

Accountable-Subgroup Multisignatures

[Extended Abstract]

Silvio Micali
MIT
Lab for Computer Science
545 Technology Square
Cambridge, MA 02139
USA

Kazuo Ohta
The University of
Electro-Communications
1-5-1 Chofugaoka, Chofu-shi
Tokyo 182-8585, JAPAN
ota@ice.uec.ac.jp

Leonid Reyzin
Boston University
Dept. of Computer Science
111 Cummington St
Boston, MA 02215 USA
reyzin@bu.edu

ABSTRACT

Formal models and security proofs are especially important for multisignatures: in contrast to threshold signatures, no precise definitions were ever provided for such schemes, and some proposals were subsequently broken.

In this paper, we formalize and implement a variant of multi-signature schemes, *Accountable-Subgroup Multisignatures (ASM)*. In essence, ASM schemes enable any subgroup, S , of a given group, G , of potential signers, to sign efficiently a message M so that the signature *provably reveals* the identities of the signers in S to any verifier.

Specifically, we provide:

1. The first formal model of security for *multisignature* schemes that explicitly includes key generation (without relying on trusted third parties);
2. A protocol, based on Schnorr's signature scheme [33], that is both *provable* and *efficient*:
 - Only three rounds of communication are required per signature.
 - The signing time per signer is the same as for the single-signer Schnorr scheme, regardless of the number of signers.
 - The verification time is only slightly greater than that for the single-signer Schnorr scheme.
 - The signature length is the same as for the single-signer Schnorr scheme, regardless of the number of signers.

Our proof of security relies on random oracles and the hardness of the Discrete Log Problem.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CCS'01, November 5-8, 2001, Philadelphia, Pennsylvania, USA.
Copyright 2001 ACM 1-58113-385-5/01/0011 ...\$5.00.

Categories and Subject Descriptors

K.6.5 [Management of Computing and Information Systems]: Security and Protection—*Authentication*; C.2.2 [Computer-Communication Networks]: Network Protocols; K.4.4 [Computers and Society]: Electronic Commerce

General Terms

Algorithms, Design, Security, Theory

Keywords

Digital Signature, Multisignature

1. INTRODUCTION

Since their introduction by Itakura and Nakamura in [19], multisignatures have been extensively studied, and yet no formal definition of this notion has been provided to date. This lack of formalism has led to some confusion as to the precise meaning of multisignature, as well as to some proposals that have been subsequently broken. We thus wish to address both problems.

1.1 Defining Multisignatures

1.1.1 The Need for Flexible Accountability

Given a group of potential signers $G = P_1, \dots, P_L$, multisignatures allow certain subsets S of signers (henceforth called subgroups) to sign messages together. A simple case, explicitly addressed, for example, in [27, 29], envisages $S = G$, that is, every signer in G must participate in producing a multisignature. The case of more general subgroups S has been, in particular, addressed in [19, 17, 22, 18].

General subgroups of signers are needed in many applications. (For example, if a certification authority is distributed on L servers, it would be useful that only a subset of the servers is needed to issue a valid certificate.) But: *what properties should such general multisignatures satisfy?* We suggest the following two (informally, for now):

- *Flexibility.* Flexibility means that *any subgroup* S of G may easily jointly sign a document. It is then up to the verifier of the signature to decide whether S was “sufficient” for the signature to be deemed valid.

This places no restriction on the size of S . In fact, it allows complex rules about S to be enforced, such as, for example, the following: “either the CEO is in S , or three senior VPs are in S .”

- *Accountability.* Accountability means that, without use of trusted third parties, individual signers can be identified from the signed document. This is desirable in many settings: if an incorrectly issued certificate is discovered, it is important to identify the corrupted servers.

Notice that, in the “paper world,” these two properties are satisfied by the traditional solution in which every member of S appends his own signature to the document to be jointly signed.

1.1.2 The Need for a Formal Model

Because of the lack of a formal model of security for multisignatures, there has been no consensus on the precise meaning of the term. Even more troublesome is the fact that few of the previous proposals for multisignature schemes ever attempted a formal security proof, and some proposals turned out, in fact, to be insecure. For example, the proposals of [17] and [22] were subsequently cryptanalyzed by [18], [21] and [26].

Absent a complete formal model, even schemes that have formal proofs of “security” are vulnerable to attacks: the “Type II” scheme of [29] is subject to a previously known attack on key generation, even though the paper contains formal statements about the scheme’s security (see footnote 3 in Section 3.2 for more details). This is so not because the theorems of [29] are wrong, but rather because the definition of security simply does not address adversarial behavior during key generation.

1.1.3 Our Model

In Section 2, we provide a formal model for multisignatures that requires both flexibility and accountability. To avoid confusion with other similar notions, we use the term “accountable-subgroup multisignatures” (ASM) to describe the schemes in our model. The model is precise enough to allow for formal security proofs, and, unlike the definition of [29], encompasses key generation. The adversary allowed in our model is quite strong, thus ensuring that ASM schemes are, in fact, secure.

1.1.4 ASM vs. Other Notions

There have been a number other notions proposed for signatures by multiple signers. The terms “multisignature,” “group signature,” and “threshold signature” are not always used consistently; moreover, other terms, such as “threshold-multisignatures” [22] are also sometimes used. We focus here only on the most commonly used models and terminology.

GROUP SIGNATURES. In a *group signature* [8, 6]), there is a total group G of potential signers, but each signature is produced by any (anonymous) *individual member* on behalf of the entire group. Therefore, group signatures are not flexible: there is no mechanism for enforcing larger than one-member subgroups. Furthermore, group signatures offer only *partial* accountability: the identity of the signing member is unknown to a signature verifier, though it is available

to a trusted party, the *group manager*, in case problems arise.¹

THRESHOLD SIGNATURES. In a *threshold signature* scheme (see, e.g., [10, 13, 14]), a message can be signed only by a sufficiently large subgroup. Therefore, threshold schemes are reasonably flexible, but do not offer accountability. In fact, there is no way of identifying the members of S from a signature.²

Such anonymity, while desirable in some settings, is dangerous in others. By removing accountability for what is signed, one may actually encourage cheating. If some senior officers of a corporation can enrich themselves (e.g., by signing the sale of certain corporate assets) with a mathematical guarantee of anonymity, then such enrichments would become more frequent than desirable.

