Information and Computation **164**, 54–84 (2001)
doi:10.1006/inco.2000.2881, available online at http://www.idealibrary.com on IDEAL®

Robust Threshold DSS Signatures

Rosario Gennaro[1]

IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598
E-mail: rosario@watson.ibm.com

Stanisław Jarecki

MIT Laboratory for Computer Science, 545 Tech Square, Cambridge, Massachusetts 02139
E-mail: stasio@theory.lcs.mit.edu

Hugo Krawczyk

Department of Electrical Engineering, Technion, Haifa 32000, Israel; and
IBM T.J. Watson Research Center, P.O. Box 704,
Yorktown Heights, New York 10598
E-mail: hugo@ee.technion.ac.il

and

Tal Rabin

IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598
E-mail: talr@watson.ibm.com

We present threshold DSS (digital signature standard) signatures where the power to sign is shared by n players such that for a given parameter $t < n/2$ any subset of $2t+1$ signers can collaborate to produce a valid DSS signature on any given message, but no subset of t corrupted players can forge a signature (in particular, cannot learn the signature key). In addition, we present a robust threshold DSS scheme that can also tolerate $n/3$ payers who refuse to participate in the signature protocol. We can also endure $n/4$ maliciously faulty players that generate incorrect partial signatures at the time of signature computation. This results in a highly secure and resilient DSS signature system applicable to the protection of the secret signature key, the prevention of forgery, and increased system availability. Assuming that secret communication between the players is available, we prove the security of our protocols solely based on the hardness of forging a regular DSS signature.

[1] A preliminary version of this paper appeared in [GJKR96a].

0890-5401/01 $35.00

1. INTRODUCTION

Using a threshold signature scheme, digital signatures can be produced by a group of players rather than by one party. In contrast to the regular signature schemes where the signer is a single entity which holds the secret key, in threshold signature schemes the secret key is shared by a group of n players. In order to produce a valid signature on a given message m, individual players produce their *partial signatures* on that message and then combine them into a full signature on m. A distributed signature scheme achieves threshold $t < n$ if no coalition of t (or less) players can produce a new valid signature, even after the system has produced many signatures on different messages. A signature resulting from a threshold signature scheme is the same as if it was produced by a single signer possessing the full secret signature key. In particular, the validity of this signature can be verified by anyone who has the corresponding unique public verification key. In other words, the fact that the signature was produced in a distributed fashion is transparent to the recipient of the signature.

Threshold signatures are mainly motivated by the need to protect signature keys from the attack of internal and external adversaries, as well as by the need that arises in some organizations to have a group of employees agree on a given message (or a document) before signing it. The former motivation becomes increasingly important with the actual deployment of public key systems in practice. The signing power of some entities, (e.g., a government agency, a bank, a certification authority) inevitably invites attackers to try and "steal" this power. The goal of a threshold signature scheme is twofold: To increase the availability of the signing agency and at the same time to increase the protection against forgery by making it harder for the adversary to learn the secret signature key. Notice that in particular, the threshold approach rules out the naive solution based on traditional secret sharing, where the secret key is shared in a group but reconstructed by a *single player* each time that a signature is to be produced. Such a protocol would contradict the requirement that no t (or less) players can ever produce a new valid signature. In threshold schemes, multiple signatures are produced without an exposure or an explicit reconstruction of the secret key.

A further crucial property of threshold signatures is that the output of the signature scheme is the same as if produced in a centralized way (i.e., by a single signer that holds the whole private key). In particular, verification of the signature is independent of the way the signature generation is implemented. This is particularly important for standard signature schemes such as DSS (digital signature standard). As an example, consider public key certificates signed with DSS signatures. Applications that need to verify these certificates are programed to do this verification regardless of the particular implementation (e.g., centralized or distributed) of the signing algorithm by the issuing certification authority.

1.1. Previous Work

Threshold signatures are part of a general approach know as *threshold cryptography* which was introduced by the works of Boyd [Boy86], Croft and Harris

[CH89], Desmedt [Des88], and Desmedt and Frankel [DF90]. This approach has received considerable attention in the literature; we refer the reader to [Des94] for a survey of some of the work in this area. While the existence of polynomial-time threshold signature schemes is implied by the general results on secure distributed computing of [GMW87], the "threshold cryptography" line of research focuses on providing efficient and practical solutions for specific signature schemes. The improvement of the specific constructions, like the one presented in this paper, over the generic methods is not only on the practical aspects of the constructions but also in providing stronger analysis. This includes weakening the cryptographic assumptions as well as providing stronger security reductions.

It is particularly important to provide threshold solutions for signatures schemes used in practice, as those systems are the ones being deployed in the real world and hence they are the ones that require real protection. As of today, RSA [RSA78] and DSS [NIST91] appear as the two most used schemes in practice. For the case of RSA signatures particular examples of threshold schemes can be found in [DF91, DDFY94, FGY96, GJKR96b, Rab98].

DSS signatures turn out to be less amenable to sharing techniques than RSA or even other ElGamal-type of signatures. For this reason, many variants of ElGamal-type signatures, have been proposed that are more suitable to being turned into threshold schemes (see, for example, [Har94, PK96].) The specific case of DSS was studied by Langford in [Lan95]. Langford has overcome some of the DSS difficulties, exhibiting a solution which requires a group of $n = t^2 - t + 1$ players in order to tolerate up to t players that might refuse to participate in the signature protocol. Thus, for n given players this solution can resist up to $\sqrt{n}$ corrupted parties.[2]

An analysis of threshold techniques applied to various ElGamal-like schemes appears in an earlier paper by Cerecedo *et al.* [CMI93]. They present formal definitions of threshold signature schemes and solutions based on the ElGamal signature scheme which require only a linear increase in the number of signers (compared to quadratic as in [Lan95]). Our work, independently developed, follows an approach similar to [CMI93]. However, by concentrating on the case of DSS signatures we achieve significantly better properties in our solution and a stronger security analysis. We discuss these properties next.

1.2. Our Contribution

We present several protocols for threshold DSS signatures which enjoy several attractive properties listed below.

Provable security. Our work is the first to present a proof of security of the proposed threshold DSS schemes which can be based solely on the unforgeability

[2] Langford presents some additional schemes but of more limited applicability: a 2-out-of-n scheme that withstands up to one faulty party and a general t-out-of-n scheme that uses precomputed tables of one-time shares and that requires a higher level of trust for the generation of these tables. See [Lan95] for details.

of regular DSS signatures. Previous work [CMI93] required additional cryptographic assumptions.[3] That is, provided that secret communication between the players is available, our schemes are secure if and only if the underlying signature algorithm is secure. Clearly, this is the strongest security claim one can hope for on any threshold signature scheme. We present rigorous proofs of the equivalent security of DSS and our threshold schemes.

Efficiency. We introduce several schemes, each with a different trade-off between security (the kind of adversary and the maximal number of corrupted players tolerated) and efficiency (the number of operations required by a player in order to complete the protocol). The achieved trade-offs are superior to the ones encountered in previous works.

Flexible thresholds. In general, one would like to have higher thresholds, because they achieve increased security at a given system cost (i.e., a given number of servers). However, one should also consider the computational cost involved in increasing the threshold of a given scheme. In our work we present threshold DSS signatures schemes, where in order to tolerate t gossip-only (i.e., eavesdropping) faults we need $2t+1$ active signers during signature computation. In other words, the adversarial threshold is $t \leqslant \frac{n-1}{2}$. This threshold goes down to $t \leqslant \frac{n-1}{3}$ if we allow the possibility of t faulty servers to refuse to participate in the signature protocol (i.e., "fail-stop", or halting faults). In all these cases we improve substantially on the quadratic bound on [Lan95]. In addition, we provide a *robust threshold signature scheme* for DSS which can withstand the participation of dishonest signers during the signature computation operation. Namely, we provide a mechanism that succeeds in constructing a valid signature even if the partial signatures contributed by some of the signers are incorrect. The solution in [Lan95] for DSS does not enjoy this property. In fact, without a mechanism for detecting wrong partial signatures, one may need to try an (exponential in t) number $\binom{n}{2t+1}$ of subsets of signers before finding a subset that generates a valid DSS signature.[4] In our case, we achieve a robust threshold solution to DSS signatures tolerating t faults: that is, t or less corrupted players will not be able to forge signatures and neither will they be able to prevent the system from computing correct signatures by behaving in any arbitrary malicious way. In this case our protocol achieves fault-tolerance of $\frac{n-1}{4}$.

Assumed trust. Our schemes do not require trusting any particular party at any time, including during the initial secret key generation. This is an important property achieved by some other ElGamal-based threshold signatures schemes (including the DSS solution in [Lan95, CMI93]).

[3] Although not explicitly mentioned in that paper the only way we were able to prove the security of the protocols in [CMI93] was under an assumption which appears to be significantly stronger than the unforgeability of DSS signatures. Also their scheme seems to be simulatable only in the presence of a limited adversary. See Section 7 for more details.

[4] The robustness property has been known for some *other* shared ElGamal-like signature schemes; see [CMI93, Har94]. As for threshold RSA, robust solutions have been only recently found (see [FGY96, GJKR96b]).

Proactive signatures. Remarkably, our solutions for robust threshold DSS signatures can be *proactivized* using the recent techniques of [HJJKY97] (based on proactive secret sharing of the signature key [HJKY95]). In this way, one can keep the DSS signature key (and its public key counterpart) fixed for a long time while its shares can be refreshed periodically. An adversary that tries to break the threshold signature scheme needs to corrupt t servers *in one single period of time* (which may be as short as one day, one week, etc.), as opposed to having the whole lifetime of the key (e.g., 2 or 5 years) to do so. It is worth noting that in order to obtain a proactive solution for DSS our schemes do not need to be changed. The proactivization is obtained by adding a periodic "share refreshment phase" that is fully compatible with our threshold signature mechanisms.

1.3. Technical Overview

The threshold DSS signatures schemes need to deal with two technical difficulties. Combining shares of two secrets, a and b, into shares of the product of these secrets, ab, and producing shares for a secret a given the shares of its reciprocal a^{-1} (computations are over a prime field Z_q). We solve the first problem (sharing of a product of secrets) using a single product of polynomials (with combined degree $2t$ resulting in the need for only $2t+1$ active signers) [BGW88, B87]. For the second problem, the sharing of a reciprocal, we use a protocol due to Bar-Ilan and Beaver [BB89]. In addition to these techniques we use many tools from other works, such as verifiable secret sharing (both computational and information-theoretic versions), shared generation–distribution of secrets, rerandomization of secret shares, and more. In particular we make extended use of Pedersen's unconditionally secure VSS protocols [Ped91b] which allows us to reduce the computational assumptions needed in the proofs of our schemes. To achieve the robustness of the $t \leqslant \frac{n-1}{4}$ scheme we apply error correcting techniques due to Berlekamp and Welch [BW]. We prove the security of our schemes assuming the infeasibility of forging DSS signatures.

1.4. Organization

Section 2 introduces the model and definitions for threshold signatures and their security. Section 3 recalls the DSS signature scheme. Section 4 describes some of the existing tools in the literature that we use in our solutions. Section 5 shows how to jointly and securely generate the initial DSS private key without the need of a trusted party. Sections 6, 7, and 8 present our secure threshold DSS signatures. Finally, Section 9 discusses the efficiency of our schemes.

2. MODEL AND DEFINITIONS

In this section we introduce our communication model and provide definitions of secure threshold signature schemes. Related definitions can be found in [CMI93].

Communication model. We assume that our computation model is composed of a set of n *players* $\{P_1, ..., P_n\}$ who can be modeled by polynomial-time randomized

Turing machines. They are connected by a complete network of private (i.e., untappable) point-to-point channels. In addition, the players have access to a dedicated broadcast channel; by dedicated we mean that if player P_i broadcast a message, it is received by every other player and recognized as coming from P_i. These assumptions (privacy of the communication channels and dedication of the broadcast channel) allow us to focus on a high-level description of the protocols. The privacy of the point-to-point channels can be implemented with standard cryptographic techniques, while the dedicated broadcast can be implemented over point-to-point channels with a Byzantine agreement protocol.

