Robust Threshold DSS Signatures*

Rosario Gennaro†, Stanisław Jarecki*, Hugo Krawczyk†and Tal Rabin†

March 3, 1997

Abstract

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 $n/3$ players 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.

We prove the security of our protocols solely based on the hardness of forging a regular DSS signature.

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.

*MIT Laboratory for Computer Science, 545 Tech Square, Cambridge, MA 02139, USA. Email: stasio@theory.lcs.mit.edu

†IBM T.J.Watson Research Center, PO Box 704, Yorktown Heights, New York 10598, USA. Email: {rosario,hugo,talr}@watson.ibm.com

*A preliminary version of this paper appeared in the proceedings of EUROCRYPT'96

Threshold signatures are motivated both 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, as well as by the need to protect signature keys from the attack of internal and external adversaries. The latter 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 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.

1.1 Previous Work

Threshold signatures are part of a general approach known as *threshold cryptography* which was introduced by the works of Boyd [Boy86], 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 the work in this area.

It is very important to provide threshold solutions for signatures schemes used in practice, as those systems are the ones that will be 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 [DF92, DDFY94, FGY96, GJKR96].

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.[1] Thus, for n given players this solution can resist up to $\sqrt{n}$ corrupted parties.

A complete analysis of threshold techniques applied to various ElGamal-like schemes appears in an earlier paper by Cerecedo, Matsumoto and Imai [CMI93]. They present formal definitions of threshold signature schemes and solutions based on the ElGamal signature scheme which occur 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 better properties in our solution. We discuss these properties next.

[1] 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 pre-computed tables of one-time shares and that requires a higher level of trust for the generation of these tables. See [Lan95] for details.

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 of regular DSS signatures. Previous work [CMI93] required additional cryptographic assumptions. That is, 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 for any threshold schemes. We present rigorous proofs of the equivalent security of DSS and our threshold schemes.

Efficiency: We introduce several schemes each of them 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 that the ones encountered in previous work.

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 signature schemes, where in order to achieve a security threshold t we need $2t+1$ active signers during signature computation; hence, achieving optimal thresholds of up to $\frac{n-1}{2}$. This threshold goes down to $\frac{n-1}{3}$ if we allow the possibility of t faulty servers to refuse to participate in the signature protocol (fail-stop faults). In all these cases we improve substantially on the quadratic bound of [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 "correction" capability as in our solution, or at least a detection mechanism for 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.[2] 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 we have two protocols: a more efficient one based on error-correcting codes which achieves $t \leq n/4$ and a less efficient one (based on the techniques of [CMI93]) which achieves $t \leq n/3$. Thus, for the latter case, we present an alternative protocol to [CMI93] that although achieving a smaller threshold provides improved efficiency.[3]

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

[2] 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, GJKR96]).

[3] In [CMI93] a solution that achieves $t \leq n/2$ is also presented. Such solution makes use of advanced techniques from the area of secure multiparty computation ([BGW88, CCD88]). Such techniques are applicable to our schemes as well, however due to the great computational overhead involved with their application the resulting schemes lose practical applicability. For this reason we have decided not to discuss them in detail.

ElGamal based threshold signature schemes (including the DSS solution in [Lan95, CMI93]), but not known for threshold RSA signatures.

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 fixed for a long time while its shares can be refreshed periodically. An adversary that tries to break the threshold signature scheme needs then 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 years) to do so.

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 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). 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, re-randomization 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 $t \leq n/4$ scheme we apply error correcting techniques due to Berlekamp and Welch [BW]. For the $t \leq n/3$ solution we adapt a clever technique from [CMI93] to our scenario.

We prove the security of our schemes assuming the infeasibility of forging DSS signatures.

1.4 Organization

Section 2 introduces 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. Similar definitions can be found in [CMI93].

Communication Model. We assume that our computation model is composed of a set of n *players* $\{P_1, \ldots, P_n\}$ 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 broadcasts a message, it will be recognized by the other players 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. However, it is worth noting that these abstractions can be substituted with standard cryptographic techniques for privacy, commitment and authentication.

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 may *also* cause corrupted players to stop sending messages during the execution of the protocol (e.g., by crashing or disconnecting a machine).
- A *Malicious Adversary* is an eavesdropping adversary that may *also* cause corrupted players to divert from the specified protocol in *any* (possibly 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.)

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

Signature Scheme. A signature scheme $\mathcal{S}$ 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 secret key x, it outputs *sig*, a signature of the message m. 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.

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* $\mathcal{S}$ =(Key-Gen,Sig,Ver) *is* unforgeable *if no adversary who is given the public key y generated by* Key-Gen*, and the signatures of k messages $m_1, \ldots, m_k$ adaptively chosen, can produce the signature on a new message m with non-negligible probability.*

Threshold secret sharing. Given a secret value s we say that the values $(s_1, \ldots, s_n)$ 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 $\mathcal{S}$=(Key-Gen, Sig, Ver) be a signature scheme. A (t, n)-threshold signature scheme $\mathcal{TS}$ for $\mathcal{S}$ is a pair of protocols (Thresh-Key-Gen, Thresh-Sig) for the set of players $\{P_1, \ldots, P_n\}$.