1.2 Designing Multisignatures

1.2.1 The Key Generation Problem

Vulnerabilities in the key generation stage are a main problem in discrete-logarithm-based multisignatures. In particular, they are the basis for the aforementioned attacks on [17], [22] and [29]. The problem stems from the fact that adversarial signers can choose their keys after seeing other signers’ keys, and can therefore potentially affect the joint public key of the subgroup S .

Past discrete-logarithm-based multisignatures have therefore required a trusted third party for key generation (in particular, [18] suggests using a trusted third party to fix the problem in [17]).

Key generation has also been a problem for threshold signatures. In this case, fortunately, the problem has been resolved, at least for the discrete logarithm case (see, e.g., [31], [15]). However, threshold signatures, by design, exclude accountability, and thus the solutions available for them do not easily extend to multisignatures (see 2.1).

1.2.2 Our Implementation

In Section 3, we provide an efficient implementation of ASM schemes, based on the Discrete Logarithm problem and the Schnorr [33] signature scheme. Our starting point is a “two-cycle” scheme suggested briefly in Section 6 of [29]. We modify it extensively, however, to obtain a scheme that is provably secure in our model and does not suffer from the vulnerabilities during key generation that the original scheme has.

Our scheme has the following desirable efficiency properties, when compared to the obvious solution that simply uses multiple single-signer signatures:

- *The signature length does not grow with the number of signers: it is the same as that of a single-signer Schnorr signature.*

¹This reliance on the trusted manager, who could be “eliminated” (at least electronically —e.g., by destroying its computer) to avoid accountability, is a drawback in many settings.

²In essence, in most threshold schemes, signing involves (implicitly) interpolating an $(\ell - 1)$ -degree polynomial from ℓ point-value pairs. Thus, once the polynomial is (implicitly) reconstructed, there is no way of knowing which ℓ out of the possible L point-value pairs were actually used in the interpolation.

- The verification is almost the same as the time needed to verify a single Schnorr signature.

The cost of these improvements is quite moderate:

- The signing protocol requires only three rounds of communication among the members of S , irrespective of the size of S .
- The signing time per member of S is almost the same as the time required to produce a single-signer Schnorr signature.
- Only the key generation protocol (done once for a group G) requires each member of G to perform communication and computation that is linear in the size of G .

1.2.3 Comparison with Prior Implementations

Implementations based on repeated use of trapdoor permutations (such as [19] and [30]) require as many rounds of communication as there are signers; the verification time grows linearly with the number of signers as well. Additionally, they are not provable in our model (although it is possible to modify the scheme of [30] to make it provable at the expense of some efficiency loss).

Implementations based on discrete logarithms or the Fiat-Shamir scheme (such as, for example, [27, 18, 28, 29], and the first scheme of [22]) require a trusted third party for key generation. Implementations that do not require a trusted party include the second scheme of [22] and the multisignature scheme of [17], both of which, however, have been successfully attacked (e.g., by [21, 26]). An implementation that does not require a trusted party and can be proven secure in our model without major modifications is the “Type III” scheme of [28, 29]. However, its security holds only for *logarithmically many signers*, and the scheme requires significantly more verification time per signature.

2. THE NOTION OF AN ASM

2.1 The Informal Notion

Let us now informally express what an ASM scheme is, that is, what it means to efficiently allow *any subgroup* S of a group G of signers to sign a message, keeping each individual member of S *accountable* for what S signs.

Informal Definition *An accountable-subgroup multisignature of a subgroup of signers S for a message M provides, without any trusted managers or third parties, a self-contained and universally verifiable proof of (1) the composition of S and (2) the fact that each member of S stood behind M .*

The above definition, due to its informality, is quite vague. For instance, it does not specify whether S should be chosen in advance, or whether the members of S are aware of their co-signers. To remove such ambiguities, ASM schemes are best defined by specifying a *basic solution* that may be quite inefficient, but spells out the *rules of the game*. The goal of an ASM scheme will then be achieving the properties of the basic solution *more efficiently*.

The Inefficient Solution: Let G be a group of signers, in which each member P_i has his own individual public key PK_i and corresponding secret key SK_i . Arbitrarily order the members of the group. Then, the

signature of subgroup $S = (P_{i_1}, \dots, P_{i_m})$ of a message M consists of the sequence $(\sigma_1, \dots, \sigma_m)$, where σ_j is P_{i_j} 's individual signature (relative to public key PK_{i_j}) of the pair (M, S) .

A formal definition of an ASM scheme is essentially obtained by adding a proper adversarial model to the basic solution. We do this in Section 2.2. For now, let us understand better the properties of our basic solution. To this end, it is important to realize that the following simple-minded attempt at adding accountability to existing threshold schemes does not work.

A Non-Solution: Start with any threshold signature scheme, and add the requirement that each signed message include the list of the members of S who participated in the signing protocol.

This does not work, because any player P not belonging to S can be “framed” by the members of S , who can simply add P 's name to the list. Moreover, the real signers may not include their own names in the list. In a 3-out-of-6 scheme, for example, this would be particularly unacceptable: the signers may be a, b , and c , but they may pretend to be d, e and f instead. Indeed, even if the threshold were to be higher than half of G , one could only conclude that one of the purported signers is genuine, but it is still impossible to determine which one.

2.1.1 Understanding the informal notion

We take our inefficient basic solution to be the ideal in terms of the properties we want from ASM schemes. Our goal is to define a model that captures these properties, and come up with schemes in this model that are more efficient than the basic solution (note that the basic solution is quite inefficient: both the signature size and the verification time grow linearly with the size of S). We therefore gain better understanding of the new notion by further examining the basic solution.

THE ROLE OF S . Specifying S in the signed message ensures that each signer know his co-signers in advance: if all members of S do not individually sign (M, S) , the signature is not deemed legitimate. Thus, each signer knows precisely with whom he shares responsibility for the signed message. (In fact, some attacks on previous multisignature schemes resulted not in forged signatures, but in providing a signer incorrect information about his co-signers.)