We assume that the communication channels provide a *partially synchronous* message delivery. In the design of the distributed cryptographic protocols it is often assumed that the message delivery is fully synchronous. This assumption is unrealistic in many cases where only partially synchronous message delivery is provided (e.g., the Internet). By partially synchronous communication model we mean that the messages sent on either a point-to-point or the broadcast channel are received by their recipients within some fixed time bound. A failure of a communication channel to deliver a message within this time bound can be treated as a failure of the sending player. While messages arrive in this partially synchronous manner, the protocol as a whole proceeds in synchronized rounds of communication; i.e., the honest players start a given round of a protocol at the same time. To guarantee such round synchronization, and for simplicity of discussion, we assume that the players are equipped with synchronized clocks.

Notice that in a partially synchronous communication model all messages can still be delivered relatively fast, in which case, in every round of communication, the malicious adversary can wait for the messages of the uncorrupted players to arrive, then decide on his computation and communication for that round, and still get his messages delivered to the honest parties on time. Therefore we should always assume the worst case that the adversary speaks last in every communication round. In the cryptographic protocols literature this is also known as a *rushing* adversary.

The adversary. We assume that an adversary, $\mathcal{A}$, can corrupt up to t of the n players in the network. We distinguish between three kinds of (increasingly powerful) adversaries:

- An *eavesdropping adversary* learns all the information stored at the corrupted nodes and hears all the broadcasted messages.
- A *halting adversary* is an eavesdropping adversary that at the beginning of each round may *also* cause corrupted players to stop sending messages for the remaining part of the protocol (e.g., by crashing or disconnecting a machine).[5]

[5] We are assuming that rounds are *atomic*, i.e., that a player can crash only at the beginning of each round. We do this just for the sake of simplicity. Indeed, the case of players that crash in the middle of a round (after they sent messages to some players but not to others) can be easily reduced to the previous one. Just add one round of acknowledgments of received messages. If from those acknowledgments it appears that a player crashed in the middle of a round, simply consider him dead for the whole round. Another approach would be to consider this kind of adversary as a "malicious" one and apply our solutions form Section 7 at the expense of added protocol and computational complexity.

• A *malicious adversary* is an eavesdropping adversary that may *also* cause corrupted players to divert from the specified protocol in *any* (possible malicious) way.

We assume that the computational power of the adversary is adequately modeled by a probabilistic polynomial time Turing machine. (In fact, it suffices for our results to assume that the adversary cannot forge regular DSS signatures, which, in turn, implies the infeasibility of computing discrete logarithms.)

Adversaries can also be categorized as *static* or *adaptive*. A static adversary chooses the corrupted players at the beginning of the protocol, while an adaptive one chooses them during the computation. In the following we will consider only static adversaries. Recently it was proven in [C+99] that by making small adjustments to the protocols in this paper, it is possible to tolerate adaptive adversaries.

Given a protocol $\mathscr{P}$ the *view* of the adversary, denoted by $\mathscr{VIEW}_{\mathscr{A}}(\mathscr{P})$ is defined as the probability distribution (induced by the random coins of the players) on the knowledge of the adversary, namely, the computational and memory history of all the corrupted players and the public communications and output of the protocol.

Singature scheme. A signature scheme $\mathscr{S}$ is a triple of efficient randomized algorithms (Key-Gen, Sig, Ver). Key-Gen is the *key generator* algorithm: on input a random string, it outputs a pair (y, x), such that y is the *public key* and x is the *secret key* of the signature scheme. Sig is the *signing* algorithm: on input a message m and the secrete key x, it outputs *sig*, a signature of the message m. Since Sig can be a randomized algorithm there might be several valid signatures *sig* of a message m under the key x; with $\mathsf{Sig}(m, x)$ we will denote the set of such signatures. Ver is the *verification* algorithm. On input a message m, the public key y, and a string *sig*, it checks whether *sig* is a proper signature of m, i.e., if $sig \in \mathsf{Sig}(m, x)$.

The notion of security for signature schemes was formally defined in [GMR88] in various flavors. The following definition captures the strongest of these notions: existential unforgeability against adaptively chosen message attack.

DEFINITION 1. We say that a signature scheme $\mathscr{S} = (\mathsf{Key\text{-}Gen}, \mathsf{Sig}, \mathsf{Ver})$ is unforgeable if no adversary who is given the public key y generated by Key-Gen, and the signatures of k messages $m_1, ..., m_k$ adaptively chosen, can produce the signature on a new message m (i.e., $m \notin \{m_1, ..., m_k\}$) with nonnegligible probability.

Threshold secret sharing. Given a secret value s we say that the values $(s_1, ..., s_n)$ constitute a (t, n)-threshold secret sharing of s if t (or less) of these values reveal no information about s and if there is an efficient algorithm that outputs s having $t+1$ of the values s_i as inputs.

Threshold signature schemes. Let $\mathscr{S} = (\mathsf{Key\text{-}Gen}, \mathsf{Sig}, \mathsf{Ver})$ be a signature scheme. A (t, n)-threshold signature scheme $\mathscr{TS}$ for $\mathscr{S}$ is a pair of protocols (Thresh-Key-Gen, Thresh-Sig) for the set of players $\{P_1, ..., P_n\}$.

Thresh-Key-Gen is a distributed key generation protocol used by the players to jointly generate a pair (y, x) of public–private keys. At the end of the protocol the

private output of player P_i is a value x_i such that the values $(x_1, ..., x_n)$ form a (t, n)-threshold secret sharing of x. The public output of the protocol contains the public key y. Pairs (y, x) of public–secret key pairs are produced by Thresh-Key-Gen with the same probability distribution as if they were generated by the Key-Gen protocol of the regular signature scheme $\mathcal{S}$.

Thresh-Sig is the distributed signature protocol. The private input of P_i is the value x_i. The public inputs consists of a message m and the public key y. The output of the protocol is a value $sig \in \mathsf{Sig}(m, x)$. *The verification algorithm is, therefore, the same as in the regular signature scheme* $\mathcal{S}$.

Secure threshold signature schemes. Our definition of security includes both *unforgeability* and *robustness*.

DEFINITION 2. We say that a (t, n)-threshold signature scheme $\mathcal{TS} = (\mathsf{Thresh\text{-}Key\text{-}Gen}, \mathsf{Thresh\text{-}Sig})$ is *unforgeable*, if no malicious adversary who corrupts at most t players can produce, with nonnegligible probability, the signature on any new (i.e., previously unsigned) message m, given the view of the protocol Thresh-Key-Gen and of the protocol Thresh-Sig on input messages $m_1, ..., m_k$ which the adversary adaptively chose.

This is analogous to the notion of existential unforgeability under chosen message attack as defined by Goldwasser *et al.* [GMR88]. Notice that now the adversary does not just see the signatures of k messages adaptively chosen, but also the internal state of the corrupted players and the public communication of the protocols. Following [GMR88] one can also define weaker notions of unforgeability.

In order to prove unforgeability we use the concept of *simulatable adversary view* [GMR89, MR92, B92]. Intuitively, this means that the adversary who sees all the information of the corrupted players and the signature of m could generate by itself all the other public information produced by the protocol Thresh-Sig. This ensures that the run of the protocol provides no useful information to the adversary other than the final signature on m.

DEFINITION 3. A threshold signature scheme $\mathcal{TS} = (\mathsf{Thresh\text{-}Key\text{-}Gen}, \mathsf{Thresh\text{-}Sig})$ is *simulatable* if the following properties hold:

1. The protocol Thresh-Key-Gen is simulatable. That is, there exists a simulator SIM_1 that, on input the public key y and the public output generated by an execution of Thresh-Key-Gen, can simulate the view of the adversary on that execution.

2. The protocol Thresh-Sig is simulatable. That is, there exists a simulator SIM_2 that, on input the public input of Thresh-Sig (in particular the public key y and the message m), t shares $x_{i_1}, ..., x_{i_t}$, and the signature sig of m, can simulate the view of the adversary on an execution of Thresh-Sig that generates sig as an output.

This is actually a stronger property than Definition 2. Indeed it would be enough for us to say that the executions of the protocols Thresh-Key-Gen and Thresh-Sig give the adversary no advantage in forging signatures for the scheme $\mathcal{S}$. In other words, we could allow the adversary to gain knowledge provided that such knowledge is useless for forging.

However, our stronger definition subsumes this specific goal and provides a proof of security for any of the "flavors" of signature security as listed in [GMR88]. Indeed one can prove that if the underlying signature scheme $\mathcal{S}$ is unforgeable (in any of the flavors of [GMR88]) and $\mathcal{TS}$ is simulatable then $\mathcal{TS}$ is unforgeable (with the same flavor as $\mathcal{S}$). Another important aspect of proving the security of our signature schemes through the stronger Definition 3 is that it allows us to maintain security when composing several of these protocols or composing it with other protocols, such as proactive secret sharing. This forms an essential ingredient in the proactivization of our protocols in the framework of [HJJKY97].

Finally, we need the important notion of *robustness*. Robustness means that the protocol will compute a *correct output* even in the presence of halting or malicious faults. We will talk about (h, c, n)-*robustness* to indicate that the adversary is allowed to halt up to h players and corrupt maliciously up to c players ($h + c \leqslant t$ where t is total number of corrupted players).

DEFINITION 4. A threshold signature scheme $\mathcal{TS} = (\text{Thresh-Key-Gen}, \text{Thresh-Sig})$ is (h, c, n)-robust if in a group of n players, even in the presence of an adversary who halts h players and corrupts maliciously c players, both Thresh-Key-Gen and Thresh-Sig complete successfully.

3. THE DIGITAL SIGNATURE STANDARD

The digital signature standard (DSS) [NIST91] is a signature scheme based on the ElGamal [ElG85] and Schnorr [Sch91] signature schemes, which was adopted as the U.S. standard digital signature algorithm. In our description of the DSS protocol we follow the notation introduced in [Lan95], which differs from the original presentation of [NIST91] by switching k and k^{-1}. This change will allow a clearer presentation of our threshold DSS signature protocols.

Key generation. A DSS key is composed of public information p, q, g, a public key y, and a secret key x, where:

1. p is a prime number of length l, where l is a multiple of 64 and $512 \leqslant l \leqslant 1024$.
2. q is a 160-bit prime divisor of $p - 1$.
3. g is an element of order q in Z_p^*. The triple (p, q, g) is public.
4. x is the secret key of the signer, a random number $1 \leqslant x < q$.
5. $y = g^x \bmod p$ is the public verification key.

SIGNATURE ALGORITHM. Let M be the message to be signed. The message is first hashed using the hash function SHA-1; let m be the resulting hash value. The signer picks a random number k such that $1 \leqslant k < q$ calculates $k^{-1} \bmod q$ and sets

$$r = (g^{k^{-1}} \bmod p) \bmod q$$

$$s = k(m + xr) \bmod q.$$

The pair (r, s) is a signature of m.

VERIFICATION ALGORITHM. A signature (r, s) of a message M can be publicly verified by first computing the SHA-1 hash m of M and then by checking that $r = (g^{ms^{-1}} y^{rs^{-1}} \bmod p) \bmod q$ where s^{-1} is computed modulo q.

Important notational convention. In the following when we refer to the "message" m it is to be intended as the SHA-1 hash of the original message M. It is indeed known that without this hashing step DSS would be existentially forgeable.

DSS assumption. The DSS signature scheme is unforgeable according to Definition 1.

Remark. Later in the simulation of our protocols we will need the value $r^* = g^{k^{-1}} \bmod p$ that is the value r *before* the reduction mod q. We note that such a value is easily computable from a regular DSS signature pair since

$$r^* = g^{ms^{-1}} y^{rs^{-1}} \bmod p.$$

This basically implies that the purpose of the extra reduction mod q is to shorten the DSS signature, but it serves no security purpose.

4. BASIC TOOLS

Here we recall a few existing techniques that we use in our solutions.

4.1. Shamir's Secret Sharing

In Shamir's secret sharing protocol [Sha79] (we will refer to it as Shamir-SS), a dealer shares a secret among n players $P_1, ..., P_n$ in the following way. Given a number $t < n$, a prime q, and a secret $\sigma \in Z_q$, the dealer chooses at random a polynomial $f(z)$ over Z_q of degree t such that $f(0) = \sigma$. It then secretly transmits to each player P_i a share $\sigma_i \triangleq f(i)$. (We use the interpolation values $i = 1, 2, ..., n$ for simplicity; any values in Z_q can be used as well.) This protocol generates no public output. It can tolerate t eavesdropping faults if $n \geqslant t + 1$. It can also tolerate t halting faults if $n \geqslant 2t + 1$. In the following we will write

$$(\sigma_1, ..., \sigma_n) \xleftrightarrow{(t, n)} \sigma \bmod q$$

to denote such a sharing.