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 $(x_1, \ldots, x_n)$ form a (t, n)-threshold secret sharing of x. The public output of the protocol contains the public key y. The 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 Key-Gen protocol of the regular signature scheme S.

Thresh-Sig is the distributed signature protocol. The private input of P_i is the value x_i. The public inputs consist of a message m and the public key y. The output of the protocol is the value $sig = \mathsf{Sig}(m, x)$. (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 $\mathcal{TS}$ =(*Thresh-Key-Gen,Thresh-Sig*) is* unforgeable, *if no malicious adversary who corrupts at most t players can produce 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 $m_1, \ldots, m_k$ which the adversary adaptively chose.*

This is the analogous to the notion of existential unforgeability under chosen message attack as defined by Goldwasser, Micali, and Rivest [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 part 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]. 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. In other words, 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 $\mathcal{TS}$ =(*Thresh-Key-Gen,Thresh-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 y and m), t shares $x_{i_1}, \ldots, x_{i_t}$ and the signature s of m, can simulate the view of the adversary on an execution of* Thresh-Sig *that generates s as an output.*

This is actually a stronger property than what we need. 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 of $\mathcal{S}$)

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 ($h + c \leq t$ where t is total number of corrupted players).

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

3 The Digital Signature Standard (DSS)

The Digital Signature Standard (DSS) [NIST91] is a signature scheme based on the El-Gamal [ElG85] and Schnorr's [Sch91] signature schemes, which was adopted as the US 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 $512 \leq l \leq 1024$.
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 $1 \leq x < q$.
5. $y = g^x \bmod p$ is the public verification key.

Signature Algorithm. Let m be a hash of the message to be signed. The signer picks a random number k such that $1 \leq k < q$, calculates $k^{-1} \bmod q$, and sets

$$\begin{aligned} r &= (g^{k^{-1}} \bmod p) \bmod q \\ s &= k(m + xr) \bmod q \end{aligned}$$

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

Verification Algorithm. A signature (r, s) of a message m can be publicly verified by checking that $r = (g^{ms^{-1}} y^{rs^{-1}} \bmod p) \bmod q$ where s^{-1} is computed modulo q.

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 notice 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 extra reduction mod q has the only purpose to shorten a DSS signature, but serves no security purpose.

4 Existing Tools

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

Shamir's Secret Sharing. [Sha79]
Given a secret σ, choose at random a polynomial $f(x)$ of degree t, such that $f(0) = \sigma$. Give to player P_i a share $\sigma_i \stackrel{\triangle}{=} f(i) \bmod q$ where q is a prime (We use the interpolation values $i = 1, 2, \ldots, n$ for simplicity; any values in Z_q can be used as well.) We will write $(\sigma_1, \ldots, \sigma_n) \stackrel{(t,n)}{\longleftrightarrow} \sigma \bmod q$ to denote such a sharing. This protocol generates no public output. It can tolerate t eavesdropping faults if $n \geq t + 1$ and, additionally, t halting faults if $n \geq 2t + 1$. By using error-correcting techniques (as first suggested in [MS81]) the protocol can also tolerate f malicious faults (among the players, excluding the dealer) if $n \geq t + 2f + 1$. In the following we will refer to this protocol by Shamir-SS.

Feldman's Verifiable Secret Sharing. [Fel87].
This protocol can tolerate up to $\frac{n-1}{2}$ malicious faults *including the dealer*. Like Shamir's scheme, it generates for each player P_i a share σ_i, such that $(\sigma_1, \ldots, \sigma_n) \stackrel{(t,n)}{\longleftrightarrow} \sigma \bmod q$. If $f(x) = \sum_j a_j x^j$ then the dealer broadcasts the values $\alpha_j = g^{a_j} \bmod p$. This will allow the players to check that the values σ_i really define a secret by checking that $g^{\sigma_i} = \prod_j \alpha_j^{i^j} \bmod p$. It will also allow 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. In the following we will refer to this protocol by Feldman-VSS.

Unconditionally Secure Verifiable Secret Sharing. [FM88, Ped91b].
In contrast to Feldman's VSS protocol, this protocol provides information theoretic secrecy for the shared secret. This is required by some of our techniques in order to achieve provable security. There are two possible implementation of this primitive. One is by Feldman and Micali [FM88] and is based on a bivariate polynomial sharing. Each player receives a share as in Shamir's case plus some extra information that will allow him to check (by exchanging messages with the other

players) that the shares do define a polynomial. This implementation tolerates $\frac{n-1}{3}$ malicious faults. Another possible implementation is the one by Pedersen [Ped91b]. In this implementation the private information of player P_i is the value σ_i such that $(\sigma_1, \ldots, \sigma_n) \overset{(t,n)}{\longleftrightarrow} \sigma \bmod p$, using sharing polynomial f. The dealer then chooses a second t-degree polynomial $f_1 = \sum_j b_j x^j$ and sends the value $\tau_i = f_1(i) \bmod p$ to player P_j. The dealer then commits to each coefficient of the polynomials f, f_1 as follows: he publishes the values $\beta_j = g^{a_j} h^{b_j} \bmod p$ where h is an element in the subgroup generated by g such that the discrete log of h in base g is unknown. This will allow the players to check the received shares by checking that $g^{\sigma_i} h^{\tau_i} = \prod_j \beta_j^{i^j} \bmod p$. At revealing time the players are required to reveal both σ_i and τ_i and the above equation is used to validate the shares. Indeed in order to have an incorrect share σ_i' accepted 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 $\beta_0 = g^{\sigma} h^{b_0}$. The scheme tolerates $\frac{n-1}{2}$ malicious faults. Both implementations can be used in our main protocol. In the following we will refer to this protocol as Uncond-Secure-VSS.

