application it can be safely assumed that the intended set of recipients G who would participate in the multisignature generation can be known to each of the participants.

Paper Organization: In the next section we describe the proposed multisignature scheme. In section 3 we describe its optimized variant suited to multicast acknowledgement aggregation. Finally, in section 4 we sketch the security proof for our scheme.

2 A Robust Discrete-Log Based Multisignature Scheme

2.1 Computational Setting and Initialization

We propose a multisignature scheme based on an extension of the Schnorr signature scheme [Sch89]. We assume common parameters (p, q, g) where p, q are large primes and g is an element of order q in $\mathbb{Z}_p^*$. As in the Schnorr signature scheme we assume a hash function $h : \{0,1\}^* \to \mathbb{Z}_q$, which we will model as a random oracle. All equations involving multiplication or exponentiation are meant modulo p.

As mentioned in the introduction, we will assume a single trusted Certification Authority who signs all participants' public keys. We will describe our

multisignature scheme using the application to acknowledgment aggregation as a 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 the group members represented as the leaves of the delivery tree, the intermediary tree nodes occupied by the multicast delivery network, and the multicast source S at the root. We note, however, that the scheme is generally applicable, in which case the tree data structure needs to be computed by the participating players, and both the intermediary nodes and the "source" node will be just special functions played by the players to whom the data structure assigns these roles.

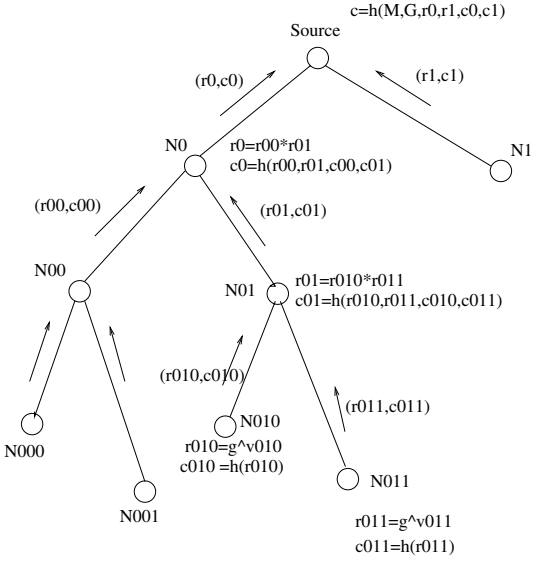

Fig. 1. Computation of the Merkle Tree

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 $x_i = DL_g(y_i)$.[3] 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.

[3] 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 [MOR01a].

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.

2.2 Overview of the Scheme

In the original Schnorr signature the signature on message M under key $y = g^x$ is generated by taking one-time "commitment" $r = g^v$ for random $v \in [0, q-1]$, computing the challenge $c = h(m, r)$, and issuing a response $z = v + cx \bmod q$. The signature is then a pair (r, z) s.t. $g^z = ry^c \bmod p$ 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-stage protocol: In the first phase everyone *broadcasts* its $r_i = g^{v_i}$, all players combine $r = \prod_{i \in G} r_i$, compute $c = h(m, r)$, each player broadcasts $z_i = v_i + cx_i$ and (c, z) where $z = \sum_{i \in G} z_i$ is a Schnorr multisignature for group G, with $y = \prod_{i \in G} y_i$ as a verification key.[4]

However, this solution is not robust in the face of node and link failures during the computation of the multisignature. For example, if any 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-like [Mer89] aggregation of the r_i values.[5] Because a Merkle Tree is a commitment to all the r_i's, the resulting challenge c is meaningful for all subsets of r_i's that were used to create it. Therefore the challenge can be used for a multisignature involving those and only those players that respond to it with proper z_i's. We note that the Merkle tree we construct is not exactly standard because we fold into it the intermediary values r_i, which allows for a more efficient handling of network or malicious faults occurring in the protocol. The exact computation of the Merkle Tree is illustrated in Figure 1.

2.3 The Multisignature Generation Protocol

Our multisignature generation protocol has 3 stages. Each player always stores all the information passing through it. As in the scheme of Micali et al. [MOR01a], for the sake of provable security we forbid the players to participate in more than one instance of this protocol at a time. Moreover, as in the scheme of Micali et al., we also require that each participant is informed about the (potential, in our case) set of participants G willing to sign the same message M.

[4] In the fuller version [MOR01b] of their work the authors show that the same scheme works also without broadcast, for example if the players communicate in a ring-like fashion. However, that version of the protocol is similarly not robust.

[5] 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, while we use it in the multisignature generation protocol instead, to enable its robustness.