We note that by using error-correcting techniques the protocol can be modified to tolerate t malicious faults (among the players, excluding the dealer) if $n \geq 3t+1$ [MS81] (i.e., the secret can still be reconstructed in the presence of $t < n/3$ corrupted shares).

4.2. Feldman's Verifiable Secret Sharing

Feldman's verifiable secret protocol [Fel87], denoted here by Exp-VSS, extends Shamir's secret sharing method in a way that allows the recipients of shares to verify that their shares are consistent (i.e., that any subset of $t+1$ shares determines the same unique secret). The protocol can tolerate up to $\frac{n-1}{2}$ malicious faults *including the dealer*. For this purpose, two large primes p and q and an element $g \in Z_p^*$ are chosen such that q divides $p-1$ and g is an element of Z_p^* of order q.[6] Like a Shamir's scheme, the dealer generates a random t-degree polynomial $f(z)$ over Z_q, s.t. $f(0)=\sigma$, and transmits to each player P_i a share $\sigma_i = f(i)$. The dealer also broadcasts values $y_j = g^{a_j} \bmod p$, where $f(z) = \sum_j a_j z^j$. This will allow the players to check that the values σ_i really define a secret by checking that

$$g^{\sigma_i} = \prod_j (y_j)^{i^j} \bmod p. \tag{1}$$

If the above equation is not satisfied player P_i asks the dealer to reveal his share (we call this a *complaint*). If more than t players complain then the dealer is clearly bad and he is disqualified. Otherwise the dealer reveals the share σ_i matching Eq. (1) for each complaining player P_i.

Equation (1) also allows detection of incorrect shares σ_i' at reconstruction time. Notice that the value of the secret is only computationally secure, e.g., the value $g^{a_0} = g^{\sigma} \bmod p$ is leaked. However, it can be shown that an adversary that learns t or less shares cannot obtain any information on σ beyond what can be derived from g^{σ}. The proof of this fact uses a simulation argument which we sketch here. Given any t (or less) shares (known to the adversary) and g^{σ}, one can generate a distribution of the other public information in the protocol as follows. Assume the known shares are $\sigma_1, ..., \sigma_t$. Thus we know g^{σ_i}, $i=1, ..., t$, as well as $g^{\sigma_0} = g^{\sigma}$. This allows us to compute g^{σ_i} for $i > t$ using the equation $g^{\sigma_i} = \prod_{j=0}^{t} (g^{\sigma_j})^{\lambda_{ij}}$ where λ_{ij} are the Lagrange interpolation coefficients (i.e., such that for $i > t$ $\sigma_i = \sum_{j=0}^{t} \lambda_{ij} \sigma_j$). In the following, we refer to the above way of computing g^{σ_i} as interpolation in the exponent.

4.3. Unconditionally Secure Verifiable Secret Sharing

Here we recall a verifiable secret sharing protocol that provides information theoretic secrecy for the shared secret. This is in contrast to Feldman's VSS protocol which leaks the value of $g^{\sigma} \bmod p$. The stronger secrecy property is

[6] In the applications in this paper we will typically use the values p, q, g as defined by the DSS scheme (Section 3).

required by some of our applications and security analysis. Two known implementations, with slightly different properties, of such a VSS protocol are known in the literature. One, by Feldman and Micali [FM88], is based on a bivariate polynomial sharing and can tolerate up to $\frac{n-1}{3}$ malicious faults. The other, by Pedersen [Ped91b], withstands up to $\frac{n-1}{2}$ faults and we use it in our applications. We describe this protocol next and will refer to it as Uncond-Secure-VSS.

Pedersen's VSS uses the parameters p, q, g as defined for Feldman's VSS. In addition, it uses an element $h \in Z_p^*$ such that h belongs to the subgroup generated by g and the discrete log of h in base g is unknown (and assumed hard to compute). The dealer first chooses two t-degree polynomials $f(z) = \sum_j a_j z^j$ and $f'(z) = \sum_j b_j z^j$ with random coefficients subject to $f(0) = \sigma$ and sends to each player P_i the values $\sigma_i = f(i)$ and $\tau_i = f'(i) \bmod q$. The dealer then commits to each coefficient of the polynomials f and f' by publishing the values $A_j = g^{a_j} h^{b_j} \bmod p$. This allows the players to verify the received shares by checking that

$$g^{\sigma_i} h^{\tau_i} = \prod_j (A_j)^{i^j} \bmod p. \tag{2}$$

As in Feldman's VSS the players who hold shares that do not satisfy the above equation broadcast a complaint. If more than t players complain the dealer is disqualified. Otherwise the dealer broadcasts the values σ_i and τ_i matching the above equation for each complaining player P_i.

At reconstruction time the players are required to reveal both σ_i and τ_i and Eq. (2) is used to validate the shares. Indeed in order to have an incorrect share σ_i' accepted at reconstruction time, player P_i has to compute the discrete log of h in base g.

Notice that the value of the secret is unconditionally protected since the only value revealed is $A_0 = g^{\sigma} h^{b_0}$ (it can be seen that for any value σ' there is exactly one value b_0' such that $A_0 = g^{\sigma'} h^{b_0'} \bmod p$ and thus A_0 gives no information on σ). This scheme is robust against $\frac{n-1}{2}$ malicious faults (with negligible probability of error) provided that computing discrete logarithm in Z_p^* is hard. As DSS already embeds this assumption, this protocol can be used in our protocol without introducing any new assumptions.

4.4. Joint Random Secret Sharing

In a joint random secret sharing [IS 90, Ped91b, Ped91c] scheme the players *collectively* choose shares corresponding to a (t, n)-secret sharing of a *random* value. At the end of such a protocol each player P_i has a share σ_i, where $(\sigma_1, ..., \sigma_n) \xleftarrow{(t,n)} \sigma$ and σ is uniformly distributed over the interpolation field. As with a regular (t, n)-secret sharing scheme the value σ is kept secret from any coalition of t players. In general, to realize a joint random secret sharing each player acts as dealer of a random local secret that he chooses, and then the final share σ_i of player P_i is computed as the sum of the shares dealt to P_i by each player. Consequently, the joint secret equals the sum of all well-dealt secrets.

Joint-Shamir-RSS

1. Each P_i chooses at random a polynomial $f_i(z)$ over Z_q of degree t. Let $a_i = f_i(0)$. P_i distributes to every player P_j a share $\sigma_{ij} \triangleq f_i(j)$.
2. Each P_j computes its final share as $\sigma_j = \sum_{i \in \{1 \ldots n\}} \sigma_{ij} \bmod q$ (where $\sigma_{ij} = 0$ if P_j did not receive anything from P_i). Then $(\sigma_1, \ldots, \sigma_n) \overset{(t,n)}{\longleftrightarrow} \sigma \bmod q$, where σ is the shared secret.

FIG. 1. Joint Shamir random secret sharing.

The realization of such a protocol depends on the type of attacker that one assumes. Against a halting adversary it suffices for each player to deal its random local value using Shamir-SS. The resulting protocol, called Joint-Shamir-RSS, achieves a sharing of a uniformly distributed secret σ, even in the presence of a halting adversary who can compromise up to t players. In Fig. 1 we present Joint-Shamir-RSS which is resistant to a halting adversary only if the players crash at the beginning of each round. Indeed, if a player crashed in the middle of a round while distributing its shares, the honest players will have an inconsistent view of this sharing. As we said in footnote 4 in Section 2, if the assumption that players crash only at the beginning of a round is to be relaxed, this problem can be easily fixed by adding one round of acknowledgment messages on the broadcast channel following Step 1 of Fig. 1.

Against a malicious adversary we will use the protocol Joint-Uncond-Secure-RSS presented in Fig. 2. It differs from Joint-Shamir-RSS in that the local secret a_{i0} is shared by player P_i using the protocol Uncond-Secure-VSS in place of Shamir-SS. This enables disqualification of dishonest players who deal inconsistent shares. The protocol achieves an unconditionally secure secret sharing of a uniformly distributed secret σ even in the presence of a malicious adversary who can compromise up to t players.

In our DSS application (e.g., see Section 5) we are going to need a joint random secret sharing where the value σ is uniformly distributed but also *the value*

Joint-Uncond-Secure-RSS

1. Player P_i generates $2t + 2$ random numbers in Z_q, $a_{i0}, a_{i1}, \ldots, a_{it}$ and $b_{i0}, b_{i1}, \ldots, b_{it}$. Consider the two polynomials $f_i(z) = \sum_{k=0}^{t} a_{ik} z^k$ and $f_i'(z) = \sum_{k=0}^{t} b_{ik} z^k$ modulo q. Player P_i does the following:
   (a) sends to player P_j the values $\sigma_{ij} = f_i(j) \bmod q$ and $\rho_{ij} = f_i'(j) \bmod q$.
   (b) computes the values $A_{ik} = g^{a_{ik}} h^{b_{ik}} \bmod p$ $(k = 0, \ldots, t)$. Broadcasts $\{A_{ik}\}_{k \in \{0 \ldots t\}}$.
2. Player P_i received the values σ_{ji} and ρ_{ji} and checks the following equations (for each $j = 1, \ldots, n$):

$$g^{\sigma_{ji}} h^{\rho_{ji}} = \prod_{k=0}^{t} (A_{jk})^{i^k} \bmod p \tag{3}$$

   If for some j the equation is not satisfied P_i broadcasts the value $COMPLAIN_j$.
3. If more than t players broadcast $COMPLAIN_j$, then player P_j is disqualified. Otherwise, player P_j reveals the values σ_{ji} and ρ_{ji} for all the players P_i that broadcasted $COMPLAIN_j$. Equation (3) is verified by everybody for the broadcasted values. If it is not satisfied, player P_j is disqualified.
4. Let *Good* be the set of players that are not disqualified. Each player P_i computes its share (σ_i, ρ_i): $\sigma_i = \sum_{P_j \in Good} \sigma_{ji} \bmod q$, $\rho_i = \sum_{P_j \in Good} \rho_{ji} \bmod q$ and all players compute the verification information used for secret reconstruction: $A_k = \prod_{P_j \in Good} A_{jk} \bmod p$, $k = 0, \ldots, t$.
   (The shared secret σ is defined as $\sigma = \sum_{j \in Good} a_{j0} \bmod q$)

FIG. 2. Joint unconditionally secure random secret sharing.

Joint-Exp-RSS

1. Run Joint-Uncond-Secure-RSS from Figure 2.
2. Let *Good* be the set of players that are not disqualified. If $P_i \in Good$ then P_i broadcasts values $y_{ik} = g^{a_{ik}} \bmod p$ for $k = 0, \ldots, t$.
3. Player P_i checks the following equation (for each j, such that $P_j \in Good$):

$$g^{\sigma_{ji}} = \prod_{k=0}^{t} (y_{jk})^{i^k} \bmod p \tag{4}$$

   If the check fails for an index j, P_i *complains* against P_j by broadcasting the values σ_{ij}, ρ_{ij} that satisfy Eq. 3 but do not satisfy Eq. 4.
4. For players P_i who receive at least one valid complaint, i.e. values which satisfy Equation (3) and not Equation (4), the other players run the reconstruction phase of Uncond-Secure-VSS to compute polynomial $f_i(z) = \sum_{k=0}^{t} a_{ik} z^k$ and values $y_{ik} = g^{a_{ik}} \bmod p$, $k = 0, \ldots, t$ in the clear.
5. All players compute the verification information used for secret reconstruction: $y_k = \prod_{P_j \in Good} y_{jk} \bmod p$, for $k \in \{0, \ldots, t\}$.

   (The shared secret σ is implicitly defined as $\sigma = \sum_{j \in Good} a_{j0} \bmod q$. Note that $y_0 = g^{\sigma} \bmod p$ is a public output of this protocol.)

FIG. 3. Joint computationally secure random secret sharing.