Joint Random Secret Sharing. [Ped91a, Ped91b].
In a Joint Random Secret Sharing 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 $(\sigma_1, \ldots, \sigma_n) \overset{(t,n)}{\longleftrightarrow} \sigma$, 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 every player, and even from any coalition of t players. To realize such a protocol, all players act as dealers of a random local secret that they choose. The final share σ_i $(i = 1, \ldots, n)$ is computed as the sum of the shares dealt to P_i by each player (consequently, the joint secret equals the sum of all dealt secrets). It can be shown that as long as all players correctly share their local secrets and (at least) one of these secrets is chosen randomly then the resultant shares interpolate to a random secret σ. In the cases where active corrupted players, that may deviate from the protocol, are considered, one needs to perform the dealing by each player using a verifiable secret sharing protocol. Shares dealt by players that are detected as corrupted are ignored in the final share computation. The basic properties of this protocol, namely, the kind of public information it generates, and its fault tolerance are inherited from the underlying secret sharing scheme. In the following we will refer to these protocols as Joint-Shamir-RSS, Joint-Feldman-RSS or Joint-Uncond-Secure-RSS depending which of the secret sharing schemes is used.

Joint Zero Secret Sharing. [BGW88]
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 protocol but instead of local random secrets each player deals a sharing of the value zero. When verifiability is required each player deals its shares using Feldman-VSS. The correct dealing of the value zero is verified by checking that the free coefficient p_0 of each dealing polynomials is 0 (i.e., by checking that $g^{p_0} = 1$). We will refer to this protocol as Joint-Zero-SS or Joint-Zero-VSS according to the fact if verification is required or not. Notice that by adding such "zero-shares" to existent shares of a 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-Zero-SS protocol.

Computing reciprocals. In the following we will be faced with the following problem. Given a

secret k mod q which is shared among players $P_1, ...P_n$, generate a sharing of the value k^{-1} mod 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) secret sharing of k, namely, $(k_1, \ldots, k_n) \stackrel{(t,n)}{\longleftrightarrow} k$. 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 any Joint-RSS protocol (Section 4). We denote the resulting shares by $a_1, a_2, \ldots, a_n$, i.e., $(a_1, \ldots, a_n) \stackrel{(t,n)}{\longleftrightarrow} a$.
2. The players execute a $(2t, n)$ Joint-Zero-SS protocol (Section 4) 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 share u_i of k^{-1} by setting $u_i \stackrel{\triangle}{=} \mu^{-1} a_i$ mod q.

We refer to the above protocol as the Reciprocal Protocol. In [BB89] it is proven that such protocol is secure, i.e. correctly computes a sharing of k^{-1} mod 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.

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 re-randomization procedure (using the Joint-Zero-SS protocol of Section 4) is required to protect the secrecy of the multiplied secret; this randomization is essential because a polynomial of degree $2t$ which is the 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 the protocols for the same problem 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. Even the simplified version of Rabin [Rab95] for repetitive multiplication involves non-trivial zero-knowledge proofs for verifiability.)

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, following the protocol of Pedersen [Ped91a]. That is the players run an execution of Joint-Feldman-RSS (Section 4). The output of such a protocol is a secret sharing $(x_1, \ldots, x_n) \stackrel{(t,n)}{\longleftrightarrow} x \bmod q$ of a random value x which in addition reveals $y = g^x \bmod p$. 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. This protocol is based on the one in [CMI93], except that we have removed all the steps needed in order to verify the behavior of the players. As such it holds that:

- It is a secure DSS threshold signature scheme in the presence of an Eavesdropping Adversary (Section 2) when the number of players is $n \geq 2t + 1$ where t is the number of faults.
- It is a secure DSS threshold signature scheme in the presence of a Halting Adversary (Section 2) when the number of players is $n \geq 3t + 1$ where t is the number of faults.

In other words, this protocol preserves security (secrecy and unforgeability) in the presence of less than a half eavesdropping faults. On 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 through a polynomial $F(\cdot)$ of degree t, i.e. $(x_1, \ldots, x_n) \stackrel{(t,n)}{\longleftrightarrow} x \bmod q$. First the players generate distributively a random k (through a random t-degree polynomial $G(\cdot)$) by running the Joint Random Secret Sharing protocol Joint-Shamir-RSS (Section 4). To compute $r = g^{k^{-1}} \bmod p \bmod q$ without revealing k, the players use a variation of the Reciprocal Protocol (Section 4) 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). The full description of protocol DSS-Thresh-Sig-1 is presented in Figure 1. It incorporates the multiplication and reciprocal protocols from Section 4.