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. 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 challenge $c = h(M, G, r_0, r_1, c_0, c_1)$. It then sends (c, r_1, c_1) to N_0 and (c, r_0, c_0) to N_1. N_0 then sends $(c, r_1, c_1, r_{01}, c_{01})$ to N_{00} 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.

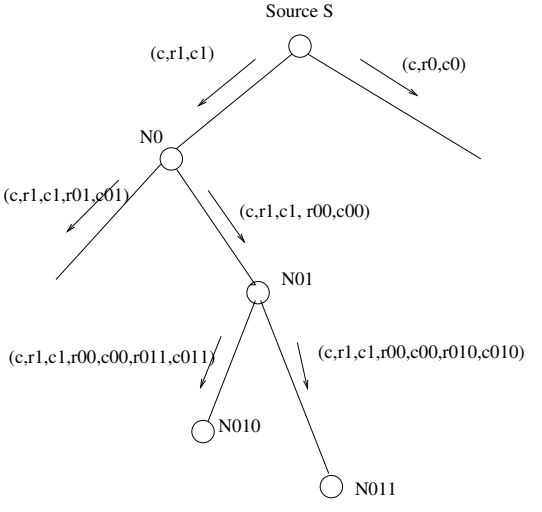

Fig. 2. Transmission of the challenge value c

As a result of this process, each member N_j receives the challenge c and the values $copath_j = \{(r_i, c_i)\}$ on its *co-path* to the root c of the Merkle tree. Every N_j can then reconstruct values $path_j = \{(r_i, c_i)\}$ that lie on its *path* to the root, and verify that c is correct. We denote this operation as checking if $c = h_{MHT}(M, G, r_j, copath_j)$. For example, N_{011} receives values c and $copath_{011} = \{(r_{010}, c_{010}), (r_{00}, c_{00}), (r_1, c_1)\}$. The verification if $c = h_{MHT}(M, G, r_{011}, copath_{011})$ 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 $c = h(M, G, r_0, r_1, c_0, c_1)$.

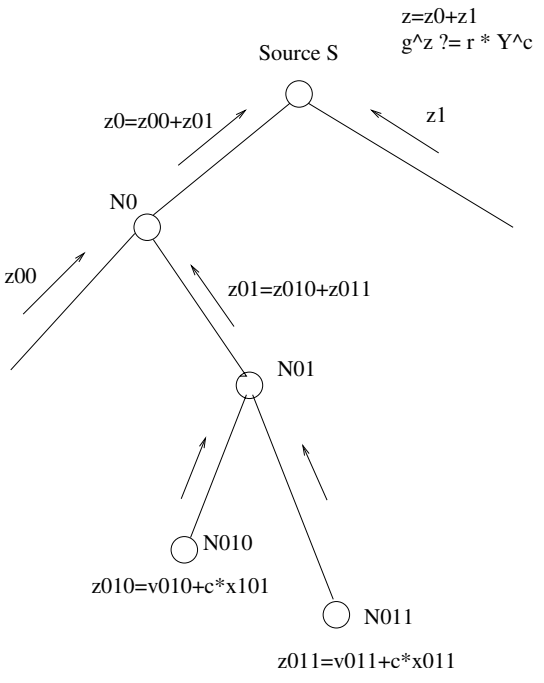

Fig. 3. Default propogation of the responses z_i

Stage 3: If the challenge c verifies, each signer N_i sends back its response $z_i = v_i + c * x_i \bmod q$. An intermediary 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 $z = z_0 + z_1$ value reaches the source, as illustrated in Figure 3.

If one of the signatures is incorrect (let's say z_{j1}), N_j sets z_j to z_{j0} instead of $z_{j0} + z_{j1}$, and sends $(z_j, r_{j0}, r_{j1}, c_{j0}, c_{j1})$ to its parent. The parent, let's say N_k such that $j = k1$, can perform two checks: (1) N_k can check if $g^{z_j} = r_j / r_{j1} * (y_j / y_{j1})^c$; and (2) N_k can check if $h(r_{j0}, r_{j1}, c_{j0}, c_{j1})$ is equal to c_j given to N_k by N_j in stage 1 of the protocol.

In general, each intermediary node N_j passes up a set $\mathcal{M}_j$ of pairs $(r_i, copath_i)$ where each r_i is a (possibly accumulated) value corresponding to players which have not delivered a valid z_i response, either due to communication failure or a malicious fault. Each node N_k upon receiving these messages first performs the following tests for both its 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 $c = h_{MHT}(M, G, r_i, copath_i)$ for each $i \in \mathcal{M}_{kb}$.