$g^{\sigma} \bmod p$ *is public*. It would seem that the simplest way to do this is to run a joint version of Feldman's VSS, namely, each player shares a random value a_i via Exp-VSS and then the secret is taken to be the sum of the properly dealt values. However, in the partially synchronous communication model (as we consider here), this is insecure since Exp-VSS reveals $g^{a_i} \bmod p$, and hence the adversary might wait till he sees the g^{a_i} values corresponding to the random values of the good players and base the values he will share on that information. Thus the sum of the shared values might not be random. For example, it is easy to show that the adversary can control the last bit of the resultant secret.[7] In the preliminary version [GJKR96a] of this paper we relied on this insecure straightforward implementation of this subprotocol. Here we correct it with the joint Feldman-like RSS protocol Joint-Exp-RSS introduced in [GJKR99] (there is a called DKG to highlight its role as distributed key generation) and shown in Fig. 3. It is proven in [GJKR99] that even in the presence of a malicious adversary who corrupts up to $(n-1)/2$ servers, an execution of Joint-Exp-RSS gives the honest players shares σ_i of a *uniformly distributed* secret σ. Furthermore, value g^{σ} is public, and the adversary does not learn anything more about σ than is revealed by g^{σ}.

For full details, motivation, and proofs of the Joint-Exp-RSS protocol the reader should consult [GJKR99]. Here we describe the protocol and some of its main properties as needed later in the description and analysis of our threshold DSS solutions. Joint-Exp-RSS (shown in Fig. 3) works as follows. The players first perform Joint-Uncond-Secure-RSS (Fig. 2). This ensures that the value σ is uniformly distributed since the contribution of the bad players (including their decision of being disqualified or not) is independent from the values of the good players. Afterward each player performs a Feldman "add on" to the previous

[7] In fact, [GJKR99] shows that this sample protocol and its various variants are insecure even in the much stronger *fully synchronous* communication model (i.e., a nonrushing adversary model).

unconditionally secure VSS (using the same sharing polynomial) just for the purpose of collectively computing g^{σ}. If a bad player fails this step then his contribution is publicly reconstructed from the information dealt during the Joint-Uncond-Secure-RSS part of the protocol and thus the corresponding value can be incorporated in the final share calculation.

We stress the central role of information-theoretic commitments (as used in Joint-Uncond-Secure-RSS) to achieve security against a rushing adversary; due to the strong secrecy guarantee of these commitments the attacker has no way to make his own decisions based on the committed data. See [GJKR99] for a full proof of security of Joint-Exp-RSS against such attackers.

Here we stress two main properties of the protocol that will be useful when proving our full threshold DSS scheme.

1. Joint-Exp-RSS produces an output σ which is random and uniformly distributed in Z_q. Consequently $y = g^{\sigma}$ will be random and uniformly distributed in the subgroup generated by g. Informally, this is the case because the value σ is determined in Joint-Uncond-Secure-RSS performed by the players in Step 1 (Fig. 3), when the set *Good* is fixed. Since at this point each player's contribution to σ has been shared with unconditional (information-theoretic) secrecy, the adversary's contribution cannot be "related" to the contribution of the honest players. Therefore σ is uniformly distributed in Z_q.

The reason that σ is fixed at the end of Step 1 is that once a player is inside the set *Good*, its contribution is entered in the computation of σ even if the player starts misbehaving in the rest of the protocol. This can be done since the misbehaving player's contribution can be recovered via reconstruction of the Uncond-Secure-VSS (Step 4).

2. It is possible to simulate the view of the adversary of an execution of Joint-Exp-RSS that results in a *specific* public value y_0. In other words, there exists a simulator Sim-Exp-RSS that, on any input y, is able to simulate an execution of Joint-Exp-RSS that looks indistinguishable to the adversary from an execution of a real Joint-Exp-RSS that outputs y as its public output y_0. The simulator is shown in Fig. 4 and it is taken from [GJKR99]. We assume w.l.o.g. that the adversary corrupted the first t players $P_1, ..., P_t$. The simulator works by running the correct protocol for all the honest players except one (say P_n). For this player the simulator produces a simulated execution of the Feldman add-on. That

Sim-Exp-RSS

Input: A value $y \in Z_p^*$

1. Run Joint-Uncond-Secure-RSS from Figure 2 on behalf of the honest players. Let $\hat{f}_i(z) = \sum_{k=0}^{t} \hat{a}_{ik} z^k$, $i = 1, ..., n$, be the secret-sharing polynomial dealt by P_i. Denote by $\hat{\sigma}_i$ the final share of player P_i. Notice that *all* these values are known to the simulator.
2. Compute $\hat{y}_{ik} = g^{\hat{a}_{ik}} \bmod p$ for $i = 1, \ldots, n-1$ and $k = 0, \ldots, t$.
   For player P_n do the following: Set $\hat{y}_{n0} = y \cdot \prod_{i=1}^{n-1} (y_{i0})^{-1} \bmod p$. Compute $\hat{y}_{nk} = y^{\lambda_{k,0}} \prod_{i=1}^{t} (g^{\hat{\sigma}_{ni}})^{\lambda_{k,i}} \bmod p$ for $k = 1, ..., t$, where $\lambda_{k,i}$'s are known coefficients.
   For each player P_i, $i = t+1, \ldots, n$ broadcast $\hat{y}_{ik}$ for $k = 0, \ldots, t$.
3. Follow the instructions of the protocol for the honest players.
4. Follow the instructions of the protocol for the honest players. Notice that the simulator can answer complaints produced by the adversary.
5. Follow the instructions of the protocol for the honest players.

FIG. 4. Simulator for Joint-Exp-RSS.

is, from the t shares held by the adversary and the value y that the simulator is required to hit, it computes all the reaming public information by interpolation in the exponent.

Intuitively, this is a good simulation because in the real protocol $\mathscr{A}$ receives t shares $\sigma_1, ..., \sigma_t$ of a proper sharing. It also receives the public information $y_0, ..., y_t$. Clearly the values $\sigma_1, ..., \sigma_t$ and $\hat{\sigma}_1, ..., \hat{\sigma}_t$ are identically distributed. The public information is also random and uniformly distributed except that $\hat{y}_0 = y$, which is required by the simulation.

4.5. Joint Zero Secret Sharing

This protocol generates a *collective* sharing of a secret whose value is zero. Such a protocol is similar to the above joint random secret sharing protocols but instead of local random secrets each player deals a sharing of the value zero.

Specifically, if no verifiability is required, we will use a protocol Joint-Shamir-ZSS, which is a version of Joint-Shamir-RSS from Fig. 1, except that in Step (1), each player picks a random t-degree polynomial $f_i(z)$ over Z_p *subject to a constraint* that $f_i(0)[=a_i]=0$.

When verifiability is required we use a protocol denoted Joint-Uncond-Secure-ZSS, which follows Joint-Uncond-Secure-RSS (see Fig. 2) with the following modifications:

Step 1. Each P_i picks only $2t$ random variables, because a_{i0} and b_{i0} are fixed as $a_{i0}=b_{i0}=0$. Consequently, values A_{ik} are computed only for $k=1, ..., t$.

Step 2. Equation (3) is modifies so that the product is computed over k ranging from 1 to t rather than from 0 to t.

Notice that by adding such zero-shares to existing shares of some secret σ, one obtains a randomization of the shares of σ without changing the secret. This is the way we will typically use the Joint-Shamir-ZSS and Joint-Uncond-Secure-ZSS protocols.

4.6. Computing Reciprocals

In the distributed DSS protocol we are faced with the following problem. Given a secret $k \bmod q$ which is shared among players $P_1, ..., P_n$, generate a sharing of the value $k^{-1} \bmod q$, without revealing information on k and k^{-1}. The solution described below is due to Bar-Ilan and Beaver [BB89].

Each player P_i holds a share k_i corresponding to a (t, n) Shamir secret sharing of k, namely, $(k_1, ..., k_n) \overset{(t,n)}{\longleftrightarrow} k$. The computation of shares for k^{-1} is accomplished as follows.

1. The players jointly generate a (t, n) sharing of a *random* element $a \in Z_q$ using a Joint-Shamir-RSS protocol. Denote the resulting shares by $a_1, a_2, ..., a_n$, i.e., $(a_1, ..., a_n) \overset{(t,n)}{\longleftrightarrow} a$.

2. The players execute a $(2t, n)$ Joint-Shamir-ZSS protocol after which each player P_i holds a share b_i of the secret 0. (The implicit interpolation polynomial is of degree $2t$).

3. The players reconstruct the value $\mu = ka$ by broadcasting the values $k_i a_i + b_i$ and interpolating the corresponding $2t$-degree polynomial.

4. Each player computes his or her share u_i of k^{-1} by setting $u_i \triangleq \mu^{-1} a_i \bmod q$.

We refer to the above protocol as the Reciprocal Protocol. In [BB89] it is proven that such a protocol is secure against an eavesdropping ($n \geqslant 2t+1$) or halting ($n \geqslant 3t+1$) adversary; i.e., it correctly computes a sharing of $k^{-1} \bmod q$ and reveals no extra information (e.g., is simulatable). Intuitively, the value μ revealed in the protocol gives no information on k since μ is the product of k with a random element a.

If robustness against a malicious adversary is required one should replace Joint-Shamir-RSS with Joint-Uncond-Secure-RSS or Joint-Exp-RSS and Joint-Shamir-ZSS with Joint-Uncond-Secure-ZSS.

4.7. Multiplication of Two Secrets

Given two secrets u and v, which are both shared among the players, compute the product uv, while maintaining both of the original values secret (aside from the obvious information which is revealed from the result).

Given that u and v are each shared by a polynomial of degree t, each player can locally multiply his shares of u and v, and the result will be a share of uv on a polynomial of degree $2t$. Consequently, the value uv can still be reconstructed from a set of $2t+1$ correct shares. An additional rerandomization procedure (using a joint zero secret sharing protocol) is required to protect the secrecy of the multiplied secret. This randomization is essential because a polynomial of degree $2t$ which is a product of two polynomials of degree t is not a random polynomial and would expose information about u and v.

We note that this solution to the problem of secret multiplication is a simplified version of the protocols presented in [BGW88, CCD88]. (In contrast to those works, in our case secrets are multiplied only once, thus saving most of the complexity of the solutions in the above works which mainly deal with the problem of repetitive multiplication.) The idea of avoiding complicated degree-reduction steps when there is only one multiplication to perform appears also in [B87].

5. DSS THRESHOLD KEY-GENERATION WITHOUT A TRUSTED PARTY

An instance (p, q, g) of DSS can be generated using a public procedure (e.g., as specified in [NIST91]) or using randomness which is jointly provided by the players. To generate a pair of public and private keys in a distributed setting *without a trusted party*, we use a *joint verifiable secret sharing* protocol.

In the presence of an eavesdropping or halting adversary, the players run Joint-Shamir-RSS (Fig. 1) to create a random Shamir secret sharing $(x_1, ..., x_n) \xleftarrow{(t,n)} x$, followed by an extra step when each player P_i publishes $y_i = g^{x_i} \bmod p$ and the players publicly interpolate $y = g^x$ "in the exponent," as explained at the end of Section 4.2. Namely, if G is a group of $t+1$ players that published their values y_i (note that some players might halt and fail to submit their y_i's), then each player

computes $y = \prod_{i \in G} y_i^{\lambda_i}$ where λ_i's are the Lagrange interpolation coefficients for the group of indices G (i.e., numbers such that $x = \sum_{i \in G} \lambda_i x_i \bmod q$).

In the presence of a malicious adversary it is necessary to run Joint-Exp-RSS (Fig. 3). The output of both protocols is a Shamir secret sharing $(x_1, ..., x_n) \xleftarrow{(t, n)} x \bmod q$ of a random value x and a public value $y = g^x \bmod p$. The pair (y, x) is taken to be the public–private key pair.

6. DSS-THRESH-SIG-1: EAVESDROPPING AND HALTING ADVERSARY

In this section we present our basic protocol for generating a distributed DSS signature which enjoys the following properties.

- It is a secure DSS threshold signature scheme in the presence of an eavesdropping adversary (Section 2) when the number of players is $n \geqslant 2t + 1$ where t is the number of faults.
- It is a secure DSS threshold signature scheme in the presence of a halting adversary when the number of players is $n \geqslant 3t + 1$ where t is the number of faults.

In other words, this protocol preserves security (secrecy and unforgeability) in the presence of less than half of the eavesdropping faults. On the other hand, this protocol is robust in the presence of an adversary that in addition to eavesdropping can halt the operation of up to a third of the players by, for example, crashing servers or disconnecting them from the communication lines.

