Robust Threshold DSS Signatures

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

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.

Our results significantly improve over a recent result by Langford from CRYPTO'95 that presents threshold DSS signatures which can stand much smaller subsets of corrupted players, namely, $t \approx \sqrt{n}$, and do not enjoy the robustness property. As in the case of Langford's result, our schemes require no trusted party. Our techniques apply to other threshold ElGamal-like signatures as well. We prove the security of our schemes 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: {rosario,stasio,talr}@theory.lcs.mit.edu

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

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.

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. Particular examples of solutions to threshold signatures can be found in [DF92, DDFY94] for the case of RSA signatures, and [Har94, Lan95] for ElGamal-type of signatures.

In this work we present a threshold signature system for DSS, the Digital Signature Standard [fST91]. The importance of providing threshold solutions for signatures schemes used in practice, is that those systems are the ones that will be deployed in the real world and hence they are the ones that require real protection. Threshold DSS signatures schemes were recently studied by Langford [Lan95]. DSS signatures turn out to be less amenable to sharing techniques than RSA or even other ElGamal-type of signatures, e.g., see [Har94]. Langford has overcome some of these difficulties in the case of DSS, 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 servers this solution can resist up to $\sqrt{n}$ corrupted parties.

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). 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 thresholds of up to $\frac{n-1}{2}$. In addition, we improve on [Lan95] by providing 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

[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.

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. 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 either refusing to cooperate ($t \leq n/3$ in this case) or by behaving in any arbitrary malicious way (in this case $t \leq n/4$.)[2]

Moreover, our schemes do not require trusting any particular party at any time, including the initial secret key generation. This is an important property achieved by some other ElGamal based threshold signature schemes (including the DSS solution in [Lan95]), but not known for threshold RSA signatures. In the complete version of the paper we will present some additional results, including the application of our techniques to solving threshold signatures for other discrete–log based signatures [ElG85, NR94, HPM94].

Remarkably, our solutions for robust threshold DSS signatures can be *proactivized* using the recent techniques of [HJJ$^+$95] (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.

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). Langford [Lan95] solves both problems by presenting a multiplicative version of secret sharing that results in polynomials of degree $O(t^2)$; this requires a high number of active signers for signature computation and allows for only a small threshold. In our case, 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 introduce a simple and novel solution, which does not incur any additional increase in the number of signers. The solution to this problem is of independent interest and has applications to other threshold ElGamal-like signatures. 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. For achieving the robustness of our schemes we apply error correcting techniques due to Berlekamp and Welch [BW] that achieve a very high rate of error correction, which in our scenario translates into supporting higher thresholds. We prove the security of our schemes assuming the infeasibility of forging a regular DSS signature. That is, our schemes are secure if and only if DSS is unforgeable.

[2] The robustness property has been known for some other shared signature schemes, e.g., Harn's solution [Har94] for threshold AMV-signatures enjoys this property. As for threshold RSA, robust solutions have been only recently found (see [FGY96, GJKR96]).

2 The Digital Signature Standard (DSS)

The Digital Signature Standard (DSS) [fST91] 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 [fST91] 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. $y = g^x \mod p$ 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 $k^{-1} \mod q$, and sets

$$r = (g^{k^{-1}} \mod p) \mod q$$
$$s = k(m + xr) \mod q$$

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

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

3 Model and Definitions

In this section we introduce our communication model and provide definitions of secure threshold signature schemes.

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. 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.

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 1. 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]. 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 2. 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 3. 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.

A complete formalization of the definition of secure threshold signature schemes can be found in [Gen96].