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

- $v = \mathsf{Interpolate}(v_1, \ldots, v_n)$. If $\{v_1, \ldots, v_n\}$ $(n \geq 2t + 1)$ 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 $v \stackrel{\triangle}{=} F(0)$. The polynomial can be computed by standard polynomial interpolation.
- $\beta = \mathsf{Exp\text{-}Interpolate}(w_1, \ldots, w_n)$. If $\{w_1, \ldots, w_n\}$ $(n \geq 2t + 1)$ 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 $\beta \stackrel{\triangle}{=} g^{G(0)}$. This can be computed by $\beta = \prod_{i \in V'} w_i^{\lambda_{i,V'}} = \prod_{i \in V'} (g^{G(i)})^{\lambda_{i,V'}}$, 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 on 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* $n \geq 3t+1$.

We need to show now that the protocol is unforgeable. We do this by showing that DSS-Thresh-Sig-1 is simulatable.

A simulator for the protocol is shown on Figure 2. $\mathcal{SIM}$-1 runs on input the public key $y \stackrel{\triangle}{=} g^x$, a message m, its DSS signature (r, s), and the secret values of t players (without loss of generality) $x_1, ..., x_t$. Note that when we say "without loss of generality", there are two issues which are addressed here: *i*) that we are taking the *first* t shares, *ii*) that $\mathcal{SIM}$-1 can not do better if it receives less than t shares. Both these points are easily argued.

Since we are facing an eavesdropping (or halting) adversary, the simulator will just run the entire simulation of the protocol and then present the adversary with its view, i.e. with all the public messages and the messages sent and received by the corrupted players. Accordingly we describe the simulator $\mathcal{SIM}$-1 as a two phase protocol. The first one computes all the necessary information, and in the second phase it carries out the communication with the adversary $\mathcal{A}$ in accordance with protocol DSS-Thresh-Sig-1 .

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.* $\mathcal{VIEW}_{\mathcal{A}}(DSS\text{-Thresh-Sig-}2\ (x_1, ..., x_n, (m, y)) = (r, s))$ *is computationally indistinguishable from* $\mathcal{SIM}\text{-}1(m, (r, s), y, x_1, ..., x_t)$.

Proof: We shall analyze the information generated by the protocol and the simulator in each step (the numbering of steps corresponds to the procedure described as $\mathcal{SIM}$–1–Conversation in Figure 2 and to the steps in protocol DSS-Thresh-Sig-1).

1. Both the protocol and the simulator execute a sharing of a random secret (k and $\hat{k}$, respectively). As the 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 step is secure.

2. The reasoning is similar to the previous step.

3. (a) In the protocol $\mathcal{A}$ receives t shares $a_1, ..., a_t$, which as before are uniformly distributed in $[0..q-1]$. The values $\hat{a}_i$ for $1 \leq i \leq t$ were chosen by $\mathcal{SIM}$-1, under the exact same distribution (Step 4), hence the two distributions are the same.

(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 $1 \leq i \leq t$, satisfies that $v_i = k_i a_i + b_i$. The share $\hat{v}_i$, for $1 \leq i \leq t$ was generated in this manner ($\mathcal{SIM}$–1–Computation Step 5).

Also 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 $g^{\hat{a}}$ are computationally indistinguishable. The rest of the values $g^{\hat{a}_i}$ for $t+1 \leq i \leq n$, are obtained through a deterministic computation from $g^{\hat{a}}$ and $g^{\hat{a}_i}$ for $1 \leq i \leq t$, hence they too are computationally indistinguishable from g^{a_i} for $1 \leq i \leq t$.

4. Same argument as above noting that the shares interpolate the secret s, and that they were properly generated in $\mathcal{SIM}$–1–Computation Step 6

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 $n \geq 2t+1$ ($n \geq 3t+1$)*

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, relies on no assumptions beyond the unforgeability of DSS signatures, and can tolerate $\frac{n-1}{4}$malicious faults.

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 Feldman-VSS ones. Although this allows to tolerate malicious faults it also produced an undesired leaking of information. Indeed notice what happens if the players generate the random value k in step 1, using Joint-Feldman-VSS. The value $g^k \bmod p$ is made public. This is information that the adversary would not get from a traditional DSS signature. So in order to simulate the protocol one has to require yet another extra assumption, namely that if one chooses 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.

Outline. Our aim is to eliminate this extra assumption and 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 an *unconditionally* secure VSS (Section 4). This guarantees that absolutely no information is leaked on the values k or k^{-1}. Then the players compute r as in

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 (Section 4), 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-Zero-SS 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

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

   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 $s \overset{\triangle}{=} \mathsf{Interpolate}(s_1, \ldots, s_n) \bmod q$.

   Public information: s

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

Figure 1: DSS-Thresh-Sig-1 – Halting ($n \geq 3t + 1$) or Eavesdropping ($n \geq 2t + 1$) Adversary