Outline. Initially every player P_i has a share x_i of the secret key x, shared with a polynomial $F(\cdot)$ of degree t, i.e., $(x_1, ..., x_n) \xleftarrow{(t, n)} x \bmod q$. First the players generate distributively a random k (with a random t-degree polynomial $G(\cdot)$) by running the joint Shamir random secret sharing protocol Joint-Shamir-RSS (Fig. 1). To compute $r = g^{k^{-1}} \bmod p \bmod q$ without revealing k, the players use a variation of the reciprocal protocol (Section 4.6) where the value $g^{k^{-1}}$ is reconstructed rather than the value k^{-1}. For the generation of the signature's value s, we note that $s = k(m + xr) \bmod q$ corresponds to the constant term of the multiplication polynomial $G(\cdot)(m + rF(\cdot))$. Since the players have shares of both $G(\cdot)$ and $m + rF(\cdot)$, they can compute s by performing the multiplication protocol (Section 4.7). The full description of protocol DSS-Thresh-Sig-1 is presented in Fig. 5.

Notation. In the description of DSS-Thresh-Sig-1 we use the following notation for two share interpolation operations:

- $v = \text{Interpolate}(v_1, ..., v_n)$. If $\{v_1, ..., v_n\}$ $(n \geqslant 2t + 1)$ is a set of values such that at most t are *null* and all the remaining ones lie on some t-degree polynomial $F(\cdot)$, then $v \triangleq F(0)$. The polynomial can be computed by standard polynomial interpolation.
- $\beta = \text{Exp-Interpolate}(w_1, ..., w_n)$. If $\{w_1, ..., w_n\}$ $(n \geqslant 2t + 1)$ is a set of values such that at most t are *null* and the remaining ones are of the form $g^{a_i} \bmod p$ where the a_i's lie on some t-degree polynomial $G(\cdot)$, then $\beta \triangleq g^{G(0)}$. This can be computed

DSS Signature Generation – Protocol DSS-Thresh-Sig-1

1. **Generate** k
   The players generate a secret random value k, uniformly distributed in Z_q, with a polynomial of degree t, using Joint-Shamir-RSS, which creates shares $(k_1, \ldots, k_n) \overset{(t,n)}{\longleftrightarrow} k \bmod q$.

   Secret information of P_i : share k_i of k

2. **Generate random polynomials with constant term 0**
   Execute two instances of Joint-Shamir-ZSS with polynomials of degree $2t$. Denote the shares created in these protocols as $\{b_i\}_{i\in\{1\ldots n\}}$ and $\{c_i\}_{i\in\{1\ldots n\}}$.

   Secret information of P_i : shares b_i, c_i

3. **Compute** $r = g^{k^{-1}} \bmod p \bmod q$

   (a) The players generate a random value a, uniformly distributed in Z_q^*, with a polynomial of degree t, using Joint-Shamir-RSS, which creates shares $(a_1, \ldots, a_n) \overset{(t,n)}{\longleftrightarrow} a \bmod q$.

   Secret information of P_i : share a_i of a

   (b) Player P_i broadcasts $v_i = k_i a_i + b_i \bmod q$ and $w_i = g^{a_i} \bmod p$. If P_i does not participate his values are set to *null*. Notice that $(v_1, \ldots, v_n) \overset{(2t,n)}{\longleftrightarrow} ka \bmod q$.

   Public information: $\{v_i\}_{i\in\{1\ldots n\}}, \{g^{a_i}\}_{i\in\{1\ldots n\}}$

   (c) Player P_i locally computes

   - $\mu \triangleq \text{Interpolate}(v_1, \ldots, v_n) \bmod q \qquad [= ka \bmod q]$
   - $\beta \triangleq \text{Exp-Interpolate}(w_1, \ldots, w_n) \bmod p \qquad [= g^a \bmod p]$
   - $r \triangleq \beta^{\mu^{-1}} \bmod p \bmod q \qquad [= (g^a)^{\mu^{-1}} = g^{k^{-1}} \bmod p \bmod q]$

   Public information: r

4. **Generate** $s = k(m + xr) \bmod q$

   (a) Player P_i broadcasts $s_i = k_i(m + x_i r) + c_i \bmod q$. If P_i does not participate, his value s_i is set to *null*. Notice that $(s_1, \ldots, s_n) \overset{(2t,n)}{\longleftrightarrow} k(m + xr) \bmod q$.

   Public information: $\{s_i\}_{i\in\{1\ldots n\}}$

   (b) Each player computes $s \triangleq \text{Interpolate}(s_1, \ldots, s_n) \bmod q$.

   Public information: s

5. **Output** (r, s) as the signature for m.

FIG. 5. DSS-Thresh-Sig-1—halting ($n \geqslant 3t+1$) or eavesdropping ($n \geqslant 2t+1$) adversary.

by the interpolation in the exponent method of Section 4.2, i.e., $\beta = \prod_{i\in V'} w_i^{\lambda_{i,V'}} = \prod_{i\in V'} (g^{G(i)})^{\lambda_{i,V'}}$, where V' is a $(t+1)$-subset of the correct w_i's and $\lambda_{i,V'}$'s are the corresponding Lagrange interpolation coefficients.

The following lemma can be easily proven by inspection of the protocol:

LEMMA 1. DSS-Thresh-Sig-1 *is a* $(t, 0, n = 3t+1)$*-robust threshold DSS signature generation protocol, namely it tolerates up to* t *eavesdropping and halting faults if the total number of players is* $n \geqslant 3t+1$.

We need to show that the protocol is unforgeable. We do this by showing that DSS-Thresh-Sig-1 is simulatable. A simulator $\mathcal{SIM}$-1 for the protocol is shown

$\mathcal{SIM}$-1

Input: public key y, message m, its DSS signature (r, s), shares $(x_1, ..., x_t)$ of the corrupted players.

0. Compute $r^* = g^{ms^{-1}} y^{rs^{-1}} \bmod p$.
1. $\mathcal{SIM}$-1 shares a random value uniformly distributed in Z_q for each good player using Shamir-SS. It also listens to the sharings done by the adversary. Let $\hat{k}$ be the resulting value of the associated Joint-Shamir-RSS thus performed. Notice that $\hat{k}$ is uniformly distributed in Z_q and also is known to $\mathcal{SIM}$-1 (since he holds the information of more than t players).
2. $\mathcal{SIM}$-1 runs the part of the good players in two instances of Joint-Shamir-ZSS. Set $\hat{b}_i$, $\hat{c}_i$ for $1 \le i \le n$ to the output of these invocations. Notice that *all* these values are known to $\mathcal{SIM}$-1.
3. (a) $\mathcal{SIM}$-1 runs the part of the good players in the Joint-Shamir-RSS. Let $\hat{a}$ be the resulting shared value and $\hat{a}_1, \ldots, \hat{a}_t$ the values held by the adversary. Note that all those values are known to $\mathcal{SIM}$-1.

   (b) Choose a random value $\hat{\mu}$ uniformly distributed in $[0..q-1]$. Denote $(r^*)^{\hat{\mu}}$ by $g^{\hat{a}}$. (We stress that $g^{\hat{a}}$ is only a notation for $(r^*)^{\hat{\mu}}$; the value $\hat{a}$ is never explicitly computed in the simulation).
   From the values $\hat{w}_0 = (r^*)^{\hat{\mu}}$ and $\hat{w}_i = g^{\hat{a}_i}$ $(i = 1, \ldots, t)$, $\mathcal{SIM}$-1 generates $\hat{w}_i = g^{\hat{a}_i}$ for $i = t+1, \ldots, n$ in such a way that all the $\hat{w}_j$'s interpolate to $g^{\hat{a}}$ "in the exponent", i.e., $\hat{w}_i = g^{\hat{a}_i} = g^{\lambda_{i,0}\hat{a} + \sum_{k=1}^{t} \lambda_{i,k}\hat{a}_k} = (g^{\hat{a}})^{\lambda_{i,0}} \prod_{k=1}^{t} (\hat{w}_k)^{\lambda_{i,k}}$ for $i = t+1, \ldots, n$ and known values $\lambda_{i,k}$.
   Compute $\hat{v}_i \triangleq \hat{a}_i \hat{k}_i + \hat{b}_i$ for $i = 1, \ldots, t$. Choose random shares $\hat{v}_i$ for $i = t+1, \ldots, 2t$. Values $\hat{v}_1, \ldots, \hat{v}_{2t}$ define a unique polynomial $f^{(\hat{v})}(z)$ of degree $2t$ such that $f^{(\hat{v})}(0) = \hat{\mu}$. Complete the shares $\hat{v}_i \triangleq f^{(\hat{v})}(i)$ for $i = 2t+1, \ldots, n$.
   Broadcast the values $\hat{w}_i$ and $\hat{v}_i$ for each player P_i, $i = t+1, \ldots, n$, i.e. for the good players.

   (c) All computations in this step follow from the already computed public information.
4. Compute $\hat{s}_i \triangleq \hat{k}_i(m + x_i r) + \hat{c}_i$ for $i = 1, \ldots, t$. Choose random shares $\hat{s}_i$ for $i = t+1, \ldots, 2t$. Values $\hat{s}_1, \ldots, \hat{s}_{2t}$ define a unique polynomial $\hat{f}^{(s)}(z)$ of degree $2t$ such that $\hat{f}^{(s)}(0) = s$. Complete the shares $\hat{s}_i \triangleq \hat{f}^{(s)}(i)$ for $i = 2t+1, \ldots, n$.
   Broadcast the values $\hat{s}_i$ for each player P_i, $i = t+1, \ldots, n$, i.e. for the good players.

FIG. 6. Simulation protocol for DSS-Thresh-Sig-1.

in Fig. 6. It takes as input the public key $y \triangleq g^x$, a message m, its DSS signature (r, s), and shares $(x_1, ..., x_t)$ of the corrupted parties and runs a simulated execution of the protocol with an adversary who controls (without the loss of generality) the first t players, i.e., learns their secret values $x_1, ..., x_t$ and may cause them to crash during the protocol. By "without loss of generality" we mean (i) that the adversary compromises the *first* t players and (ii) that the simulation is successful if the adversary compromises *less* than t shares. Both these points are easily argued.

LEMMA 2. *Fix an eavesdropping or halting adversary $\mathcal{A}$. Let the number of players be $n = 2t+1$, where t is the number of faults. $\mathcal{VIEW}_{\mathcal{A}}$(DSS-Thresh-Sig-1$(x_1, ..., x_n(m, y)) = (r, s)$) has the same probability distribution as $\mathcal{SIM}$-1$(m, (r, s), y, x_1, ..., x_t)$.*

Proof. We exhibit the proof by comparing the information generated by each step of the $\mathcal{SIM}$-1 protocol in Fig. 6 to the information generated by DSS-Thresh-Sig-1 in Fig. 5.

1. Both the protocol and the simulator execute a sharing of a random secret (k and $\hat{k}$, respectively). As Shamir's secret sharing is information theoretically secure, all subsets of t shares have the same probability. Thus, the sharings of two (possibly) different secrets generate the same distribution for the sets of size t. As $\mathcal{A}$ sees t shares in the protocol and receives t shares from the simulator, this distribution is identical.

2. The reasoning is similar to the previous step.

3. (a) Again, by the same argument as in Step 1, the distribution of the view of the adversary after this step is identical between a real and a simulated run.

(b) The public values $v_1, ..., v_n$ interpolate to some random uniformly distributed value in $[1..q-1]$. The shares $\hat{v}_1, ..., \hat{v}_n$ interpolate the value $\hat{\mu}$ which is random and uniformly distributed in $[1..q-1]$. In addition, the share v_i, for $1 \leqslant i \leqslant t$, satisfies that $v_i = k_i a_i + b_i$. But the share $\hat{v}_i$, for $1 \leqslant i \leqslant t$, also has this property.

The value $g^{\hat{a}}$ was generated by choosing a random value $\hat{\mu}$ uniformly distributed in $[1..q-1]$ and computing $(r^*)^{\hat{\mu}}$ which is equal to $g^{k^{-1}\hat{\mu}}$. The value $k^{-1}\hat{\mu}$ is uniformly distributed in $[1..q-1]$; hence the distribution of g^a and $g^{\hat{a}}$ is the same. The rest of the values $g^{\hat{a}_i}$ for $t+1 \leqslant i \leqslant n$ are obtained through a deterministic computation from $g^{\hat{a}}$ and $g^{\hat{a}_i}$ for $1 \leqslant i \leqslant t$; hence they too have the same distribution as g^{a_i} for $1 \leqslant i \leqslant t$.

