# Simplified VSS and Fast-track Multiparty Computations with Applications to Threshold Cryptography

(Regular presentation submission)

ROSARIO GENNARO[*] MICHAEL O. RABIN[†] TAL RABIN[‡]

**Abstract**

The goal of this paper is to introduce a simple verifiable secret sharing scheme, and to improve the efficiency of known secure multiparty protocols and, by employing these techniques, to improve the efficiency of applications which use these protocols.

First we present a very simple Verifiable Secret Sharing protocol which is based on fast cryptographic primitives and avoids altogether the need for expensive zero-knowledge proofs.

This is followed by a highly simplified protocol to compute multiplications over shared secrets. This is a major component in secure multiparty computation protocols and accounts for much of the complexity of proposed solutions. Using our protocol as a plug in unit for known protocols reduces their complexity.

We show how to achieve efficient multiparty computations in the computational model, through the application of homomorphic commitments.

Finally, we borrow from other fields and introduce into the multiparty computation scenario the notion of fast-track computations. In a model in which malicious faults are rare we show that it is possible to carry out a simpler and more efficient protocol which does not perform all the expensive checks needed to combat a malicious adversary from foiling the computation. Yet, the protocol still enables detection of faults and recovers the computation when faults occur without giving any information advantage to the adversary. This results in protocols which are much more efficient under normal operation of the system i.e. when there are no faults.

As an example of the practical impact of our work we show how our techniques can be used to greatly improve the speed and the fault-tolerance of existing threshold cryptography protocols.

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

[†]Harvard University and Hebrew University. Email: rabin@cs.huji.ac.il

[‡] IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, New York 10598, USA Email: talr@watson.ibm.com. Contact author.

# 1 Introduction

The past twenty years have witnessed an exciting development of research in the area of cryptography and network security. From the introduction of public-key cryptography [DH76, RSA78], to the invention of zero-knowledge proofs [GMR89], to the definition of the problem of secure multiparty computation and the somewhat surprising proof that any multiparty computation can be performed securely [Yao82, GMW87, BGW88, CCD88]. The combination of these results is extremely powerful, as they show that virtually any cryptographic problem can be solved under some reasonable appropriate assumptions.

Although theoretically impressive, these results lack in the area of practical feasibility. In today's applications even a simple public-key operation is sometimes considered too slow in comparison to the speed required by the application. Thus, the complicated exchanges of messages and zero-knowledge proofs in protocols like [Yao82, GMW87, BGW88, CCD88], might render them impractical. Thus, it is a high priority to optimize such techniques. Yet, they do provide for a sound basis for our solutions, in particular we will draw heavily on the solution introduced in [BGW88].

For the problem of verifiable secret sharing, attempts have been made to simplify the protocols by moving into the computational model. Such results were achieved by Feldman and Pedersen [Fel87, Ped91a], and in fact exhibit improved results with respect to communication.

We shall concentrate in this paper on the problems of verifiable secret sharing and multiparty computations. The inefficiency of the general secure multiparty protocols is partially caused by the "generality" of the algorithms. Thus, optimization can be achieved in (at least) two ways. One is to tailor protocols to the specific problem at hand. Examples of this kind of approach include works on threshold cryptography (see Section 6) where efficient multiparty computation protocols are devised for the task of shared generation of digital signatures.

Another possible approach, the one which we follow in this paper, is to go back to the original works and see if their efficiency can be directly improved. If one can devise general techniques to improve on the computation/communication of secure multiparty protocols it is also likely that these techniques would improve the efficiency of "ad-hoc" optimizations.

OUR CONTRIBUTION. In this paper we present new algorithms to perform specific computations more efficiently. Furthermore, we initiate new modes of operation to enhance overall performance. The major contributions can be summarized as follows:

- A new simple and efficient design for Verifiable Secret Sharing scheme
- Computational simplifications of the Ben-Or et al. [BGW88] protocol
- Efficient multiparty computations in the computational model
- Expediting computations through the notion of "fast-track"
- Applying all the above to a specific cryptographic problem

VSS. The first algorithm is a very simple and efficient Verifiable Secret Sharing protocol (Section 2). The main novelty of our protocol is that it is based on an efficient commitment scheme and it avoids altogether the expensive zero-knowledge proofs, which are usually carried out to ensure the correctness of actions of the participants in the protocol. Our protocol improves considerably over all existing verifiable secret sharing schemes, either in communication and/or in computation.

COMPUTATIONAL SIMPLIFICATIONS. The second protocol is a highly simplified protocol to compute multiplication over shared secrets. That is, in the model where there are two secrets $a$ and $b$ which are shared distributively among a set of $n$ players, the protocol enables the players to secretly compute the product $ab$. This protocol can be used in any existing multiparty computation protocol. For example when used inside [BGW88] it improves the speed of the computation of a multiplication gate by a factor of at least 2. When used inside our general multiparty protocol, gains are even greater.

EFFICIENT PROTOCOLS COMPUTATIONAL MODEL. We achieve efficient multiparty computations using constructions based on homomorphic commitments. Some of these techniques have been independently devised by [CDM97], yet they use them in the context of span programs.

FAST-TRACK. The following observation leads to an additional contribution. Secure multiparty protocols pay a heavy cost in terms of communication/computation in order to guarantee robustness against malicious adversaries who may cause players to behave arbitrarily during the protocol. It is a well-known phenomenon that "private" computations (i.e. secure only against passive adversaries) are usually much simpler and efficient, as they eliminate all verification of proper conduct.

Typically, however, one can expect malicious faults to happen quite rarely. Consider for example a very sensitive distributed signature generation system (like a root certification authority) where the servers are heavily protected by firewalls and other security mechanisms. In this case one cannot rule out malicious faults (and thus cannot blindly use the simpler private protocols), but on the other hand would like to take advantage in some way of the fact that faults are rare.