One may envisage a different basic solution in which any member of G that wishes to sign M does so, sending the signature to a clerk. The clerk then “constructs” S using the signatures received. This approach has the advantage of not requiring any advance agreement on S . The disadvantage, however, is that the clerk, acting adversarially, can become a “guardian angel” for some signers. Suppose a member of G wants to get a document M signed but prefers not to have to take responsibility for it if possible; that is, if sufficiently many other signers are willing to sign M , she would like her signature to be excluded. She can then send her signature to the clerk, but the clerk will include it in the final output only if it is absolutely necessary for the document to be deemed valid.

Of course, both notions are valid and have applications. In this paper, however, we only consider the first one, in

which S is explicitly specified. It is worth pointing out that designing a non-trivial provable scheme of the other kind, where the composition of S can be decided after the signatures are received, is an interesting open problem.

ASM vs. THRESHOLD. ASM schemes are different from threshold schemes not only because they “add accountability”, but also because they empower *any* subgroup (rather one of sufficient cardinality) to sign a message. At the same time, a threshold can be easily added to an ASM scheme: the verifier simply checks that the cardinality of S exceeds the threshold.

SECURITY vs. ROBUSTNESS. Notice that the basic solution is *secure*, in the sense that the adversary may not forge the signature of a subgroup S (containing a “good” player). Notice too, however, that the basic solution is *not robust*: if a corrupted player of S “shuts off”, then, *by definition*, S ’s signature of a message M cannot be computed. We are therefore not seeking robustness in our model. This allows us to define the model in terms of a very strong adversary who controls all possible communication lines, and thus can prevent also the messages from good players to good players from reaching their destinations. Such extreme adversarial behavior, naturally, prevents robustness. At the same time, such adversarial ability makes the security properties of ASM schemes much stronger.

WEAK ROBUSTNESS. One could endow the basic solution with a *weak robustness* property. Roughly said, if a subgroup S fails to produce S ’s signature of a message M , then at least one corrupted player P in S will be exposed. This way, the players may have the option of trying to sign M again on behalf of a different subgroup S' not containing P . Weak robustness could be achieved in the basic solution if broadcasting is available. Broadcasting could also be added (in a simple fashion) to make our ASM scheme in Section 3 weakly robust. (Of course, weak robustness could also be achieved by using some variant of *secure computation* [16, 4, 7], but with some efficiency loss.)

2.2 The Formal Notion

We will assume that the total group G consists of L signers, and that every signer is a probabilistic polynomial-time Turing machine that initially knows nothing but its unique identification number (which is, w.l.o.g., one of the numbers $1, 2, \dots, L$) and a unary value 1^k called the “security parameter.” We will also assume that 1^k is the same for all the signers. As will be explained in more detail later, we will allow the adversary to control the network connecting the members of G .

COMPONENTS OF AN ACCOUNTABLE-SUBGROUP MULTISIGNATURE. A ASM scheme has three components:

1. A (probabilistic) *key generation protocol*.

This protocol is performed only once (at the very beginning) by all members of G . Each member receives as input a description of G , that is, the list of the identities of all members of G . (If these identities are the integers 1 through L , then it suffices to have just the integer L as an input.)

The key generation protocol produces a local output for each party P_i : a secret key, SK_i , and the corresponding public key, PK_i .

If an adversary is present during key generation, it may provide different inputs “ G ” to different parties.

2. A (probabilistic) *signing protocol*.

This protocol is performed by the actual signers in S for every message being signed. The input of each signer consists of (a) a description of the subgroup S ; (b) the public keys of the members of S ; (c) the message M ; and (d) the signer’s own secret key. The signature σ is generated jointly by the members of S , and is actually output by one of the parties in S .

If an adversary is present during an execution of the signing protocol, it may provide different inputs S and M to each signer, as well as incorrect public keys for the other signers.

3. A (deterministic) *verification algorithm*.

This algorithm is run to verify a given signature by an individual verifier, possibly not belonging to G . The inputs of the verification algorithm are: the subgroup S , the public keys of the members of S , the message M , and the alleged signature σ . The output is “YES” or “NO”.

We require that these components be “correct.” That is, suppose the signers in a subgroup S follow the protocols faithfully, and suppose key generation terminates successfully for every party in S . Then, if the signers in S perform a signing protocol on a message M (with correct inputs), they will produce a signature σ that the verification algorithm will accept (if, again, given correct inputs).

THE ADVERSARIAL MODEL. We consider an adversary F (for “forger”) with the following capabilities:

- F fully controls all messages exchanged in the network: whether the sender or the recipient is good or bad, it can read any message sent, modify it or prevent its delivery. In addition, F can send any message it wants on behalf of any player. (In a sense, therefore, there are no private or authenticated channels: all players communicate via the adversary.)
- F can *corrupt* any player at any time, during both key generation and signing. Upon corrupting a player P_i , F learns the entire internal state of P_i (including all secret information and past coin tosses).
- F controls the input of any uncorrupted player during key generation (e.g., it can specify different total groups G to different players).
- For any uncorrupted player P_i , F can conduct an adaptive *chosen-message-and-subgroup* attack: at any time, it can request that P_i execute the signing protocol on some specified message with some specified subgroup of co-signers. (Because the adversary fully controls the network, it can choose whether the co-signers will actually be really involved in this execution.)

DEFINITION OF SECURITY. Because the adversary fully controls the network, it can always prevent the parties from signing a message. Our security goal, therefore, is to prevent forgeries of new signatures.

DEFINITION 1. We will say that an ASM scheme is secure if, for all constants $c > 0$ and all sufficiently large security parameters k , no polynomial-time (in k) adversary has better than k^{-c} chance of outputting a triple (σ, M, S) such that:

- σ is a valid signature on the message M by the subgroup S of players
- there exists an uncorrupted player $P \in S$ who has never been asked by F to execute the signing protocol on M and S .