4. Same argument as above noting that the shares interpolate the secret s and that they were properly generated by $\mathcal{SIM}$-1.

This completes the proof of Lemma 2. ▮

From the above lemmas we derive the following:

THEOREM 1. *Under the DSS assumption,* DSS-Thresh-Sig-1 *is a secure, i.e., robust and unforgeable, threshold DSS signature generation protocol in the presence of t eavesdropping (halting) faults if the total number of players is* $n \geqslant 2t+1$ $(n \geqslant 3t+1)$.

7. ROBUST THRESHOLD DSS PROTOCOLS

In this section we present a robust version of protocol DSS-Thresh-Sig-1 which remains secure even in the presence of a fully *malicious adversary*. The protocol, DSS-Thresh-Sig-2, requires only the assumption of the unforgeability of DSS signatures and can tolerate $\frac{n-1}{4}$ malicious faults. Using some ideas from [CMI93] the protocol can be modified to resist $\frac{n-1}{3}$ malicious faults at the expense of added computational complexity (see Section 8).

Outline. Our aim is to prove the security of our protocol based only on the unforgeability of DSS signatures. Thus we require that the random value k is jointly generated by the players using Joint-Uncond-Secure-RSS instead of Joint-Shamir-RSS. This guarantees that no information is leaked on the values k or k^{-1} in the presence of a malicious adversary.[8] Then the players compute r as in DSS-Thresh-Sig-1, with the only difference that the protocol Joint-Shamir-RSS that generates a random value a and the two protocols Joint-Shamir-ZSS that generate randomizers $\{b_i, c_i\}$ are replaced by protocols that handle a malicious adversary, i.e., Joint-Exp-RSS and Joint-Uncond-Secure-ZSS. We use Joint-Exp-RSS rather than Joint-Uncond-Secure-RSS to generate a because the value $g^a \bmod p$ *needs* to be revealed to enable the public computation of $r = g^{k^{-1}} = (g^a)^{\mu^{-1}}$. As

[8] Note that if we used Joint-Exp-VSS in the first step of the protocol then we would reveal g^k, which is not available from a regular DSS signature.

DSS Signature Generation – Protocol DSS-Thresh-Sig-2

1. **Generate k**
The players generate a secret value k, uniformly distributed in Z_q, by running Joint-Uncond-Secure-RSS with a polynomial of degree t. Notice that this generates $(k_1, \ldots, k_n) \overset{(t,n)}{\longleftrightarrow} k \bmod q$.

Secret information of P_i : a share k_i of k

2. **Generate random polynomials with constant term 0**
Execute two instances of Joint-Uncond-Secure-ZSS with polynomials of degree $2t$. Denote the shares created in these protocols as $\{b_i\}_{i\in\{1\ldots n\}}$ and $\{c_i\}_{i\in\{1\ldots n\}}$.

Secret information of P_i : shares b_i, c_i

3. **Generate $r = g^{k^{-1}} \bmod p \bmod q$**

(a) Generate a random value a, uniformly distributed in Z_q^*, with a polynomial of degree t, using Joint-Exp-RSS.

Secret information of P_i : a share a_i of a Public information: g^a

(b) Player P_i broadcasts $v_i = k_i a_i + b_i \bmod q$. If P_i doesn't broadcast a value set v_i to *null*.

Public information: $v_1, \ldots, v_n$ where for at least $n - t$ values j it holds that $v_j = k_j a_j + b_j \bmod q$

(c) Player P_i computes locally

- $\mu \triangleq \mathsf{EC\text{-}Interpolate}(v_1, \ldots, v_n) \bmod q \qquad [= ka \bmod q]$
- $\mu^{-1} \bmod q \qquad [= k^{-1}a^{-1} \bmod q]$
- $r \triangleq (g^a)^{\mu^{-1}} \bmod p \bmod q \qquad [= g^{k^{-1}} \bmod p \bmod q]$

Note: Even though the above computations are local, as they are done on public information we can assume that:

Public information: r

4. **Generate $s = k(m + xr) \bmod q$**
Player P_i broadcasts $s_i = k_i(m + x_i r) + c_i \bmod q$.

Public information: $s_1, \ldots, s_n$ where for at least $n - t$ values j it holds that $s_j = k_j(m + x_j r) + c_j \bmod q$

Set $s \triangleq \mathsf{EC\text{-}Interpolate}(s_1, \ldots, s_n)$.

5. **Output** the pair (r, s) as the signature for m

FIG. 7. Malicious adversary, $n \geq 4t + 1$.

before, s is computed from the appropriate shares, and the randomizing polynomials shared by values $\{b_i, c_i\}$ are used to hide possible partial information. The full protocol is exhibited in Fig. 7.

Notation. In the protocol, we use the following notation:

$$v = \mathsf{EC\text{-}Interpolate}(v_1, \ldots, v_n).$$

If $\{v_1, \ldots, v_n\}$ $(n = 4t + 1)$ is a set of values such that at least $3t$ of the values lie on some $2t$-degree polynomial $F(\cdot)$, then $v \triangleq F(0)$. The polynomial can be computed by using the Berlekamp–Welch decoder [BW].

Convention of Fig. 7. In the protocol description in Fig. 7 we include boxes labeled "Public information" or "Private information of P_i" that provide *only* the information that is subsequently used by the honest players in this protocol. Note that much more data are actually produced as public and private outputs at each step.

THEOREM 2. *Under the DSS assumption, Protocol* DSS-Thresh-Sig-2 *is a secure (unforgeable and robust) threshold signature protocol for Dss resistant to t faults against a static malicious adversary, when the number of player is* $n \geq 4t+1$.

The proof of Theorem 2 easily follows from the proofs of Lemmas 3 and 4 below. It is important to note that unforgeability is obtained for $n \geq 2t+1$ (see Lemma 4), while $n \geq 4t+1$ is needed only for robustness (see Lemma 3.)

LEMMA 3. DSS-Thresh-Sig-2 *is a* (h, c, n)*-robust threshold DSS signature generation protocol, if* $h+c \leq t$ *and* $n \geq 4t+1$; *that is,* DSS-Thresh-Sig-2 *can tolerate up to* $\frac{n-1}{4}$ *malicious faults.*

Proof. The correctness of the protocol is due to the error correcting capabilities of polynomial interpolation. Since we are interpolating a polynomial of degree $deg = 2t$ and we have $faults = t$ possible errors, using the Berlekamp–Welch bound we get that the number of points needed in order to correctly interpolate in $deg + 2faults + 1 = 4t+1$. Hence, we set $n \geq 4t+1$. ∎

As in the previous section, in order to prove the unforgeability of DSS-Thresh-Sig-2 protocol against a malicious adversary, we present in Fig. 8 a simulator $\mathcal{SIM}$-2, which on input the public key $y \triangleq g^x$, a message m, and its DSS signature (r, s), runs a simulated execution of the protocol with an adversary who controls the first t players.

$\mathcal{SIM}$-2

Input: public key y, message m, its DSS signature (r, s), shares $(x_1, \ldots, x_t)$ of the corrupted players.

0. Compute $r^* = g^{ms^{-1}} y^{rs^{-1}} \bmod p$.
1. Run Joint-Uncond-Secure for the honest players. Let $\hat{k}$ be the resulting shared value and $\hat{k}_i$ the share held by player P_i. Notice that *all* these values are known to $\mathcal{SIM}$-2.
2. Run two instances of Joint-Uncond-Secure-ZSS for the honest players.. Set $\hat{b}_i$, $\hat{c}_i$ for $1 \leq i \leq n$ to the output of these invocations. Notice that *all* these values are known to $\mathcal{SIM}$-2.
3. (a) Choose a random value $\hat{\mu}$ uniformly distributed in $[0..q-1]$. $\mathcal{SIM}$-2 simulates a run of Joint-Exp-RSS protocol with $g^{\hat{a}} = (r^*)^{\hat{\mu}}$ as the public output. That is $\mathcal{SIM}$-2 runs Sim-Exp-RSS on input $(r^*)^{\hat{\mu}}$.
   Let $\hat{a}_i$ for $i = 1, \ldots, t$ the shares held by the adversary at the end of this simulation. Notice that these values are known to $\mathcal{SIM}$-2.
   (b) Compute $\hat{v}_i \triangleq \hat{a}_i \hat{k}_i + \hat{b}_i$ for $i = 1, \ldots, t$. Choose random shares $\hat{v}_i$ for $i = t+1, \ldots, 2t$. Let $f^{(\hat{v})}$ be the $2t$-degree polynomial defined by $f^{(\hat{v})}(0) = \hat{\mu}$ and $f^{(\hat{v})}(i) = \hat{v}_i$ for $i = 1, \ldots, 2t$. Complete the shares $\hat{v}_i \triangleq f^{(\hat{v})}(i)$ for $i = 2t+1, \ldots, n$.
   Broadcast $\hat{v}_{t+1}, \ldots, \hat{v}_n$.
   (c) All computations in this step follow from the already computed public information.
4. Compute $\hat{s}_i \triangleq \hat{k}_i(m + x_i r) + \hat{c}_i$ for $i = 1, \ldots, t$. Choose random shares $\hat{s}_i$ for $i = t+1, \ldots, 2t$. Let $f^{(\hat{s})}$ be the $2t$-degree polynomial defined by $f^{(\hat{s})}(0) = s$ and $f^{(\hat{s})}(i) = \hat{s}_i$ for $i = 1, \ldots, 2t$. Complete the shares $\hat{s}_i \triangleq f^{(\hat{s})}(i)$ for $i = 2t+1, \ldots, n$.
   Broadcast $\hat{s}_{t+1}, \ldots, \hat{s}_n$.

FIG. 8. Simulation protocol for DSS-Thresh-Sig-2.

LEMMA 4. *Fix a malicious adversary $\mathcal{A}$. Let the number of players be $n = 2t + 1$, where t is the number of faults. $\mathcal{VIEW}_{\mathcal{A}}$(DSS-Thresh-Sig-2$(x_1, ..., x_n, (m, y)) = (r, s)$) has the same probability distribution as $\mathcal{SIM}$-2$(m, (r, s), y, x_1, ..., x_t)$.*

Outline of the proof. The simulator is shown in Fig. 8.

For the first two rounds of the protocol (rounds 1–2) the simulator simply follows the protocol for the honest players. Since this step does not involve the secret keys x_i (that the simulator does not know) it is clear that the view of the adversary in these first two rounds will be the same as in the real execution.

At the end of this step the simulator has a sharing of a dummy value $\hat{k}$. But in the third round it must produce the first part of the signature $r = g^{k^{-1}} \bmod p \bmod q$ without knowing k. In order to produce r it cheats as follows. It chooses a random $\hat{\mu}$ and simulates the execution of Joint-Exp-RSS in order to get $g^a = (r^*)^{\hat{\mu}}$ as the output. See Section 4.4 for the details of the simulation of Joint-Exp-RSS. This is the crucial step where the result of [GJKR99] helps.

Then when it comes to reconstruct $ak = \mu$ (in the real protocol) the simulator cheats by broadcasting shares for the honest players that interpolate to $\hat{\mu}$ while still being consistent with the ones held by the adversary. It is easy to verify that this will result in the correct r as the first part of the signature in the simulated execution.

As for s the trick is even simpler. The simulator has to hit s without knowing the secret keys x_i of the honest players. But it knows the shares s_i held by the adversary; hence it broadcasts for the honest players random shares that interpolate to s and are consistent with $s_1, ..., s_t$.

Proof. 1. Both the protocol and the simulator execute a sharing of a random secret (k and $\hat{k}$, respectively) with an unconditionally secure VSS. As the sharing is information theoretically secure all subsets of t shares plus the public information have the same distribution of the adversary's shares and of the public information. Since this is all $\mathcal{A}$ sees in the protocol and in the simulation, this step is secure.

Notice that this distribution on subsets of t shares is guaranteed even if the corrupted parties contribute nonrandom shares to the generation of k, as long as these shares are consistent (i.e., they interpolate to a single polynomial). Inconsistent shares are always detected and then discarded.

2. The reasoning is similar to the previous step, with the only difference being that here the sharing is of a zero value.

3. (a) This is the step where the properties of Joint-Exp-RSS make the proof go through. In the real execution of this step, Joint-Exp-RSS results in the value g^a being public while in the simulated execution the value $g^{\hat{a}} = (r^*)^{\mu}$ is chosen by the simulator.