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 $\sigma_i \triangleq f(i) \bmod q$ 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 $(\sigma_1, \ldots, \sigma_n) \overset{(t,n)}{\longleftrightarrow} \sigma \bmod q$ 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 $(\sigma_1, \ldots, \sigma_n) \overset{(t,n)}{\longleftrightarrow} \sigma \bmod q$. 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 $g^{\sigma_i} = \prod \alpha_j^{i^j}$. 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 $(\sigma_1, \ldots, \sigma_n) \overset{(t,n)}{\longleftrightarrow} \sigma \bmod p$. The dealer then commits to each share using an unconditionally secure commitment scheme based on the hardness of discrete log (that is the secrecy of the committed value is unconditional, but it is possible to open the commitment in a different way if one is able to solve discrete log.) The commitment has homomorphic properties that allow the players to check that the shares define a secret as in Feldman's VSS. If one assumes that players are not able to open the commitment in different ways, then at reconstruction time bad shares are detected. 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. 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. 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.

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 [fST91]), or using randomness which is jointly provided by the trustees. 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 $(x_1, \ldots, x_n) \overset{(t,n)}{\longleftrightarrow} x \bmod q$ 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 Basic Modules of our Solution

We start by introducing two building blocks which are central to our solution for threshold DSS signatures. The first is an elegant and simple procedure for the shared computation of reciprocals. This procedure is used in our protocols in the following way: the players first produce a joint sharing of a random k, and then compute from these shares a sharing of the reciprocal k^{-1}; the latter in turn are used to compute $r = g^{k^{-1}}$

(it is essential for the security of our application that information on k is not revealed during this process).

Problem 1: Computing reciprocals
Given a secret $k \bmod q$ which is shared among players $P_1, \ldots P_n$, generate a sharing of the value $k^{-1} \bmod q$, without revealing information on k and k^{-1}.

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 $u_i \triangleq \mu^{-1} a_i \bmod q$.

We refer to the above protocol as the Reciprocal Protocol. The following lemmas can be proven concerning this protocol.

Lemma 4. *It holds that* $(u_1, \ldots, u_n) \stackrel{(t,n)}{\longleftrightarrow} k^{-1}$.

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. This property is stated in the following lemma.

Lemma 5. *(Informal) There exists a simulator $\mathcal{SIM}$ such that for any adversary $\mathcal{A}$ with access to t shares $k_{i_1}, \ldots, k_{i_t}$ of k, $\mathcal{VIEW}_{\mathcal{A}}$(Reciprocal-Protocol$(k_1, \ldots, k_n)$) is computationally indistinguishable from $\mathcal{SIM}(k_{i_1}, \ldots, k_{i_t})$.*

The proofs of the above lemmas are omitted here as they are implicit in the proofs of our protocols.

Problem 2: 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.)

7 DSS-Thresh-Sig-1: Eavesdropping & Halting Adversary

In this section we present our basic protocol for generating a distributed DSS signature.

- It is a secure DSS threshold signature scheme in the presence of an Eavesdropping Adversary (Section 3) 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 3) 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 $r = g^{k^{-1}} \bmod p$ without revealing k, the players use a variation of the Reciprocal Protocol (Section 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 6). The full description of protocol DSS-Thresh-Sig-1 is presented in Figure 1. It incorporates the multiplication and reciprocal protocols from Section 6.

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

- $v =$ 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.
- $\beta =$ Exp-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.

Lemma 6. *DSS-*Thresh-Sig*-1 is a simulatable (in particular unforgeable) threshold DSS signature generation protocol in the presence of up to t eavesdropping faults, where the total number of players is $n \geq 2t+1$.*

Lemma 7. *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 $n \geq 3t+1$.*

The proofs of these lemmas follow the same lines of the proof of Theorem 9 in Section 8. From the above lemmas we derive the following:

Theorem 8. *DSS-*Thresh-Sig*-1 is a secure, i.e. robust and unforgeable, threshold DSS signature in the presence of t eavesdropping (halting) faults if the total number of players is $n \geq 2t+1$ ($n \geq 3t+1$)*

8 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 regular DSS signatures, and can tolerate $\frac{n-1}{4}$ malicious faults.

Outline. The protocol is very similar to DSS-Thresh-Sig-1. The only difference is that here we need verifiable sharing of secrets since we assume a Malicious Adversary. 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-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. 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]. As before randomization of polynomials (through the joint zero secret sharing protocols) is added in various places in order to hide possible partial information. The full protocol is exhibited in Figure 2

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

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

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].