If everything verifies, N_k passes up $z_k = z_{k0} + z_{k1}$ and $\mathcal{M}_k = \mathcal{M}_{k0} \cup \mathcal{M}_{k1}$. In case of a failure in branch b, N_k passes up only the correct values, i.e. $z_k = z_{k\bar{b}}$,

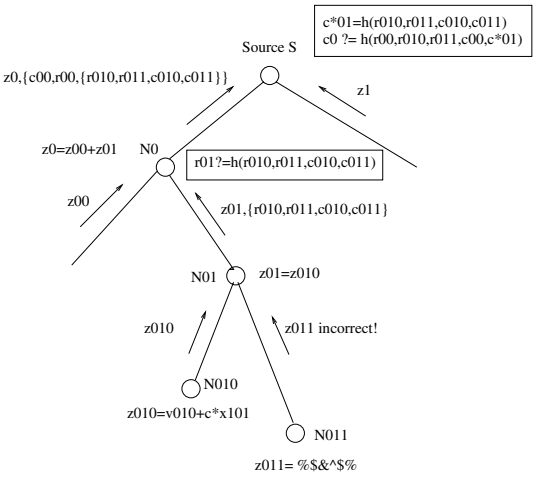

Fig. 4. Propagation of responses z_i in case of faults

and passes up the set of the missing values as $\mathcal{M}_k = \mathcal{M}_{k\bar{b}} \cup \{(r_{kb}, copath_{kb})\}$. If both branches fail, N_k passes up just $\mathcal{M}_k = \{(r_k, copath_k)\}$.

Figure 4 illustrated this step when one of the member's signature is incorrect. In this example, N_{01} detects that the signature generated by N_{011} is incorrect because $g^{z_{011}} \neq r_{011} * y_{011}^c$. N_{01} then sets z_{01} to z_{010} and forwards the message $\{z_{01}, r_{010}, r_{011}, c_{010}, c_{011}\}$ to its parent N_0. N_0 then computes $c_{01}^* = h(r_{010}, r_{011}c_{010}, c_{011})$. If c_{01}^* is equal to the value committed by N_{01} in the first stage of the protocol, then N_0 can verify if the signature z_{01} is correct by checking whether $g^{z_{01}} = r_{01}/r_{011} * (y_{01}/y_{011})^c$.

2.4 Multisignature Verification

We call σ a multisignature on message M issued by the group $G \setminus \mathcal{M}$ if

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

such that:

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

where

$$c = h(M, G, r_0, r_1, c_0, c_1)$$
$$r = r_0 * r_1$$

and moreover:

1. $c = h_{MHT}(M, G, r_i, copath_i)$ for each $i \in \mathcal{M}$, and all the co-paths contain values (r_0, r_1, c_0, c_1) in the appropriate places
2. $|G| \leq n_{max}$, and the number of *individual participants* (implicitly) specified by the missing set $\mathcal{M}$ is smaller or equal to t_{max}

Importantly, the criterion in point 2 above limits the number of the missing *individual participants* represented by the $\mathcal{M}$ set, and not the *size* of that set, i.e. the number of (possibly aggregated) r_i values supplied in $\{(r_i, copath_i)\}_{i \in \mathcal{M}}$.

The values n_{max}, t_{max} are set so quantity $S_{t_{max}, n_{max}}/q$ is negligible, e.g. less than 2^{-80}, where $S_{t,n} = \binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{t}$.

3 A "MultiMAC" Variant of the Multisignature Scheme

If acknowledgment non-repudiation is not required, the multicast source can have a copy of each participant's private key, in which case our aggregation scheme can be called "multiMAC" rather than a multisignature. Moreover, while the basic scheme described above requires three stages of communication, its MultiMAC variant can run in an "optimistic" fashion, which requires only one communication stage if none of the intermediary nodes acts maliciously. Since this is very likely to be the common case of operation, the communication cost of the resulting solution matches the cost of the scheme of Nicolosi and Mazieres.

In this variant, each member has a unique secret key x_i shared with the source. We assume that each such key is agreed upon or distributed at the time when the member joins the source's multicast 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 sets into one set G, it can verify if all these members indeed acknowledge receipt of M by checking whether $ack = h(M)^{x_G}$.

Note that this "optimized" protocol by itself is secure but not robust against malicious faults. It is, however, robust against non-malicious communication faults. Note also that to save memory, the source could 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 the secrets keys since it can easily retrieve each of them.

The optimization allows the source to verify the aggregated acknowledgment in one stage. However, this solution is not robust since, if the aggregated acknowledgment is invalid, the source is unable to identify the malicious member(s). We therefore propose 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 – in one stage – the aggregated acknowledgment. If this acknowledgment is incorrect, *Stage 2* and *Stage 3* of the basic scheme can be executed to trace the malicious nodes and robustly compute the desired multisignature.

4 Security Analysis of the New Multisignature Scheme

We briefly recall the definition of security for a multisignature scheme given by Micali et al. [MOR01a]. The adversary A can corrupt any group member at any time, 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 gets to participate in the multisignature generation protocol involving group G and message M.