Note that, in the above definition, S may not be a subgroup of the original G . That is, we want also to prevent the adversary from adding one or more “fictitious” players, so as to (1) form a different total group G' , and then (2) be able to forge a signature of (M, S) , where S is a subgroup of G' . (Naturally, the single uncorrupted player P cannot be fictitious: it should be a member of the original G .)

Of course, as is also the case for the single-signer schemes, it is assumed that, when verifying an ASM signature of (M, S) , the verifier obtains the proper public keys of the members of $S \cap G$. (The mechanism for enforcing the authenticity and availability of such public keys is, as usual, outside the scope of our definition.) The public keys of the fictitious players ($S \setminus S \cap G$) might as well be successfully faked by the adversary.

THE MEANING OF S IN A SIGNATURE. Given that there are no authentic channels and the adversary can provide incorrect inputs during the signing phase, one can reasonably ask what exactly it means for signer P_1 to be assured that she is signing M with a signer named “ P_2 ,” when P_1 doesn’t even necessarily know who P_2 is. It means the following. While P_1 may not know who P_2 is, the verifier (necessarily) must know authentically who P_2 is, and must obtain P_2 ’s authentic public key for verification. Then, assuming that P_2 has not been corrupted, P_1 is assured that the verifier will deem the signature valid only if the person whom the verifier knows as P_2 actually participated in the signing protocol on M, S .

RANDOM ORACLES. As usual, it is possible to extend the above definitions to the random oracle model, and the actual schemes we present will be in that model. To extend the definitions, we will add a second security parameter k_2 and assume the existence of an oracle $H : \{0, 1\}^* \rightarrow \{0, 1\}^{k_2}$ to which all the parties have access. As is usual in the random oracle model [3], security will be based on the assumption that the oracle is chosen at random from all functions $\{0, 1\}^* \rightarrow \{0, 1\}^{k_2}$. The adversary is now also allowed queries to H , which we will call “hash queries.”

EQUIVALENT, BUT SIMPLER ADVERSARY. The adversary described above is extremely powerful, and provides for a compelling notion of security. However, in Section A of the Appendix, we show it equivalent to a different type of adversary, for which proofs of security become much easier.

3. AN IMPLEMENTATION OF ASM

The ASM scheme proposed here has a complex key generation, but it allows for very efficient signing and verifying. Namely, a subgroup S signs a message M by means of a 3-round protocol, where each signer sends/receives a total of 3 messages and performs a single modular exponentiation.

The main cost of verification is $|S|$ modular multiplications (that need be performed only once for a given S), and two modular exponentiations. The signature length is that of a single-signer signature, and does not grow with the number of signers.

We construct our scheme by modifying the “two-cycle” scheme of Section 6 of [29]. The scheme is based on the discrete logarithm problem (DLP); more precisely, on the signature scheme of Schnorr [33], summarized below, which is known to be equivalent to the DLP in the random oracle model.

3.1 The Schnorr (Single-Signer) Signature Scheme

A user U generates two primes p and q such that q divides $p - 1$, $g \in Z_p^*$ of order q , and a random $s \in [0, q - 1]$. U ’s secret key is s and its public key is (p, q, g, I) , where $I = g^s \pmod p$. To sign a message M , U does the following:

- picks a random $r \in [0, q - 1]$;
- computes a *commitment* $X = g^r \pmod p$;
- queries the random oracle H to compute the *challenge* $e = H(X, M)$;
- computes $y = es + r \pmod q$;
- outputs (X, y) as the signature of M .

To verify a signature (X', y') for M , one computes $e' = H(X', M)$ and checks whether $g^{y'} \equiv X' \cdot I^{e'} \pmod p$.

3.2 Informal Description

This subsection provides an introduction to our scheme by presenting an (underlying) naive scheme (essentially, the “two-cycle” scheme from [29]), and then pointing out different reasons for which it does not work, together with the corresponding fixes.

THE NAIVE SCHEME. All signers in G know each other and common parameters p, q and g as in the Schnorr scheme. Each signer i randomly and independently selects $s_i \in [0, q - 1]$ and sets $I_i = g^{s_i} \pmod p$.

An (unordered) subgroup $S = \{P_{i_1}, \dots, P_{i_\ell}\}$ signs a message M in three rounds: all players i in S select a random $r_i \in [0, q - 1]$, compute individual commitments $X_i = g^{r_i} \pmod p$, and multiply their X_i ’s together (modulo p) to obtain a joint commitment, \tilde{X} . Then, all players i in S compute the joint challenge $e = H(\tilde{X}, M, S)$; the “individual signatures” $y_i = es_i + r_i \pmod q$; and finally add (modulo q) their individual signatures together to obtain \tilde{y} , and output (\tilde{X}, \tilde{y}) as S ’s signature of M .

One verifies (\tilde{X}', \tilde{y}') to be S ’s signature of M by computing $e' = H(\tilde{X}', M, S)$ and checking whether

$$g^{\tilde{y}'} \equiv \tilde{X}' \cdot \left(\prod_{P_i \in S} I_i \right)^{e'} \pmod p.$$

PROBLEMS AND FIXES.

Problem 1: How to generate common p, q and g ? Only the security parameter k and the random oracle H are assumed to be common to all players. Thus, common p, q and g can

be individually generated by the players in G using a common generating subroutine and relying on a canonical use of H as a (common) source of randomness. Unlike the Schnorr scheme, however, the adversary now also knows additional information about p , q and g , namely, the very coin tosses that generated them. This that may help the adversary solve the discrete log problem in the g -generated subgroup modulo p . (For instance, if p and q are found by running Bach’s algorithm [2], then one also gets the entire factorization of $p - 1$, which may perhaps be useful to a clever DLP algorithm.)

Fix 1. The fix simply consists of realizing this weakness and incorporating the (p, q, g) generation process into the DLP assumption. To be precise, one needs to incorporate also the performance of this generation process. (For instance, starting with a large prime q and searching for a prime $p \equiv 1 \pmod{q}$ is not known to be guaranteed to terminate in expected $\text{poly}(|q|)$ time.) To keep this fix simple, we actually propose to make q “as big as possible”, that is, we assume that one can easily find primes p of the form $2q + 1$ (though weaker assumptions, described in more detail in the full version of this paper, can be used).