An important technical contribution of our paper is the simulation and the proof of the security of this protocol. We prove the following theorem:

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

1. **Generate k**
   The trustees 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 T_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 T_i : shares b_i, c_i

3. **Compute** $r = g^{k^{-1}} \bmod p \bmod q$
   (a) The trustees 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 T_i : share a_i of a

   (b) Trustee T_i broadcasts $v_i = k_i a_i + b_i \bmod q$ and $w_i = g^{a_i} \bmod p$. If T_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) Trustee T_i locally computes
   - $\mu \triangleq$ Interpolate$(v_1, \ldots, v_n) \bmod q$ $\quad [= ka \bmod q]$
   - $\beta \triangleq$ Exp-Interpolate$(w_1, \ldots, w_n) \bmod p$ $\quad [= g^a \bmod p]$
   - $r \triangleq \beta^{\mu^{-1}} \bmod p \bmod q$ $\quad [= (g^a)^{\mu^{-1}} = g^{k^{-1}} \bmod p \bmod q]$

   Public information: r

4. **Generate** $s = k(m + xr) \bmod q$
   (a) Trustee T_i broadcasts $s_i = k_i(m + x_i r) + c_i \bmod q$. If T_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 trustee computes $s \triangleq$ Interpolate$(s_1, \ldots, s_n) \bmod q$.

   Public information: s

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

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

Theorem 9. *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* $n \geq 4t + 1$.

It is important to note that unforgeability is obtained for $n \geq 2t + 1$ (see Lemma 11), while $n \geq 4t + 1$ is needed only for robustness (see Lemma 10.)

Lemma 10. *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 $n \geq 4t + 1$.

In order to prove the unforgeability of our protocol, we shall generate a simulator $\mathcal{SIM}$-2 which on input the public key $y \stackrel{\triangle}{=} g^x$, a message m and its signature (r, s), 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. 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 $\mathcal{SIM}$ can not do better if it receives less than t shares. Both these points are easily argued.
The simulator $\mathcal{SIM}$-2 is 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. The input to the second protocol is the output of the first one. $\mathcal{SIM}$-2 is described in Figure 3.

Lemma 11. *Fix a Malicious adversary* $\mathcal{A}$*. where t is the number of faults.* $\mathcal{VIEW}_{\mathcal{A}}(DSS$-Thresh-Sig*-2* $(x_1, ..., x_n, (m, y)) = (r, s))$ *is computationally indistinguishable from* $\mathcal{SIM}(m, (r, s), y, x_1, ..., x_t)$.

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 as $\mathcal{SIM}$–Conversation in Figure 3 and to the steps in protocol DSS-Thresh-Sig-2).

1. Both the protocol and the simulator execute a sharing of a random secret. 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, 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.

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

1. **Generate** k
   The trustees 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 T_i : a share k_i of k

2. **Generate random polynomials with constant term 0**
   Execute two instances of Joint-Zero-SS with polynomials of degree $2t$ 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 T_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 T_i : a share a_i of a
Public information: $g^a,\ g^{a_i},\ 1 \le i \le n$

   (b) Trustee T_i broadcasts $v_i = k_i a_i + b_i \bmod q$. If T_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) Trustee T_i computes locally
   - $\mu \triangleq$ EC-Interpolate$(v_1, \ldots, v_n) \bmod q$ $\quad [= ka \bmod q]$
   - $\mu^{-1} \bmod q$ $\quad [= k^{-1}a^{-1} \bmod q]$
   - $r \triangleq (g^a)^{\mu^{-1}} \bmod p \bmod q$ $\quad [= g^{k^{-1}} \bmod p \bmod q]$

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

Public information: r

4. **Generate** $s = k(m + xr) \bmod q$
   Trustee T_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 $s \triangleq$ EC-Interpolate$(s_1, \ldots, s_n)$.
5. **Output** the pair (r, s) as the signature for m

Fig. 2. DSS - Distributed signature generation - Malicious Adversary, $n \ge 4t + 1$