$\mathcal{SIM}$

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

$\mathcal{SIM}$–1–Computation

1. Pick random value $\hat{k}$ uniformly distributed in $[0..q-1]$. Generate sharing using Joint-Shamir-RSS for $\hat{k}$, denote the output of the sharing by $\hat{k}_1,...,\hat{k}_n$.
2. Execute two instances of Joint-Zero-RSS with polynomials of degree $2t$. Set $\hat{b}_i$, $\hat{c}_i$ for $1 \leq i \leq n$ to the output of these invocations.
3. Compute $r^* = g^{ms^{-1}} y^{rs^{-1}} \bmod p$. 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).
4. Choose t random values $\hat{a}_1,...,\hat{a}_t$ uniformly distributed in $[0..q-1]$. Compute $\hat{w}_i = g^{\hat{a}_i}$ for $1 \leq i \leq t$. From the values $g^{\hat{a}}$ and $\hat{w}_1,...,\hat{w}_t$, generate $\hat{w}_i = g^{\hat{a}_i}$ for $t+1 \leq i \leq n$ in such a way that all the $\hat{w}_i$'s interpolate to $g^{\hat{a}}$ "in the exponent". I.e., $\hat{w}_j = g^{\hat{a}_j} = g^{\lambda_{j,0}\hat{a} + \sum_{i=1}^{t} \lambda_{j,i}\hat{a}_i} = (g^{\hat{a}})^{\lambda_{j,0}} g^{\sum_{i=1}^{t} \lambda_{j,i}\hat{a}_i}$ for known values $\lambda_{j,i}$. Perform a Joint-Shamir-RSS such that the first t shares are $\hat{a}_1,\ldots,\hat{a}_t$.
5. Compute $\hat{v}_i \stackrel{\Delta}{=} \hat{a}_i\hat{k}_i + \hat{b}_i$ for $1 \leq i \leq t$. Compute share $\hat{v}_i$ for $t+1 \leq i \leq 2t$, such that $\hat{v}_i$ for $1 \leq i \leq 2t$ define a polynomial $f(x)$ of degree $2t$, such that $f_{\hat{\mu}}(0) = \hat{\mu}$. Complete the shares $\hat{v}_i$ for $2t+1 \leq i \leq n$ so that $\hat{v}_i \stackrel{\Delta}{=} f_{\hat{\mu}}(i)$.
6. Compute $\hat{s}_i \stackrel{\Delta}{=} \hat{k}_i(m + x_i r) + \hat{c}_i$ for $1 \leq i \leq t$. Compute share $\hat{s}_i$ for $t+1 \leq i \leq 2t$, such that $\hat{s}_1,...,\hat{s}_{2t}$ define a polynomial $f_{\hat{s}}(x)$ of degree $2t$, such that $f_{\hat{s}}(0) = s$. Complete the shares $\hat{s}_i$ for $2t+1 \leq i \leq n$ so that $\hat{s}_i \stackrel{\Delta}{=} g(i)$.

$\mathcal{SIM}$–1–Conversation

Comment: In each of the following steps we describe the information which $\mathcal{SIM}$ gives to $\mathcal{A}$. Each of these steps relates to the same numbered step in protocol DSS-Thresh-Sig-2.

1. The first t shares of each of the n sharings in the Joint-Shamir-SS. In particular $\mathcal{A}$ knows $\hat{k}_1,...,\hat{k}_t$
2. The first t shares of each of the $2n$ sharings in the Joint-Zero-SS. In particular $\mathcal{A}$ knows shares $\hat{b}_i$, $\hat{c}_i$ for $1 \leq i \leq t$
3. (a) The first t shares of each of the n sharings of the Joint-Shamir-RSS. In particular $\mathcal{A}$ knows shares $\hat{a}_1,...,\hat{a}_t$.
   (b) Public values $\hat{v}_1,...,\hat{v}_n$, $g^{\hat{a}} = (r^*)^{\hat{\mu}}$ and $\hat{w}_1,\ldots,\hat{w}_n$.
   (c) twiddles his thumbs
4. public values: $\hat{s}_1,...,\hat{s}_n$

Figure 2: Simulation Protocol for DSS-Thresh-Sig-1

DSS-Thresh-Sig-1, with the only difference that now the random value a is jointly generated using Feldman's VSS protocol. As before s is computed from the appropriate shares and randomization of polynomials (through the joint zero secret sharing protocols) is added in various places in order to hide possible partial information. Another difference between our protocol and the [CMI93] one is that whenever we reconstruct a secret, in order to detect bad shares contributed by malicious players we perform error-correcting using the Berlekamp and Welch decoder [BW]. This will allow us to obtain $\frac{n-1}{4}$fault-tolerance. Although this is slightly worse than [CMI93] (they obtain $\frac{n-1}{3}$ fault-tolerance), the use of error-correcting codes produces a much more efficient protocol. In the next section we will show how to adapt the techniques of [CMI93] to our protocol in order to obtain $\frac{n-1}{3}$ fault-tolerance as well. The full protocol is exhibited in Figure 3.

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 $v \stackrel{\triangle}{=} F(0)$. The polynomial can be computed by using the Berlekamp-Welch decoder [BW].

We prove the following theorem:

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 Malicious Adversary, when the number of players 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 is $deg + 2faults + 1 = 4t + 1$. Hence, we set $n \geq 4t + 1$. □

As in the previous section in order to prove the unforgeability of our protocol, we shall present a simulator $\mathcal{SIM}$-2 which on input the public key $y \stackrel{\triangle}{=} g^x$, a message m, its DSS signature (r, s), and the secret values of t players (without loss of generality) $x_1, ..., x_t$, can generate for the adversary a view of the protocol which is computationally indistinguishable from the actual view of the execution of the protocol.

Since we are facing a malicious adversary, the simulator has to actually interact with the adversary. This means that the simulator will actually run a simulated version of the protocol for the adversary, running all the $n - t$ good players. $\mathcal{SIM}$-2 is described in Figure 4.