Problem 2: *The naive scheme is not secure at all if the adversary attacks key generation.* Assume that player L is bad and generates his public key last by choosing a secret key $s \in [0, q - 1]$ and then setting

$$I_L = \left(\prod_{i=1}^{L-1} I_i \right)^{-1} \cdot g^s \pmod{p}.$$

In this case, player L can sign any message M it wants on behalf of the entire group G : in fact s ends up being the “group’s secret key,” corresponding to the “public key” $\prod_{i=1}^L I_i \pmod{p}$.³

Fix 2. We fix this problem by requiring that each player i provides a zero-knowledge proof of knowledge (ZKPoK) of the secret key relative to I_i (i.e., a ZKPoK of the discrete log of I_i in base g). To remove interaction in this ZKPoK, we use the random oracle á la Fiat-Shamir [12].

Problem 3: *Fix 2 is insufficient.* Indeed, what is the verifier that checks the validity of this ZKPoK? Assuming that the public-key database is implemented via a trusted center (or certifying authority), this verifier could be the center itself (this, in fact, was the solution proposed in previous works on DLP-based multisignatures). However, this is an additional requirement that also introduces potential weaknesses and constraints. Indeed, there are ways to implement exchange of public keys without trusted centers. For instance, each signer can hand its public key to all potential verifiers at the next CCS conference. Alternatively, signer i can use its previously certified public key to sign its current public key I_i . (And, though a trusted center may have been involved in publicizing the previous key, it may not be around to certify the current I_i .)

Fix 3. These problems can be solved by having each signer i include in its public key not only I_i , but also the (non-interactive, random-oracle based) ZKPoK. This results in

³ This attack has appeared in the past, in particular, in the works of [18, 21, 26]. The same attack can be used against the “Type II” scheme of [28, 29].

a moderate loss of efficiency: each public key gets slightly longer, and each verification requires $2|S| + 2$ exponentiations rather than just two. (This is so because the verifier of S ’s signature of M also needs to verify the proofs in the public keys of the members of S , and each verification takes two exponentiations). However, each public key of a member of G needs to be checked at most once by each verifier if the verifier keeps careful records.

Problem 4: *We cannot prove Fix 3 secure for many signers.* For Fix 3 to work in the security proof, it is necessary that, for each bad player P_j , a polynomial-time simulator succeeds in extracting the discrete log of I_j from the ZKPoK that the P_j provides. However, for all known proof techniques in the random-oracle model, if player P_j computes I_j after making q queries to oracle H , then the simulator succeeds in computing the discrete log of I_j with probability at most $1/q$. Thus, if there are k bad signers, then the simulator will succeed for all of them with probability at most $1/q^k$. That is, for the simulator to be polynomial time, there can be at most logarithmically many signers.

Fix 4. All signers i in G after computing their own s_i and I_i , exchange the I_i values and their commitments X_i for the ZKPoK. Then, each individual signer i proves knowledge of s_i by using the “joint challenge” $e = H(X_1, I_1, \dots, X_L, I_L)$ and the Schnorr signature relative to I_i . The above simulator can now extract the s_j for all k bad signers P_j with the total success probability of about $1/(kq)$ (because there are kq total queries to the oracle). Notice that the adversary may cause the key generation protocol to fail (and thus, it will need to be restarted with a different total group G). All in all, this is a modest problem, because the same phenomenon occurs during each signature computation, while key generation is only done once.

Problem 5: *The public keys in Fix 4 are too long.* In order for the ZKPoK in Fix 4 to be verifiable, each player i has to include its signature σ_i in its public key, as well as $X_1, I_1, \dots, X_L, I_L$ on which it was computed. In fact, the verifier better check that the vector (I_1, \dots, I_L) is the same for all members of S . Such public keys are too long, because their length is proportional to G , regardless of how small S may be. Note that, in the basic solution of Section 1, the verifier needs to retrieve only $|S|$ ordinary (and, thus, short) public keys. Even if the public key database contains a special entry for $(X_1, I_1, \dots, X_L, I_L)$, the verifier of a single signature by, say, a 3-member subgroup has to download a vector of length proportional to L .

Fix 5. After the ZKPoKs, as in Fix 4, are exchanged, each signer i verifies all of them and then computes a Merkle tree (using the random oracle H as the hash function) with the leaves I_1, \dots, I_L . (Merkle trees are recalled in Section B of the Appendix.) This Merkle tree will have depth exactly $\log L$ (for simplicity, assume that L is a power of 2). Then, signer i includes in its public key PK_i , the value I_i and the authenticating path of I_i in the Merkle tree. Notice that PK_i is quite short: it contains one public key and $\log L$ hash values (for instance, if each ordinary public key is 2000 bits long, the length of the hash values is 200, and there are 1000 potential signers, then each PK_i is only as long as two ordinary public keys).

To verify S ’s signature of M , for each signer i in S , the verifier uses I_i and the alleged authenticating path for I_i to

compute the alleged root value V_i , and checks that $V_i = V_j$ for all signers i, j in S . In a sense, if player i is honest and puts I_i together with its authenticating path into PK_i , it implicitly puts into PK_i the root value V_i , and claims that any I_j that correctly “Merkle-hashes” to V_i has been checked by i to be part of a valid ZKPoK. (Of course, a corrupted player j can use V_i so as to find some other value $I'_j \neq I_j$, also Merkle-hashing to V_i , for which it knows no ZKPoK; but this can be done only by finding a collision for the random oracle H , which is extremely hard to do.) Thus, if at least one good signer exists in S , all the other signers in S are forced to have correct keys.

Problem 6: Concurrent signing. The naive scheme is silent about the possibility of concurrent signing. That is, a good player i belonging to two subgroups S and S' , would be permitted to participate simultaneously in signing protocols for (M, S) and (M', S') . Our proof of security, however, needs to use rewinding, for reasons explained in Section 3.5. It is a well-know fact that rewinding is incompatible with concurrency (unless the amount of concurrency is very small).