Thanks to the correctness property of Joint-Exp-RSS (see Section 4.4) the value g^a is random and uniformly distributed. But the value $(r^*)^{\mu}$ is also random and uniformly distributed since μ was chosen uniformly at random in Z_q.

The remaining part of the view of the adversary in this step can be simulated by Sim-Exp-RSS as we show in Section 4.4.

(b) The values $\hat{v}_1, ..., \hat{v}_n$ are identically distributed to the values $v_1, ..., v_n$ since

- the public values $v_1, ..., v_n$ interpolate to some random uniformly distributed value in $[1..q-1]$. The shares $\hat{v}_1, ..., \hat{v}_n$ interpolate the value $\hat{\mu}$ which is also random and uniformly distributed in $[1..q-1]$.

- the share v_i, for $1 \leqslant i \leqslant t$, satisfies that $v_i = k_i a_i + b_i$. The share $\hat{v}_i$, for $1 \leqslant i \leqslant t$, was generated in this manner (see Step 3b).

- In the real execution let $f^{(k)}$, $f^{(a)}$ be the t-degree polynomials defined respectively by the shares k_i, a_i. Moreover let $f^{(b)}, f^{(v)}$ be the $2t$-degree polynomial defined respectively by the shares b_i, v_i. It holds that $f^{(v)} = f^{(a)} f^{(k)} + f^{(b)}$ and $f^{(b)}(0) = 0$. In the simulated execution we have the polynomial $f^{(\hat{a})}$ which is uniquely defined by the public information revealed in Step 3a and the polynomial $f^{(\hat{v})}$ defined by the shares $\hat{v}_i$. Let $f^{(\hat{k})}$ be the t-degree polynomial defined by $f^{(\hat{k})}(i) = \hat{k}_i$ $(i = 1, ..., t)$ and $f^{(\hat{k})}(0) = f^{(\hat{v})}(0)/f^{(\hat{a})}(0) \bmod q$. Define the $2t$-degree polynomial $f^{(\hat{b})} \triangleq f^{(\hat{v})} - f^{(\hat{a})} f^{(\hat{k})}$. Clearly $f^{(\hat{b})}$ "agrees" with the t points $\hat{b}_i$ held by the adversary (i.e., $f^{(\hat{b})}(i) = \hat{b}_i$) and $f^{(\hat{b})}(0)$. This means that, as in the real execution, the polynomial $f^{(\hat{v})}$ can be written as $f^{(\hat{a})} f^{(\hat{k})} + f^{(\hat{b})}$ where $f^{(\hat{a})}, f^{(\hat{k})}, f^{(\hat{b})}$ are polynomials of the appropriate degree which match the points held by the adversary. Notice that the polynomials $f^{(\hat{k})}, f^{(\hat{b})}$ may *not* be the ones resulting from the sharings in Steps 1 and 2 of the simulation. But since those VSS's are information theoretically secure, the above polynomials are as likely as any other pair of polynomials, given the public information and the t shares held by the adversary.

Thus the view of the adversary in this step is identically distributed between the real and the simulated execution.

4. The same argument as above noting that the shares interpolate the secret s and that they were properly generated by $\mathcal{SIM}$-2 in Step 4.

This completes the proof of Lemma 4. ▮

Comparison with the [CMI93] *approach.* In [CMI93] the authors add robustness to the basic protocol by turning all the regular Shamir-SS secret sharing protocols into robust Exp-VSS ones. Although this allows us to tolerate malicious faults it also produces an extra *information leak*: namely, both g^k and $g^{k^{-1}}$ are revealed. As we pointed out before, this is not information that can be derived from a regular DSS signature. To claim security in this case one needs to assume that if one choose u, v at random, uniformly and independently in Z_q, then the following probability distributions $(g^u \bmod p,\ g^v \bmod p)$ and $(g^u \bmod p,\ g^{u^{-1}} \bmod p)$ are computationally indistinguishable. We were not able to reduce this assumption to the basic unforgeability of the DSS assumption that we make. Another drawback of the [CMI93] approach is that all the joint Feldman VSS protocols are done in a straightforward way which suffers from the problems described in Section 4.4 Finally, and importantly, we note that our approach using error-correction techniques in the signature reconstruction step results in a significantly more efficient protocol (at the expense of tolerating $\frac{n-1}{4}$ malicious faults instead of $\frac{n-1}{3}$).

8. MALICIOUS ADVERSARY, $n \geqslant 3t+1$

It is possible to devise a protocol that works in the presence of t maliciously behaving players, when n is greater than $3t+1$. The gain in fault-tolerance, however, comes at the expense of an increased amount of computation (e.g., modular exponentiations) required from the players in order to compute a single signature.

The previous protocol used the Feldman and Pederson VSS protocols [Fel87, Ped91b] only to ensure that secrets were shared correctly. However, when there was a need to verify the shares broadcasted by players in order to reconstruct a value, we relied on error-correcting codes (i.e., in Steps (3c) and (4) of Fig. 7). This allowed us to reconstruct the values without further modular exponentiations, but forced us to set the fault-tolerance to $\frac{n-1}{4}$. The protocol sketched in this section will make more extensive use of the properties of Feldman's VSS protocol for the authentication of shares.

In a regular execution of Exp-VSS the share s_i broadcasted by player P_i can be checked against the publicly known value g^{s_i} for authenticity. Notice that this easily generalizes to the reconstruction of the linear combination of secrets. For example, we know that if $(a_1, ..., a_n) \xleftarrow{(t,n)} a$ and $(b_1, ..., b_n) \xleftarrow{(t,n)} b$ then $(a_1+b_1, ..., a_n+b+n) \xleftarrow{(t,n)} a+b$. If we want to reconstruct $a+b$, then player P_i broadcasts a_i+b_i which can be checked against $g^{a_i+b_i} = g^{a_i}g^{b_i}$. Things, however, get more complicated if we want to reconstruct ab because we cannot sieve out bad shares as before, since we do not know how to compute $g^{a_ib_i}$ from g^{a_i} and g^{b_i}.

This is exactly the situation in our previous protocol. The values reconstructed are the product of other shared values. On top of that, one of those shared values, k, has been shared with Uncond-Secure-VSS and not Exp-VSS. However, there is also an advantage: at least one of the values multiplied is *random* and has been "recently" shared using a Joint-Exp-VSS protocol.

This observation led to a clever trick employed in [CMI93]. They show that in the situation described above it is possible to create "authentication pieces" for the resulting shares of the product. In their case both secrets are shared using Exp-VSS. We show that the trick can be adapted to our case in which one of the values is shared with Uncond-Secure-VSS.

We will now proceed to sketch the method in more detail. Let $(a_1, ..., a_n) \xleftarrow{(t,n)} a$ be a sharing obtained from a run of Joint-Exp-VSS. Let $A(z) = A_0 + A_1 z + \cdots + A_t z^t$ be the actual t-degree polynomial that shares the value $a = A_0$. Player P_i holds share $a_i = A(i) \bmod p$ and the values $g^{A_j} \bmod p$ are publicly known.

Now assume that the players together generate the sharing of a random value k using Joint-Uncond-Secure-RSS. This means in particular that each player P_i runs one instance of Uncond-Secure-VSS using a random t-degree polynomial $K^{(i)}(z) = K_0^{(i)} + K_1^{(i)} z + \cdots + K_t^{(i)} z^t$ playing the role of $f_i(z)$ in Step 1 of Fig. 2. We also assume that the players together generate a random sharing of zero with a $2t$ degree polynomial using Joint-Uncond-Secure-ZSS. This means that each player runs an instance of Uncond-Secure-ZSS with a $2t$-degree polynomial $B^{(i)}(z) = B_1^{(i)} z + \cdots + B_{2t}^{(i)} z^{2t}$ playing the role of $f_i(z)$ in Step 1 of Fig. 2. For all i, every player P_j hold shares $K^{(i)}(j)$ and $B^{(i)}(j)$ of these polynomials. Let $K(z) = \sum_i K^{(i)}(z)$ and $B(z) = \sum_i B^{(i)}(z)$.

Consider now the polynomial $C(z) = A(z)\,K(z) + B(z)$ which shares the value $C(0) = ak$ that we want to robustly reconstruct. Our purpose is to make values $\gamma_j = g^{C_j} \bmod p$ public, where $C(z) = C_0 + C_1 z + \cdots + C_{2t} z^{2t}$. If such $\gamma_0, ..., \gamma_{2t}$ were public in Step (3c) of the DSS-Thresh-Sig-2 protocol (Fig. 7) then we could sieve out the bad shares using Feldman's procedure instead of relying on error-correcting codes. Note that

$$C(z) = A(z)\,K(z) + B(z) = \sum_i A(z)\,K^{(i)}(z) + \sum_i B^{(i)}(z) = \sum_i C^{(i)}(z)$$

if one lets $C^{(i)}(z) = A(z)\,K^{(i)}(z) + B^{(i)}(z)$. Let $C^{(i)}(z) = C_0^{(i)} + C_1^{(i)} z + \cdots + C_{2t}^{(i)} z^{2t}$. Since $C(z) = \sum_i C^{(i)}(z)$, we see that values $\gamma_0, ..., \gamma_{2t}$ will be known if each player P_i broadcasts

$$g^{C_0^{(i)}},\ g^{C_1^{(i)}},\ ...,\ g^{C_{2t}^{(i)}} \bmod p$$

which he can easily compute from values g^{A_j} and the coefficients of $K^{(i)}(z)$ and $B^{(i)}(z)$ (assume $B_0^{(i)} = 0$):

$$g^{C_k^{(i)}} = \left(\prod_{\substack{\alpha+\beta=k \\ 0 \leqslant \alpha,\, \beta \leqslant t}} (g^{A_\alpha})^{K_\beta^{(i)}} \right) g^{B_k^{(i)}} \bmod p. \tag{5}$$

The broadcasted values can be checked by the other players using the usual Feldman's verification procedure since every player P_j holds a value $C^{(i)}(j) = A(j)\,K^{(i)}(j) + B^{(i)}(j)$ of polynomial $C^{(i)}(z)$: Each player checks that the point on the polynomial he owns does indeed interpolate in the exponent to the values broadcasted by P_i.

In this way, when the product ka is reconstructed, the t faulty shares can be immediately sieved out, and the number of players can be reduced to $3t + 1$ (since we still need at least $2t + 1$ players to reconstruct a polynomial of degree $2t$).

Protocol DSS-Thresh-Sig-3. The application of the above method to our DSS signature generation protocol is almost immediate. To create a DSS-Thresh-Sig3 protocol which is secure against a malicious adversary with threshold $n \geqslant 3t + 1$, we transform protocol DSS-Thresh-Sig-2 (Fig. 7) by replacing the error-correcting mechanism with the above method in the computation of the products ka (Step 3c) and kx (Step 4).

The drawback of this version of the robust protocol is that while error-correction is fast, this method requires several extra modular exponentiations.

9. EFFICIENCY CONSIDERATIONS

In this section we give an analysis of the computational effort required to compute DSS signatures in a distributed way using our protocols. We use as a measure the number of long modular exponentiations required by a single player during the execution of the protocol.

We note that in DSS-Thresh-Sig-1 and DSS-Thresh-Sig-2 all the modular exponentiations happen during the computation of r (i.e., in Steps 1–3). This is also a property of a centralized DSS signature algorithm. Since r is independent of the message being signed, this part of the protocol can be moved off-line, and then the computation of the actual on-line signature would be a very fast and noninteractive protocol (i.e., it would consist of only Step 4). This nice off-line–on-line feature of DSS is, however, not preserved in DSS-Thresh-Sig-3 (since there modular exponentiations are required to verify the correctness of individual shares s_i to compute s). This shows the advantage of our approach based on error-correcting codes (used in DSS-Thresh-Sig-2) over the [CMI93] approach to obtain robustness (used in DSS-Thresh-Sig-3). Beyond being more efficient overall (less modular exponentiations are performed), it also maintains the on-line efficiency typical of DSS signatures.

A close analysis of the three protocols presented in the previous sections reveals that:

1. DSS-Thresh-Sig-1 requires $t+3$ long modular exponentiations per server.

   (Step 3b): 1 exponentiation.

   (Step 3c): $t+1$ long exponentiations in Exp-Interpolate of β and 1 exponentiation in computing r

2. DSS-Thresh-Sig-2 requires $8t+6n+1$ modular exponentiations per server if no faults occur.

   (Step 1) Joint-Uncond-Secure-RSS with a t degree polynomial: $2(t+1)$ exponentiations in Step 1b of Fig. 2 plus $2(n-1)$ (long) exponentiations in verification of Eq. (3) for each other player.