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 $(k_1, \ldots, k_n) \stackrel{(t,n)}{\longleftrightarrow} k \bmod q$.

Secret information of P_i : a share k_i of k

2. **Generate random polynomials with constant term 0**
   Execute two instances of Joint-Zero-VSS with polynomials of degree $2t$ as underlying scheme. 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 Public information: $g^0 = 1,\ g^{b_i},\ g^0 = 1,\ g^{c_i},\ 1 \le i \le n$

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-Feldman-RSS.

Secret information of P_i : a share a_i of a Public information: $g^a,\ g^{a_i},\ 1 \le i \le n$

   (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: $v_1, ..., v_n$ 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

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

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

   Set $s \stackrel{\triangle}{=} \text{EC-Interpolate}(s_1, \ldots, s_n)$.

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

Figure 3: DSS - Distributed signature generation - Malicious Adversary, $n \ge 4t + 1$

$\mathcal{SIM}$-2

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

1. $\mathcal{SIM}$-2 shares a random value uniformly distributed in Z_q for each good player using Uncond-Secure-VSS. It also listens to the sharings done by the adversary. Let $\hat{k}$ be the resulting value of the associated Joint-Uncond-Secure-RSS thus performed. Notice that $\hat{k}$ is uniformly distributed in Z_q and also is known to $\mathcal{SIM}$-2 (since he holds more than t players).

2. As above $\mathcal{SIM}$-2 runs its part in the two instances of Joint-Zero-VSS. Set $\hat{b}_i$, $\hat{c}_i$ $g^{\hat{b}_i}$ and $g^{\hat{c}_i}$ 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 then runs its part for the good players in the Join-Fledman-VSS protocol. Let $\hat{a}$ be the resulting shared value (known to $\mathcal{SIM}$-2). If $(r^*)^{\hat{\mu}} = g^{\hat{a}}$ continue to the next step.

   Otherwise reset $\mathcal{A}$ and start the whole simulation from scratch using the same coin tosses (which means that all the messages up to this point will be the same.)

   When the repeated simulation comes back to this step, $\mathcal{SIM}$-2 knows in advance what values will be shared by $\mathcal{A}$ for the Joint-Feldman-VSS. $\mathcal{SIM}$-2 then chooses its shared value so that the value $g^{\hat{a}}$ equals $(r^*)^{\hat{\mu}}$. (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).

   This is done by first fixing the t shares of the adversary $\hat{a}_1, \ldots, \hat{a}_t$ at random, and then using the values $(r^*)^{\hat{\mu}} = g^{\hat{a}}$ and $g^{\hat{a}_i}$ perform an interpolation "in the exponent" to obtain the other public information of Joint-Feldman-VSS. I.e. choose t random values $\hat{a}_1, ..., \hat{a}_t$ uniformly distributed in $[0..q-1]$. Compute $g^{\hat{a}_i}$ for $1 \leq i \leq t$. From the values $g^{\hat{a}}$ and $\hat{a}_1, ..., \hat{a}_t$, generate $g^{\hat{a}_i}$ for $t+1 \leq i \leq n$. Note that $g^{\hat{a}_j} = g^{\lambda_{j,0}\hat{a} + \sum_{i=1}^{t} \lambda_{j,i}\hat{a}_i} = (g^{\hat{a}})^{\lambda_{j,0}} g^{\sum_{i=1}^{t} \lambda_{j,i}\hat{a}_i}$ for known values $\lambda_{j,i}$.

   (b) Compute $\hat{v}_i \stackrel{\triangle}{=} \hat{a}_i\hat{k}_i + \hat{b}_i$ for $1 \leq i \leq t$. Compute share $\hat{v}_i$ for $t+1 \leq i \leq 2t$, such that $\hat{v}_i$ for $1 \leq i \leq 2t$ define a polynomial $f_{\hat{\mu}}(x)$ of degree $2t$, such that $f_{\hat{\mu}}(0) = \hat{\mu}$. Complete the shares $\hat{v}_i$ for $2t+1 \leq i \leq n$ so that $\hat{v}_i \stackrel{\triangle}{=} f_{\hat{\mu}}(i)$. Broadcast the values $\hat{v}_i$ for $t+1 \leq i \leq n$, i.e. the good players.

   (c) Twiddles his thumbs.

4. Compute $\hat{s}_i \stackrel{\triangle}{=} \hat{k}_i(m + x_i r) + \hat{c}_i$ for $1 \leq i \leq t$. Compute share $\hat{s}_i$ for $t+1 \leq i \leq 2t$, such that $\hat{s}_1, ..., \hat{s}_{2t}$ define a polynomial $f_s(x)$ of degree $2t$, such that $f_s(0) = s$. Complete the shares $\hat{s}_i$ for $2t+1 \leq i \leq n$ so that $\hat{s}_i \stackrel{\triangle}{=} f_s(i)$. Broadcast the values s_i for $i = t+1, \ldots, n$, i.e. for the good players.