$\mathcal{SIM}$

Input: public key y, message m, signature (r, s), shares $x_1, \ldots, x_t$

$\mathcal{SIM}$–Computation

1. Pick random value $\hat{k}$ uniformly distributed in $[0..q-1]$. Generate sharing using unconditionally secure VSS for $\hat{k}$, denote the output of the sharing by $\hat{k}_1, \ldots, \hat{k}_n$.
2. Execute two instances of Joint-Zero-SS with polynomials of degree $2t$. Set $\hat{b}_i,\ \hat{c}_i\ g^{\hat{b}_i}$ and $g^{\hat{c}_i}$ for $1 \le i \le n$ to the output of these invocations.
3. 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 $\hat{a}_1, \ldots, \hat{a}_t$ uniformly distributed in $[0..q-1]$. Compute $g^{\hat{a}_i}$ for $1 \le i \le t$. From the values $g^{\hat{a}}$ and $\hat{a}_1, \ldots, \hat{a}_t$, generate $g^{\hat{a}_i}$ for $t+1 \le i \le n$. 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 $\lambda_{j,i}$.
5. Compute $\hat{v}_i \triangleq \hat{a}_i\hat{k}_i + \hat{b}_i$ for $1 \le i \le t$. Compute share $\hat{v}_i$ for $t+1 \le i \le 2t$, such that $\hat{v}_i$ for $1 \le i \le 2t$ define a polynomial $f(x)$ of degree $2t$, such that $f(0) = \hat{\mu}$. Complete the shares $\hat{v}_i$ for $2t+1 \le i \le n$ so that $\hat{v}_i \triangleq f(i)$.
6. Compute $\hat{s}_i \triangleq \hat{k}_i(m + x_i r) + \hat{c}_i$ for $1 \le i \le t$. Compute share $\hat{s}_i$ for $t+1 \le i \le 2t$, such that $\hat{s}_1, \ldots, \hat{s}_{2t}$ define a polynomial $g(x)$ of degree $2t$, such that $g(0) = s$. Complete the shares $\hat{s}_i$ for $2t+1 \le i \le n$ so that $\hat{s}_i \triangleq g(i)$.

$\mathcal{SIM}$–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. shares $\hat{k}_1, \ldots, \hat{k}_t$
2. shares $\hat{b}_i,\ \hat{c}_i$ for $1 \le i \le t$
public values $g^0 = 1,\ g^{\hat{b}_i},\ 1 \le i \le n$
$g^0 = 1,\ g^{\hat{c}_i},\ 1 \le i \le n$
3. (a) shares $\hat{a}_1, \ldots, \hat{a}_t$
public values $g^{\hat{a}}$ and $g^{\hat{a}_i}$ for $1 \le i \le n$
(b) public values $\hat{v}_1, \ldots, \hat{v}_n$
(c) twiddles his thumbs
4. public values: $\hat{s}_1, \ldots, \hat{s}_n$

Fig. 3. Simulation Protocol for DSS-Thresh-Sig-2

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 $\mathcal{SIM}$, under the exact same distribution (Step 4), 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 ($\mathcal{SIM}$-Computation Step 5).
4. Same argument as above noting that the shares interpolate the secret s, and that they were properly generated in $\mathcal{SIM}$-Computation Step 6

This completes the proof of Lemma 11.

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

We have also devised a DSS distributed signature generation protocol which is secure in the presence of a Malicious Adversary when $n \geq 3t+1$ where t is the number of faults. In other words, it is secure against an adversary who can corrupt at most a third of the players and can make them deviate arbitrarily from their prescribed instructions. For lack of space we present only an outline of the protocol. The details will appear in the complete version of the paper.

However, this algorithm is provably secure only under the following assumption: let p be a prime of the form $p = kq+1$ where q is another large prime and g an element of order q in Z_p^*. Let G be the subgroup generated by g.