Definition 1. ([MOR01a]) *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 on M.*

The multisignature scheme resulting from the addition of the optimistic "multiMAC" protocol as explained in section 3, is similar enough to the basic scheme of section 2 that its security proof follows very simply from the security proof for the basic multisignature scheme, given below.

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 [MOR01a]. 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 [PS96] of security of Schnorr signatures, except that as in the security proof [MOR01b] of the Schnorr-based multisignature scheme of Micali et al., the simulator will need 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, r_0, r_1, c_0, c_1) to the h oracle. If this is not the c that comes down to N_n in the second stage of the protocol together with some co-path $copath_n$ such that $c = h_{MHT}(M, G, r_n, copath_n)$, 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.) If q_h is the maximal number of hash queries made by A, this trial and error procedure will eventually succeed in expected number of at most q_h repeats,[6] which will slow 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. Similarly as in the "forking lemma" proof of [PS96], we can argue that with high enough probability it is the case that if A uses some values $(M, G, (r_0, r_1, c_0, c_1))$ in this forgery, then A has a high enough probability of forging a message using the same tuple of values, where the probability is taken over all the remaining randomness used by the simulator in answering A's oracle queries, *including* the randomness c used in answering the very query $c = h(M, G, r_0, r_1, c_0, c_1)$. Thus, following the "forking lemma" technique, the simulator re-runs the adversary A from the point of this query on, each time using fresh randomness and thus answering this query with a fresh random value c. In any successful round of such rewinding, the simulator gets a forgery which consists of:

1. set $\mathcal{M}$ such that the number of individual participants implicitly specified by this set is no more than t_{max}
   (For simplicity, we will use $\mathcal{M}$ here to describe this set of participants; Note that then $\mathcal{M} \subseteq G$ and $n \notin \mathcal{M}$.)
2. a set of pairs $\{(r_i, copath_i)\}_{i \in \mathcal{M}}$ s.t. for every $i \in \mathcal{M}$ we have $c = h_{MHT}(M, G, r_i, copath_i)$, and all the co-paths $copath_i$ contain values that match value (r_0, r_1, c_0, c_1) above
3. value z s.t. $g^z = r/r_{\mathcal{M}} * (y_n \bar{y}/y_{\mathcal{M}})^c$, where $r = r_0 * r_1$, $r_{\mathcal{M}} = \prod_{i \in \mathcal{M}} r_i$, $\bar{y} = \prod_{i \in G \setminus \{n\}} y_i$, and $y_{\mathcal{M}} = \prod_{i \in \mathcal{M}} y_i$

[6] Thanks to the Merkle Tree hashing, this bound can be improved to q_h/n.

Note that if there are n members in G then there can be at most n values r_i which the adversary can "open" in item (2) above, unless A finds a collision in the hash function, but that can 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 $v, v_1, \ldots, v_n, x_n$:

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

For every successful round of such re-run of A, the simulator gets another equation of type (1). Once the simulator gets $n + 2$ of such equations then with an overwhelming probability it can can solve for x_n (and thus answer its DLP challenge). This is because for every choice of membership in the set $\mathcal{M}$, there is at most one c which can make the new equation linearly dependent on the previous ones. Thus the number of c's which can possibly make the new equation dependent on the previous ones is at most $S_{t,n}$. Since c is 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 rewinding A creates a polynomial factor blow-up in the running time of the simulation. However, it is not substantially worse then the blow-up encountered in the security argument for the regular Schnorr signature scheme, because the expected number of simulation rewindings that leads to a single successful forgery is the same as in Schnorr signatures, and since we need $n + 2$ successes, our simulation running time will only grow by the additional factor of $n + 2$.

References

[BGLS03] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiable encrypted signatures from bilinear maps. In *Advances in Cryptology - Eurocrypt 2003*, 2003.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. In Colin Boyd, editor, *ASIACRYPT'01*, pages 514–532, 2001.

[Bol03] A. Boldyreva. Efficient threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography 2003*, 2003.

[Gag02] Martin Gagne. Applications of bilinear maps in cryptography. Master's thesis, University of Waterloo, 2002.

[IN83] K. Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. In *NEC Research and Development (71):1-8*, October 1983.

[Jou02] 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.

[Mer89] Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology - CRYPTO 1989*, 1989.

[MOR01a] S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures. In *ACM Conference on Computer and Communications Security*, October 2001.

[MOR01b] S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures. available from *www.cs.bu.edu/∼reyzin/research.html*, 2001.

[NM04] 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.

[PS96] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology - Eurocrypt 1996*, pages 387 – 398, 1996.

[Sch89] C. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology - CRYPT0 1989*, Santa Barbara, CA, August 1989.