   (Step 2) Joint-Uncond-Secure-ZSS with a $2t$ degree polynomial: $2(2t+1)+2(n-1)$ exponentiations, in the similar steps as above.

   (Step 3a) Joint-Exp-RSS with t-degree polynomial: $2(t+1)+2(n-1)$ exponentiations as in Joint-Uncond-Secure-RSS, with no additional (long) exponentiations in verifying Eq. (4).

   (Step 3c): 1 exponentiation.

3. DSS-Thresh-Sig-3 requires $t^2+10t+8n$ long exponentiations per server if no faults occur. The cost is as above plus: $(t+1)^2$: the cost of computing the values $g^{C_k^{(i)}}$. Note that values g^{A_α}, $g^{B_k^{(i)}}$ used in Eq. (5) are already known.

   $2(n-1)$: the verification of the broadcasted values $g^{C_k^{(j)}}$ and the verification of the final values $C(j)$ for each other player P_j.

For example, for values of $n \leqslant 20$ and for maximal thresholds of $t = \lfloor (n-1)/3 \rfloor$ and $t = \lfloor (n-1)/4 \rfloor$, respectively, DSS-Thresh-Sig-3 is at most about 50% more costly than DSS-Thresh-Sig-2.

If a fault occurs in DSS-Thresh-Sig-2 or DSS-Thresh-Sig-3, the corrupted server can force the other servers to incur the largest computational cost by misbehaving in the Joint-Exp-RSS protocol in Step 3a in Fig. 7. Consider the protocol in Fig. 3: if there are less than t complain's in Step 6, each server pays up to $2t$ long exponentiations to verify Eqs. (3) and (4) per each submitted pair (σ_{ji}, ρ_{ji}), and if

this leads to P_j's exposure, the public reconstruction of values a_{jk}, y_{jk} takes at most an additional $2n + t$ long exponentiations. Therefore, the additional cost is up to $2n + 3t$ exponentiations per fault. However, note that each fault results in identification of the faulty party, which from then on is ignored by the honest players. Therefore, no more than t faults per *lifetime* of the distributed DSS signing service can occur. Consequently, the cost caused by failures is negligible when amortized per each signature computation.

ACKNOWLEDGMENTS

Many thanks are due to Don Beaver who pointed out to us the problems with the simulation against a rushing adversary of the original version of Joint-Exp-RSS which we used in [GJKR96a]. The authors also thank Matthias Birkner for pointing out the difference between r and r^* in the simulation of the protocols. Finally, thanks to Don Beaver and Kaoru Kurosawa for useful pointers to references.

Received March 7, 1997; final manuscript received July 26, 1999

REFERENCES

[BB89] Bar-Ilan, J., and Beaver, D. (1994), Noncryptographic fault-tolerant computing in a constant number of rounds, *in* "Proceedings of the ACM Symposium on Principles of Distributed Computation," pp. 201–209.

[B87] Beaver, D. (1987), "Oblivious Secret Computation," TR-12-87, Harvard University.

[B92] Beaver, D. (1992), Foundations of secure interactive computing, *in* "Advances in Cryptology-CRYPTO'91" (J. Feigenbaum, Ed.), Lecture Notes in Computer Science, Vol. 576, pp. 377–391, Springer-Verlag, Berlin/New York.

[BGW88] Ben-Or, M., Goldwasser, S., and Wigderson, A. (1988), Completeness theorems for non-cryptographic fault-tolerant distributed computations, *in* "Proceeding 20th Annual Symposium on the Theory of Computing," pp. 1–10, Assoc. Comput. Mach., New York.

[Boy86] Boyd, C. (1986), Digital multisignatures, *in* "Cryptography and Coding" (H. Baker and F. Piper, Eds.), pp. 241–246, Clarendon Press, Oxford.

[BW] Berlekamp, E., and Welch, L. "Error Correction of Algebraic Block Codes," U.S. Patent, 4, 633, 470

[C+99] Canetti, R., Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. (1999), Adaptive security for threshold cryptosystems, *in* "Advances in Cryptology-CRYPTO'99," Lecture Notes in Computer Science, Vol. 1666, pp. 98–115, Springer-Verlag, Berlin/New York.

[CCD88] Chaum, D., Crepeau, C., and Damgard, I. (1988), Multiparty unconditionally secure protocols, *in* "Proceeding 20th Annual Symposium on the Theory of Computing," pp. 11–19, Assoc. Comput. Mach., New York.

[CMI93] Cerecedo, M., Matsumoto, T., and Imai, H. (1993), Efficient and secure multiparty generation of digital signatures based on discrete logarithms, *IEICE Trans. Fund.* **E76-A**(4), 532–545.

[CH89] Croft, R. A., and Harris, S. P. (1989), Public-key cryptography and re-usable shared secrets, *in* "Cryptography and Coding" (H. Baker and F. Piper, Eds.), pp. 189–201, Clarendon Press, Oxford.

[DDFY94] De Santis, A., Desmedt, Y., Frankel, Y., and Yung, M. (1994), How to share a function securely, *in* "Proc. 26th ACM Symp. on Theory of Computing; Santa Fe, 1994," pp. 522–533, IEEE Press, New York.

[Des88] Desmedt, Y. (1988), Society and group oriented cryptography: A new concept, *in* "Advances in Cryptology-CRYPTO'87" (C. Pomerance, Ed.), Lecture Notes in Computer Science, Vol. 293, pp. 120–127, Springer-Verlag, Berlin/New York.

[Des94] Desmedt, Y. G. (1994), Threshold cryptography, *Europ. Trans. Telecomm.* **5**, 449–457.

[DF90] Desmedt, Y., and Frankel, Y. (1990), Threshold cryptosystems, *in* "Advances in Cryptology-CRYPTO'89" (G. Brassard, Ed.), Lecture Notes in Computer Science, Vol. 435, pp. 307–315, Springer-Verlag, Berlin/New York.

[DF91] Desmedt, Y., and Frankel, Y. (1992), Shared generation of authenticators and signatures, *in* "Advances in Cryptology-CRYPTO'91" (J. Feigenbaum, Ed.), Lecture Notes in Computer Science, Vol. 576, pp. 457–469, Springer-Verlag, Berlin/New York.

[ElG85] ElGamal, T. (1985), A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. Info. Theory* **31**,.

[Fel87] Feldman, P. (1987), A practical scheme for non-interactive verifiable secret sharing, *in* "Proceeding 28th Annual Symposium on the Foundations of Computer Science," pp. 427–437, IEEE Press, New York.

[FM88] Feldman, P., and Micali, S. (1988), An optimal algorithm or synchronous Byzantine agreement, *in* "Proceeding 20th Annual Symposium on the Theory of Computing," pp. 148–161, Assoc. Comput. Mach., New York.

[FGY96] Frankel, Y., Gemmell, P., and Yung, M. (1996), Witness-based cryptographic program checking and robust function sharing, *in* "Proceedings of the ACM Symposium on Theory of Computing."

[GJKR96a] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. (1996), Robust threshold DSS signatures, *in* "Advances in Cryptology—Eurocrypt'96" (U. Maurer, Ed.), Lecture Notes in Computer Science, Vol. 1070, pp. 354–371, Springer-Verlag, Berlin/New York.

[GJKR96b] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. (1996), Robust and efficient sharing of RSA functions, *in* "Advances in Cryptology-CRYPTO'96" (N. Koblitz, Ed.), Lecture Notes in Computer Science, Vol. 1109, pp. 157–172, Springer-Verlag, Berlin/New York.

[GJKR99] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. (1999), Secure distributed key generation in discrete-log based cryptosystems, *in* "Advances in Cryptology—Eurocrypt'99" (J. Stern, Ed.), Lecture Notes in Computer Science, Springer-Verlag, Berlin/New York. [Also available at http://www.research.ibm.com/security/dkg.ps]

[GMR88] Goldwasser, S., Micali, S., and Rivest, R. L. (1988), A digital signature scheme secure against adaptive chosen-message attacks, *SIAM J. Comput.* **17**, 281–308.

[GMR89] Goldwasser, S., Micali, S., and Rackoff, C. (1989), The knowledge complexity of interactive proof-systems, *SIAM J. Comput.* **18**, 186–208.

[GMW87] Goldreich, O., Micali, S., and Wigderson, A. (1987), How to play any mental game, *in* "Proc. 19th Annual Symp. on the Theory of Computing," pp. 218–229, Assoc. Comput. Mach., New York.

[Har94] Harn, L. (1994), Group oriented (t, n) digital signature scheme, *IEE Proc. Comput. Digital Tech.* **141**(5).

[HJKY95] Herzberg, A., Jarecki, S., Krawczyk, H., and Yung, M. (1995), Proactive secret sharing, or: How to cope with perpetual leakage, *in* "Advances in Cryptology—Crypto'95" (D. Coppermisth, Ed.), Lecture Notes of Computer Science, Vol. 963, Springer-Verlag, Berlin/New York.

[HJJKY97] Herzberg, A., Jakobbson, M., Jarecki, S., Krawczyk, H., and Yung, M. (1997), Proactive public-key and signature systems, *in* "Proceedings of the 4th ACM Conference on Computer and Communication Security" pp. 100–110.

[IS90] Ingemarsson, I., and Simmons, G. J. (1990), A protocol to set up shared secret schemes without the assistance of a mutually trusted party, *in* "Advances in Cryptology—Eurocrypt'91" (I. B. Damgård, Ed.), Lecture Notes in Computer Science, Vol. 473, pp. 266–282, Springer-Verlag, Berlin/New York.

[Lan95] Langford, S. (1995), Threshold DSS signatures without a trusted party, *in* "Lecture Notes in Computer Science" (D. Coppermisth, Ed.), Lecture Notes in Computer Science, Vol. 963, pp. 397–409, Springer-Verlag, Berlin/New York.

[MR92] Micali, S., and Rogaway, P. (1992), Secure computation, *in* "Advances in Cryptology—CRYPTO'91" (J. Feigenbaum, Ed.), Lecture Notes in Computer Science, Vol. 576, pp. 392–404, Springer-Verlag, Berlin/New York.

[MS81] McEliece, R., and Sarwate, D. (1981), On sharing secrets and Reed-Solomon codes, *Commun. Assoc. Comput. Mach* **24**, 583–584.

[NIST91] National Institute for Standards and Technology, *in* "Digital Signature Standard (DSS)," Technical Report, 169, August 30, 1991.

[PK96] Park, C., and Kurosawa, K. New ElGamal type threshold digital signature scheme, *IEICE Trans. Fund.* **E79-A**, 86–93.

[Ped91a] Pedersen, T. (1991), Distributed provers with applications to undeniable signatures, *in* "Advances in Cryptology—EUROCRYPT'91" (D. Davies, Ed.), Lecture Notes in Computer Science, Vol. 547, pp. 221–242, Springer-Verlag, Berlin/New York.

[Ped91b] Pedersen, T. (1992), Non-interactive and information-theoretic secure verifiable secret sharing, *in* "Advances in Cryptology—CRYPTO'91" (J. Feigenbaum, Ed.), Lecture Notes in Computer Science, Vol. 576, pp. 129–140, Springer-Verlag, Berlin/New York.

[Ped91c] Pedersen, T. (1991), A threshold cryptosystem without a trusted party, *in* "Advances in Cryptology–EUROCRYPT'91" (D. Davies, Ed.), Lecture Notes in Computer Science, Vol. 547, pp. 522–526, Springer-Verlag, Berlin/New York.

[Rab98] Rabin, T. (1998), A simplified approach to threshold and proactive RSA, *in* "Advances in Cryptology—CRYPTO'98" (H. Krawczyk, Eds.), Lecture Notes in Computer Science, Vol. 1462, pp. 89–104, Springer-Verlag, Berlin/New York.

[RSA78] Rivest, R., Shamir, A., and Adleman, L. (1978), A method for obtaining digital signatures and public-key cryptosystems, *Commun. Assoc. Comput. Mach.* **21**, 120–126.

[Sch91] Schnorr, C. P. (1991), Efficient signature generation by smart cards, *J. Cryptology* **4**, 161–174.

[Sha79] Shamir, A. (1979), How to share a secret, *Commun. Assoc. Comput. Mach.* **22**, 612–613.