Fix 6: To prevent concurrency from messing up our security proof, we do not allow a signer to begin a new signing protocol until the previous one has been completed or aborted. This is not a serious loss, given that signing in our DLP-based scheme is a 3-round protocol. (Note, however, that if two subgroups are disjoint, then their signing processes can go on simultaneously.)

3.3 The DLP' Assumption

1. *Samplability.* The following probabilistic algorithm $Gen(1^k)$ runs in expected polynomial time:

Repeat

Let q be a random k -bit string

Until q is prime and $p = 2q + 1$ is prime

Output p and q .

2. *Hardness.* For any algorithm A , denote by p_k^A the probability that, on inputs

(a) a random k -bit prime q such that $p = 2q + 1$ is also prime,

(b) a random element $g \in Z_p^*$ of order q , and

(c) a random I in the g -generated subgroup of Z_p^* ,

A outputs $s \in [0, q - 1]$ such that $I \equiv g^s \pmod{p}$. Then, \forall probabilistic polynomial-time A , $\forall c > 0$, and \forall sufficiently large k ,

$$p_k^A < k^{-c}.$$

Remark. If p and q are generated by Gen above, then it is easy to find a random element $g \in Z_p^*$ of order q (by picking a random element $r \in Z_p^*$ until $g = r^{(p-1)/q} \pmod{p} \neq 1$).

We note that our scheme continues to be provably secure based on a more general, but more complex, DLP assumption, described in the full version of this paper.

3.4 Description

• Preliminaries

Security Parameters. All players are assumed to have, as a common input, the security parameter k , and the number L of players is assumed to be polynomial in k . A second security parameter $k' < k$ is deterministically computed from k . (Typically, $k' = 100$.)⁴

Random Oracles. The players use (in any fixed standard way) the single random oracle H to implement five independent random oracles H_1, H_2, H_3, H_4 and H_5 , such that

$$\begin{aligned} H_1, H_2 &: \{0, 1\}^* \rightarrow \{0, 1\}, \\ H_3, H_5 &: \{0, 1\}^* \rightarrow \{0, 1\}^{k'}, \text{ and} \\ H_4 &: \{0, 1\}^* \rightarrow \{0, 1\}^{2k'}. \end{aligned}$$

Subroutine Gen. Key generation will use the algorithm Gen of assumption DLP' (or DLP'').

Distinguished Player. In describing the scheme, we shall use a distinguished player D . This is for convenience only, and does not require changing our model. In fact, D is a player who knows no secrets and can be implemented by any one of the players in the subgroup S , or even collectively by the members of S .

Merkle Trees. We assume some familiarity with the notion of a Merkle tree [24], whose description is given in Section B of the Appendix.

• Key generation

Common keys. All players run $Gen(1^k)$ using $H_1(2^k), H_1(2^k+1), \dots$, as the random tape in order to generate primes p and q . They then generate a random $g \in Z_p^*$ of order q using $H_2(2^k), H_2(2^k+1), \dots$, as the random tape.

Individual keys. Each player P_i ($1 \leq i \leq L$) does the following:

1. chooses $s_i \in [0, q - 1]$ and sets the secret key $SK_i = s_i$;
2. computes its *public value* $I_i = g^{s_i} \pmod{p}$;
3. chooses a random $r_i \in [0, q - 1]$ and computes a *commitment* $X_i = g^{r_i} \pmod{p}$;
4. broadcasts (X_i, I_i) to all the players;
5. upon receipt of $(X_1, I_1), \dots, (X_L, I_L)$, computes $e = H_3(X_1, I_1, \dots, X_L, I_L)$ and $y_i = es_i + r_i$;
6. broadcasts y_i to all the players;

⁴As for all Fiat-Shamir-like schemes, including Schnorr's, k' controls the security of our scheme in a way that is quite different and independent of k . Indeed, k should be large enough so that solving the discreet logarithm problem when q is a k -bit prime is practically impossible. Parameter k' should be large enough so that it is practically impossible to perform $2^{k'}$ steps of computation, and the probability $2^{-k'}$ is practically negligible. Of course, we can always set $k' = k - 1$, but our scheme can be made much more efficient while maintaining the same level of security by selecting an appropriately lower value of k' .

7. for each (X_j, y_j) received from player P_j , verifies that $g^{y_j} \equiv X_j I_j^e \pmod p$. If all checks are satisfied, P_i computes the authenticating path, Path_i , of leaf i in the L -leaf binary Merkle tree whose j -th leaf contains I_j and whose hash function is H_4 .
8. registers $PK_i = (p, q, g, I_i, \text{Path}_i)$ as its public key.

(*Comment:* p, q, g are included in PK_i only to save time for the verifier. Alternatively, p, q and g could be a special entry in the public key database, if one exists, or could be re-computed by the verifier.)

- **Signing:** Suppose the players in a subgroup $S = \{P_{i_1}, \dots, P_{i_m}\}$ wish to sign jointly a message M . Then they perform the following three-round protocol:

1. Each signer $P_{i_j} \in S$, if not currently involved in another signing protocol,
 - 1.1 picks a random $r_j \in [0, q - 1]$;
 - 1.2 computes its individual commitment $X_j = g^{r_j} \pmod p$;
 - 1.3 sends X_j to D .
2. D computes the joint commitment as the product $\tilde{X} = \prod_{j=1}^m X_j \pmod p$ and broadcasts it to each signer $P_{i_j} \in S$.
3. Each signer $P_{i_j} \in S$
 - 3.1 queries the random oracle H_5 to compute the challenge $e = H_5(\tilde{X}, M, S)$
 - 3.2 computes $y_j = es_j + r_j \pmod q$;
 - 3.3 sends y_j to D .
4. D computes $\tilde{y} = \sum_{j=1}^m y_j \pmod q$ and outputs $\sigma = (\tilde{X}, \tilde{y})$ as the signature.

(*Remark:* The role of D in steps 2 and 4 can be performed by P_{i_1} or by all players in S , e.g., “arranged in a circle” by having P_{i_j} send $\prod_{\alpha=1}^j X_\alpha$ —or $\sum_{\alpha=1}^j y_\alpha$, respectively— to $P_{i_{j+1}}$.)