Conjecture 1 *Choose u, v at random, uniformly and independently in Z_q. The following probability distributions on $G \times G$, $(g^u \bmod p,\ g^v \bmod p)$ and $(g^u \bmod p,\ g^{u^{-1}} \bmod p)$ are computationally indistinguishable.*

In other words, we assume that for random u, the value g^u reveals no computational information on the value $g^{u^{-1}}$.

Outline. This protocol differs from the DSS-Thresh-Sig-2 by more extensive use of Feldman-type verifiability instead of using unconditioanlly secure VSS and error-correcting codes. This shift allows for achieving robustness in the presence of larger number of malicious faults (a third instead of one fourth). The random value k is distributively generated using Feldman's VSS. Notice that this expose the value g^k which is extra information that the adversary would not receive from a regular DSS signature. However if Conjecture 1 holds we can claim that this knowledge would not help an adversary in forging signatures (indeed if it did, such an adversary could be used to distinguish between "reciprocals in the exponent" – where k replaces the value u in the conjecture.) A difficulty arises when in the protocol we need to reveal the product of two secrets (i.e., when using the Multiplication Protocol of section 6). In this case, the public information of Feldman's VSS is not enough to detect faulty players who reveal incorrect multiplication shares. In order to overcome this difficulty we require the players to perform Chaum's zero–knowledge proof of equality of discrete-logs [Cha90] (originally designed in the context of undeniable signatures). The basic idea is that if two secrets a and b are shared with Feldman's VSS, then each player has a share $c_i = a_i b_i$ of $c = ab$. However if we want to reconstruct c, we cannot sieve out bad shares as in Feldman, since we do not have the values g^{c_i} but only g^{a_i} and g^{b_i}. So we require each player to publish $g^{a_i b_i}$ and prove using Chaum's proof that $DL_g(g^{a_i}) = DL_{g^{b_i}}(g^{a_i b_i})$. As before, randomization of polynomials is added when needed in order to protect partial information.

10 Efficiency Considerations

As in the case of the generation of regular DSS signatures the most expensive part of our protocols is the computation of r, as it includes all the modular exponentiations and the interactive exchange of messages between players. However (as in the case of regular DSS signatures) such computation can be performed off-line. In this case the signature generation becomes extremely efficient and non-interactive.

References

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-cryptographic Fault-Tolerant Distributed Computations. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 1–10, 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 *Proc. 20th ACM Symp. on Theory of Computing*, pages 11–19, 1988.

[Cha90] D. Chaum. Zero–knowledge undeniable signatures. In *Proc. EUROCRYPT 90*, pages 458–464. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 473.

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

[Des88] Yvo 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] Yvo G. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July 1994.

[DF90] Yvo Desmedt and Yair 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-Verlag, 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 *Proc.* 28*th IEEE Symp. on Foundations of Comp. Science*, pages 427–437, 1987.

[FGY96] Y. Frankel, P. Gemmel, and M. Yung. Witness-based cryptographic program checking and robust function sharing. To appear in proceedings of STOC96, 1996.

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

[fST91] National Institute for Standards and Technology. Digital Signature Standard (DSS). Technical Report 169, August 30 1991.

[Gen96] Rosario Gennaro. Theory and practice of verifiable secret sharing. Ph.D. thesis, Massachusetts Institute of Technology, to appear, 1996.

[GJKR96] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust and efficient sharing of rsa functions. manuscript, 1996.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald 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. *IEEE Proc.-Comput.Digit.Tech*, 141(5), Sept 1994.

[HJJ+95] Amir Herzberg, Markus Jakobson, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive proactive public key and signature systems. manuscript, 1995.

[HJKY95] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing, or: How to cope with perpetual leakage. In *Proc. CRYPTO 95*. Springer-Verlag, August 1995. Lecture Notes in Computer Science No. 963.

[HPM94] P. Horster, H. Petersen, and M. Michels. Meta-elgamal signatures schemes. In *2nd ACM Conference on Computer and Communications Security*, pages 96–107, 1994.

[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-Verlag, 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.

[NR94] K. Nyberg and R. Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. In *Proc. EUROCRYPT 94*, pages 175–190, 1994.

[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.

[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.