Figure 4: 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. $\mathcal{VIEW}_{\mathcal{A}}(DSS\text{-Thresh-Sig-}2\ (x_1, ..., x_n, (m, y)) = (r, s))$ is computationally indistinguishable from $\mathcal{SIM}\text{-}2(m, (r, s), y, x_1, ..., x_t)$.*

Proof: We shall exhibit the proof by analyzing the information generated by the protocol and the simulator in each step (the numbering of steps corresponds to the procedure described in Figure 4 and to the steps in protocol DSS-Thresh-Sig-2).

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 probability. Thus, the sharings of two (possibly) different secrets, generate the same distribution for the sets of size t plus the public information. As 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 non-random 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, but here the sharing is of a zero value, and the attached verifiability procedure is computationally secure. The simulatability of this step follows from the simulatability of Joint-Feldman-RSS.

3. (a) In the protocol $\mathcal{A}$ receives t shares $a_1, ..., a_t$ of a proper sharing including g^a and g^{a_i} for $1 \leq i \leq n$. As before a_i for $1 \leq i \leq t$ is uniformly distributed in $[0..q-1]$. The values $\hat{a}_i$ for $1 \leq i \leq t$ were choosen by $\mathcal{SIM}$-2, under the exact same distribution (Step 3a), hence the two distributions are the same. 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 $g^{\hat{a}}$ are computationally indistiguishable. The rest of the values $g^{\hat{a}_i}$ for $t+1 \leq i \leq n$, are obtained through a deterministic computation from $g^{\hat{a}}$ and $g^{\hat{a}_i}$ for $1 \leq i \leq t$, hence they too are compuationally indistinguishable from g^{a_i} for $1 \leq i \leq t$.

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

4. 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. □

8 Malicious Adversary, $n \geq 3t + 1$

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

The previous protocol used the Feldman and Pedersen VSS protocols [Fel87, Ped91b] only to assure that secrets where shared correctly. When it came to authenticate the shares broadcasted by players in order to reconstruct a value, however we relied on error-correcting codes. 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 Feldman-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 linear combination of secrets. For example, we know that if $(a_1, \ldots, a_n) \overset{(t,n)}{\longleftrightarrow} a$ and $(b_1, \ldots, b_n) \overset{(t,n)}{\longleftrightarrow} b$ then $(a + b_1, \ldots, a + b_n) \overset{(t,n)}{\longleftrightarrow} a + b$. 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 $g^{a_i b_i}$ 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 Feldman-VSS. However there is also an advantage: at least one of the values muiltiplies is *random* and has been "recently" shared using a Joint-Feldman-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 Feldman-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 a general fashion. Let $(a_1, \ldots, a_n) \overset{(t,n)}{\longleftrightarrow} a$ be the the sharing obtained ¿from a run of Feldman-VSS. Let $A(x) = A_0 + A_1 x + \ldots + A_t x^t$ be the t-degree polynomial used to share the value $a = A_0$. Player P_i holds share $a_i = A(i) \bmod p$ and the values $\alpha_j = g^{A_j} \bmod p$ are publicly know. We also assume that we have performed a Joint-Zero-VSS with a $2t$-degree polynomial $B(x) = B_1 x + \ldots + B_{2t} x^{2t}$. Player P_i holds value $b_i = B(i) \bmod p$ and the values $\beta_j = g^{B_j} \bmod p$ are publicly known.

Now assume that the players together generate the sharing of a random value k using Joint-Uncond-Secure-RSS. This means that each player P_i runs one instance of Uncond-Secure-VSS using a random value $k^{(i)}$ as the secret and a random t-degree polynomial $K^{(i)}(x) = K_0^{(i)} + K_1^{(i)} x + \ldots + K_t^{(i)} x^t$. Let $K(x) = \sum_i K^{(i)}(x)$.

Consider now the randomized sharing polynomial of the product secret ar. This the polynomial

$$C(x) = A(x)K(x) + B(x) = B(x) + \sum_i A(x)K^{(i)}(x) = B(x) + \sum_i C^{(i)}(x) \tag{1}$$

if one lets $C^{(i)}(x) = A(x)K^{(i)}(x)$. Let

$$C(x) = C_0 + C_1 x + \ldots + C_{2t} x^{2t}$$

and

$$C^{(i)}(x) = C_0^{(i)} + C_1^{(i)} x + \ldots + C_{2t}^{(i)} x^{2t}$$

If we could make the values $\gamma_j = g^{C_j} \bmod p$ public, then we could sieve out bad shares using Feldman's procedure. Using Equation (1) we see that such γ_j's values would be known if each player P_i broadcasts the values

$$g_0^{C^{(i)}}, g_1^{C^{(i)}}, \ldots, g_{2t}^{C^{(i)}} \bmod p$$

which he can easily compute from the values α_j and $C_k^{(i)}$. The broadcasted values can be checked by the other players using the usual Feldman's verification procedure (i.e., check that the point on the polynomial they own, does indeed interpolate in the exponent to the values broadcasted by P_i.)

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 to the above method to our construction of a DSS signature generation protocol is almost immediate. Look at our protocol DSS-Thresh-Sig-2. The above method needs to be used when we compute the products ka (round 3c) and kx (round 4) instead of the error-correcting mechanism.

The drawback is that while error-correction is fast, this method requires several extra modular exponentiations (see next section.)

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 modular exponentiations required by a single player during the execution of the protocol.