- **Verification:** To verify a signature $\sigma = (\tilde{X}, \tilde{y})$ of a message M of a subgroup $S = \{P_{i_1}, \dots, P_{i_m}\}$ whose members have public keys $PK_{i_1}, \dots, PK_{i_m}$, one does the following:

1. Check that all public keys contain the same p, q and g .
2. For each $P_{i_j} \in S$, use I_{i_j} and Path_{i_j} to compute a candidate root value V_{i_j} for that player’s alleged Merkle tree, and check that all such V_{i_j} are equal.
3. Compute $\tilde{I}_S = \prod_{j=1}^m I_{i_j} \pmod p$.
4. Compute $e = H_5(\tilde{X}, M, S)$.
5. Check that $g^{\tilde{y}} \equiv \tilde{X} \tilde{I}_S^e \pmod p$.

(*Remark:* Steps 1-3 need be performed only once for a given subgroup S .)

3.5 Security

THEOREM 1. *Under the DLP’ assumption, the above is a secure ASM scheme.*

See the full version of this paper for the proof of this theorem. Below we describe the two main ingredients of the proof.

1. We use the “forking lemma” [11, 32] technique in order to violate the DLP’ assumption.

Unlike the usual forking-lemma-based proofs, however, we need to use the technique twice, because adversarial players can participate in the forgery output by the adversary. Thus, we use the forking lemma during key generation to obtain the secret keys of the adversarial players⁵, and again when the adversary outputs a forged signature on (M, S) to compute the discrete logarithm α of $\prod_{P_i \in S} I_i$. The secret keys of the adversarial players in S are then subtracted from α to get the desired discrete logarithm.

2. We use rewinding, not commonly used in similar random oracle proofs, for reasons explained below.

In ASM schemes, as in single-signer signatures, the adversary is entitled to a chosen message attack. That is, it can ask for and receive the signature of any message M it wants. However, this capability is much more powerful and dangerous for ASM schemes than for single-signer signatures.

In the single-signer Schnorr scheme (as in any other Fiat-Shamir-like scheme in the random oracle model), during a signature query the adversary can provide the (single) signer only with a message M . The signer will then select its own commitment X relative to which it produces the signature of M (via the challenge $e = H(X, M)$). Assume, however, that, in our DLP-based ASM scheme, adversary F wishes that a good player i in S to sign a message M as a member of S . Then, F will first receive i ’s individual commitment, X_i . Now, F can give player i a (fake) joint commitment \tilde{X} of its choice. And it will be relative to this \tilde{X} that player i will provide its own individual signature, y_i , of (M, S) .

In previous proofs, for single-signer random-oracle signatures, answering signature queries was easy because the simulator could imitate the oracle so as to produce commitments and challenges “simultaneously.” In our security proof, however, the simulator needs to *rewind* the adversary (even though the simulator controls the random oracle). This is so because the simulator must commit to X_i before knowing what the challenge e (which may be based on a previously asked \tilde{X}) will actually be.

⁵Note that our *definition* does not require adversarial players to participate in the key generation protocol, or to have any secret keys at all. Our *construction*, however, ensures that they will both participate and know their secret keys: otherwise, their public keys will not be included in the Merkle tree, and the forgery will not be deemed valid.

Acknowledgements

The work of the second author was performed, in part, during his visit to the MIT Laboratory for Computer science. The work of the third author was performed at the MIT Laboratory for Computer Science and was supported by the National Science Foundation graduate research fellowship and by a grant from the NTT corporation.

4. REFERENCES

- [1] *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, Chicago, Illinois, 2–4 May 1988.
- [2] E. Bach. How to generate factored random numbers. *SIAM Journal on Computing*, 17(2):179–193, Apr. 1988.
- [3] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communication Security*, pages 62–73, November 1993. Revised version appears in <http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html>.
- [4] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In ACM [1], pages 1–10.
- [5] G. Brassard, editor. *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990, 20–24 Aug. 1989.
- [6] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In B. S. Kaliski Jr., editor, *Advances in Cryptology—CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer-Verlag, 17–21 Aug. 1997.
- [7] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In ACM [1], pages 11–19.
- [8] D. Chaum and E. van Heyst. Group signatures. In Davies [9], pages 257–265.
- [9] D. W. Davies, editor. *Advances in Cryptology—EUROCRYPT 91*, volume 547 of *Lecture Notes in Computer Science*. Springer-Verlag, 8–11 Apr. 1991.
- [10] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In Brassard [5], pages 307–315.
- [11] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.
- [12] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology—CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987, 11–15 Aug. 1986.
- [13] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of RSA functions. In Koblitz [20], pages 157–172.
- [14] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In [23], pp. 354–371.
- [15] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In J. Stern, editor, *Advances in Cryptology—EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310. Springer-Verlag, 2–6 May 1999.
- [16] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, 25–27 May 1987.
- [17] L. Harn. Group-oriented (t, n) threshold digital signature scheme and digital multisignature. *IEE Proc.-Comput. Digit. Tech.*, 141(5), Sept. 1994.
- [18] P. Horster, M. Michels, and H. Petersen. Meta-multisignatures schemes based on the discrete logarithm problem. In *Information Security: The Next Decade. Proceedings of the IFIP TC11 Eleventh International Conference on Information Security, IFIP/Sec '95*, pages 128–141. Chapman & Hall, 1995.
- [19] K. Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, (71):1–8, Oct. 1983.
- [20] N. Koblitz, editor. *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*. Springer-Verlag, 18–22 Aug. 1996.
- [21] S. K. Langford. Weaknesses in some threshold cryptosystems. In Koblitz [20], pages 74–82.
- [22] C.-M. Li, T. Hwang, and N.-Y. Lee. Threshold-multisignature schemes where suspected forgery implies traceability of adversarial shareholders. In A. De Santis, editor, *Advances in Cryptology—EUROCRYPT 94*, volume 950 of *Lecture Notes in Computer Science*, pages 194–204. Springer-Verlag, 1995, 9–12 May 1994.
- [23] U. Maurer, editor. *Advances in Cryptology—EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*. Springer-Verlag, 12–16 May 1996.
- [24] R. C. Merkle. A certified digital signature. In Brassard [5], pages 218–238.
- [25] S. Micali. CS proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.
- [26] M. Michels and P. Horster. On the risk of disruption in several multiparty signature schemes. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology—ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 334–345, Kyongju, Korea, 3–7 Nov. 1996. Springer-Verlag.
- [27] K. Ohta and T. Okamoto. A digital multisignature scheme based on the Fiat-Shamir scheme. In H. I. H, R. Rivest, and T. Matsumoto, editors, *Advances in Cryptology — ASIACRYPT 91*, pages 139–148. Springer-Verlag, 1993, 11–14 Nov. 1991.
- [28] K. Ohta and T. Okamoto. On concrete security treatment of signatures derived from identification. In H. Krawczyk, editor, *Advances in Cryptology—CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 354–369. Springer-Verlag, 23–27 Aug. 1998.

- [29] K. Ohta and T. Okamoto. Multi-signature schemes secure against active insider attacks. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, E82-A(1):21–31, Jan. 1999.
- [30] T. Okamoto. A digital multisignature schema using bijective public-key cryptosystems. *ACM Transaction on Computer Systems*, 6(4):432–441, Nov. 1988.
- [31] T. P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In Davies [9], pages 522–526.
- [32] D. Pointcheval and J. Stern. Security proofs for signature schemes. In Maurer [23], pages 387–398.
- [33] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

APPENDIX

A. SIMPLIFYING THE ADVERSARY

The adversary F described in Section 2 is extremely powerful. It can corrupt and attack arbitrary parties at arbitrary, adaptively determined times. The resulting notion of security is, therefore, very compelling, but is also difficult to work with when analyzing a concrete implementation of ASM schemes. We will therefore define a *weak adversary* F' that operates in a rather simple and easy to analyze manner, and prove that, despite such simplicity, an ASM scheme is secure against F' if and only if it is secure against F . In the sequel, therefore, we shall analyze the security of schemes in terms of F' rather than F .

Unlike F , the weak adversary F' does not have the ability to corrupt players. In fact, it does not even interact with a network of players: at the outset (before key generation), it has to pick one player P_i that it will be attacking. The other players then cease to exist, and F' has to provide all the inputs and network traffic for P_i . It can also see all the outputs and network traffic coming out of P_i .

After P_i generates its keys, F' is allowed to carry out an adaptive chosen-message-and-subgroup attack on P_i , just like F . The goal of F' is similar to that of F .

DEFINITION 2. *An ASM scheme is called weakly secure if, for all constants $c > 0$ and all sufficiently large security parameters k , no polynomial-time (in k) weak adversary has better than k^{-c} chance of outputting a triple (σ, M, S) such that:*

- σ is a valid signature on the message M by the subgroup S of players
- $P_i \in S$,
- P_i has not been asked by F' to execute the signing protocol on M and S .

Because the other players don't exist, in some sense, they will now necessarily be "fictitious."

Note that the weak adversary can also be seen as the strong adversary that is not allowed to adaptively corrupt players: rather, it has to pick at the outset the $L - 1$ play-

ers it will corrupt. This observation leads to the following theorem, whose proof will appear in the full version of this work.

THEOREM 2. *Assume that there exists a polynomial Q such that for any security parameter k , the size L of G is bounded by $L < Q(k)$. Then an ASM signature scheme is weakly secure if and only if it is (strongly) secure.*

B. MERKLE TREES

(The following description is taken almost verbatim from [25].) Recall that a binary tree is a tree in which every node has at most two children, hereafter called the *0-child* and the *1-child*. A *collision-free hash function* is, informally speaking, a polynomial-time computable function H mapping binary strings of arbitrary length into reasonably short ones, so that it is computationally infeasible to find any *collision* (for H), that is, any two different strings x and y for which $H(x) = H(y)$.

A *Merkle tree* [24] then is a binary tree whose nodes store values, some of which computed by means of a collision-free hash function H in a special manner. A leaf node can store any value, but each internal node should store a value that is the one-way hash of the concatenation of the values in its children.⁶ Thus, if the collision-free hash function produces k -bit outputs, each internal node of a Merkle tree, including the root, stores a k -bit value. Except for the root value, each value stored in a node of a Merkle tree is said to be a 0-value, if it is stored in a node that is the 0-child of its parent, a 1-value otherwise.

The crucial property of a Merkle tree is that, unless one succeeds in finding a collision for H , *it is computationally hard to change any value in the tree (and, in particular, a value stored in a leaf node) without also changing the root value.* This property allows a party A to "commit" to L values, v_1, \dots, v_L (for simplicity assume that L is a power of 2 and let $d = \log L$), by means of a single k -bit value. That is, A stores value v_i in the i -th leaf of a full binary tree of depth d , and uses a collision-free hash function H to build a Merkle tree, thereby obtaining a k -bit value, V , stored in the root. This root value V "implicitly defines" what the L original values were. Assume in fact that, as some point in time, A gives V , but not the original values, to another party B . Then, whenever, at a later point in time, A wants to "prove" to B what the value of, say, v_i was, he may just reveal all L original values to B , so that B can recompute the Merkle tree and the verify that the newly computed root-value indeed equals V . More interestingly, A may "prove" what v_i was by revealing just $d + 1$ (that is, just $1 + \log L$) values: v_i together with its *authenticating path*, that is, the values stored in the siblings of the nodes along the path from leaf i (included) to the root (excluded), w_1, \dots, w_d . Party B verifies the received alleged leaf-value I_i and the received alleged authenticating path w_1, \dots, w_d as follows. She sets $u_1 = v_i$ and, letting i_1, \dots, i_d be the binary expansion of i , computes the values u_2, \dots, u_d as follows: if $i_j = 0$, she sets $u_{j+1} = H(w_j u_j)$; else, she sets $u_{j+1} = H(u_j w_j)$. Finally, B checks whether the computed k -bit value u_d equals V .

⁶I.e., if an internal node has a 0-child storing the value U and a 1-child storing a value V , then it stores the value $H(UV)$.