A close analysis of the three protocols presented in the previous sections reveals that:
DSS-Thresh-Sig-1 requires $2t + 3$ modular exponentiations
DSS-Thresh-Sig-2 requires $O(nt)$ modular exponentiations (the constant hidden in the O notation is very small)
DSS-Thresh-Sig-3 requires also $O(nt)$ modular exponentiations, but the constant is larger (approximately twice as the number of exponentiations required in DSS-Thresh-Sig-2).

In the case of the generation of regular DSS signatures, the most expensive part is the computation of r, as it involves modular exponentiations. This property is preserved in our DSS-Thresh-Sig-1 and DSS-Thresh-Sig-2 protocols. All the modular exponentiations and message exchanges happen during the computation of r. Thus such computation can be moved off-line and the computation of the actual on-line signature is very fast and non-interactive. In the case of DSS-Thresh-Sig-3 the computation of s still involves $O(nt)$ modular exponentiations per player.

This evidentiates the advantage of our error-correction approach over the [CMI93] approach. The improvement in fault-tolerance has to be measured against a total doubling of computational effort and the loss of the on-line efficiency typical of DSS signatures.

10 Acknowledgments

The authors wish to thank Matthias Birkner for pointing the difference between r and r^* in the simulation of the protocols. Also the authors thank Don Beaver and Kaoru Kurosawa for useful pointers to references.

References

[BB89] J. Bar-Ilan, and D. Beaver. Non-Cryptographic Fault-Tolerant Computing in a Constant Number of Rounds. In *Proceedings of the ACM Symposium on Principles of Distributed Computation*, pp.201–209, 1989.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Noncryptographic Fault-Tolerant Distributed Computations. In *Proceeding 20th Annual Symposium on the Theory of Computing*, pages 1–10. ACM, 1988.

[Boy86] C. Boyd. Digital Multisignatures. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 241–246. Claredon Press, 1986.

[BW] E. Berlekamp and L. Welch. Error correction of algebraic block codes. US Patent 4,633,470.

[CCD88] D. Chaum, C. Crepeau, and I. Damgard. Multiparty Unconditionally Secure Protocols. In *Proceeding 20th Annual Symposium on the Theory of Computing*, pages 11–19. ACM, 1988.

[CMI93] M. Cerecedo, T. Matsumoto, and H. Imai. Efficient and Secure Multiparty Generation of Digital Signatures Based on Discrete Logarithms. *IEICE Trans. Fundamentals*, E76-A(4):532–545, April 1993.

[DDFY94] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 522–533, Santa Fe, 1994. IEEE.

[Des88] Y. Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *Proc. CRYPTO 87*, pages 120–127. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 293.

[Des94] Y.G. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July 1994.

[DF90] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *Proc. CRYPTO 89*, pages 307–315. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[DF92] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 457–469. Springer, 1992. Lecture Notes in Computer Science No. 576.

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

[Fel87] P. Feldman. A Practical Scheme for Non-Interactive Verifiable Secret Sharing. In *Proceeding 28th Annual Symposium on the Foundations of Computer Science*, pages 427–437. IEEE, 1987.

[FM88] P. Feldman and S. Micali. An Optimal Algorithm for Synchronous Byzantine Agreement. In *Proceeding 20th Annual Symposium on the Theory of Computing*, pages 148–161. ACM, 1988.

[FGY96] Y. Frankel, P. Gemmell, and M. Yung. Witness-based Cryptographic Program Checking and Robust Function Sharing. In *Proceedings of the ACM Symposium on Theory of Computing*, 1996.

[GJKR96] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and Efficient Sharing of RSA Functions. In *Advances in Cryptology–CRYPTO'96*, Lecture Notes in Computer Science vol.1109, pp.157–172, Springer-Verlag, 1996

[GMR88] S. Goldwasser, S. Micali, and R.L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM. J. Computing*, 18(1):186–208, February 1989.

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

[HJKY95] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing, or: How to cope with perpetual leakage. In *Advances in Cryptology–Crypto'95*, Lecture Notes of Computer Science vol.963, Springer-Verlag, 1995.

[HJJKY97] A. Herzberg, M. Jakobbson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public–key and signature schemes. In *Proceedings of the ACM Conference on Computer and Communication Security*, to appear, 1997.

[Lan95] S. Langford. Threshold DSS signatures without a trusted party. In *Crypto'95*, pages 397–409. Springer-Verlag, 1995. Lecture Notes in Computer Science No. 963.

[MR92] S. Micali and P. Rogaway. Secure computation. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 392–404. Springer, 1992. Lecture Notes in Computer Science No. 576.

[MS81] R. McEliece and D. Sarwate. On Sharing Secrets and Reed-Solomon codes. *Communications of the ACM*, 24(9):583–584, September 1981.

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

[PK96] C. Park, and K. Kurosawa. New ElGamal Type Threshold Digital Signature Scheme. *IEICE Trans. Fundamentals*, E79-A(1):86–93, January 1996.

[Ped91a] T. Pedersen. Distributed provers with applications to undeniable signatures. In *Proc. EUROCRYPT 91*, 1991.

[Ped91b] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proc. CRYPTO 91*, pages 129–140, 1991.

[Rab95] M. Rabin. A Simplification Approach to Distributed Multiparty Computations. personal communication, 1995.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Sch91] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991.

[Sha79] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22:612–613, 1979.