We would like to build on the efficiency of private protocols, which operate under the assumption that no faults occur, while avoiding the trap of assuming that you can execute the private computation until a fault occurs and then re-compute. Indeed such a computation might turn out to be insecure, and expose secret information.

Thus, we borrow from other fields and introduce into the multiparty computation scenario the notion of fast-track computations. The idea is to avoid carrying out all the verification steps, but rather to identify "critical" verification points. Only at these critical points some verification will be carried out. Once the verification is carried out in a critical point we are guaranteed that the computation up to this point is correct. These critical points must be chosen in such a manner that if faults occur between two consecutive critical points $c_1$ and $c_2$, where $c_2$ is a later point in the protocol, then the faults will be detected at point $c_2$. Furthermore, recomputing the section from critical point $c_1$ to $c_2$ will not violate the security of the computation. Thus, if no faults occurred between $c_1$ and $c_2$ we "saved" all the verifications which should have been carried out between these two points.

An attractive feature of our approach is that most of the verification at the critical points will not be the standard verification steps of the protocol, but rather a subset of the verification steps which should have been computed. For example in the general multiparty computation of an arithmetic circuit, critical points are placed on multiplications gates. At these gates we need to verify only *one* VSS compared to, for example, [BGW88] where $O(n)$ such VSS's must be checked (at least one for each player).

APPLICATIONS. As an example of the practical impact of our approach, we present its application in the area of threshold cryptography. We show that existing threshold signature protocols can be greatly enhanced in speed using our techniques. We exemplify this over the threshold DSS protocol of [GJKR96b]. The improvements are quite substantial. We improve the fault-tolerance from $n/4$ to $n/2$ without increasing the communication or the computational complexity, thanks to our simplified VSS and multiplication protocols. We also present a fast-track version of the protocol which requires >from each server a factor of $n$ modular exponentiations less than a fully fault-tolerant protocol (e.g. in [GJKR96b]) (see Section 6).

# 2 Verifiable Secret Sharing Made Very Simple

Since the appearance of Shamir's [Sha79] and Blakley's [Bla79] seminal papers on secret sharing which introduced the notion of sharing a secret and gave very simple solutions to the problem, the research on this topic has been extensive. These two solutions worked in the model where there are no faults in the system. Tompa and Woll [TW88] and McEliece and Sarwate [MS81] gave the first (partial) solutions for a model with faults. Finally the paper of Chor et al. [CGMA85] defined the complete notion of Verifiable Secret Sharing (VSS), and gave a solution. Under various assumptions, solutions to the problem were given [CGMA85, GMW91, Fel87, BGW88, CCD88, RB89, Ped91a]. In order to achieve the goal of verifiability, these protocols deviate from the original solutions' simplicity. They require either heavy computations and/or extensive zero-knowledge proofs of proper conduct. Furthermore, in order to reconstruct the secret there is again a need for extensive computations.

In this section we will describe a VSS protocol which returns to the original simplicity of Shamir's scheme, furthermore the implementation requires very little computational and communication overhead (both for sharing and reconstructing). This simple solution is enabled through an observation that all existing protocols achieve much more than is required, and by eliminating all the overhead, efficiency can be regained.

In Appendix A we present Shamir's Secret Sharing, and in Appendix B a definition of verifiable secret sharing due to [FM].

## 2.1 Our VSS protocol

We now proceed to describe a protocol which satisfies the above definition of VSS. It will be based on Shamir's secret sharing, with an additional low cost added construction. This construction will basically be an efficient commitment of the dealer to each one of the shares held by the players. The commitment to shares as a whole commits the dealer to a single secret. The individual commitments can be opened as we have enough good players who will expose their values and, through those, verify all other commitments. Our VSS protocol appears in Figure 1. In order to construct our protocol we need some form of commitment which satisfies the following conditions. We shall denote our commitment function by $\mathcal{H}$. It will be a randomized function which will receive as input the secret value $x$ and a random value $r$.

**secrecy** given $\mathcal{H}(x, r)$ it is infeasible to compute any information about $x$

**collision resistance** it is infeasible to find two strings $x_1, r_1$ and $x_2, r_2$ such that $\mathcal{H}(x_1, r_1) = \mathcal{H}(x_2, r_2)$

**universal verifiability** given $x, r$ and $y$ everybody can verify if $y = \mathcal{H}(x, r)$ (i.e. the computation of $\mathcal{H}$ does not require knowledge of a secret key).

For example one could conjecture that $\mathcal{H}(x, r) =$SHA-1$(x, r)$.

**Theorem 1** *The protocol* New-VSS *in Figure 1 is a VSS protocol.*

Proof appears in Appendix C.

EFFICIENCY AND SECURITY. If $\mathcal{H}$ is implemented via a cryptographic hash function (e.g. $\mathcal{H}(x, r) =$SHA-1$(x, r)$) then we would like to stress the efficiency of the above VSS protocol. During the sharing phase the dealer has to compute $n$ executions of the function $\mathcal{H}$ while each player computes a single evaluation, and each such computation is highly efficient. During the recover phase each player has to compute the hash $n$ times. No costly modular exponentiations or complex ZK proofs are required.

The security of $\mathcal{H}(x, r)$ =SHA-1$(x, r)$ can however be only conjectured on the basis of the collision resistance of SHA-1. However if one wants provable security without losing in efficiency one can use the efficient provably secure commitment scheme of [DPP96] based on collision resistant hashing.

## 2.2 Previous approaches

Almost all the VSS protocols in the literature (with the curious exception of the first one [CGMA85]) are based on Shamir's protocol. On top of that they add some proof from the dealer that the values shared lie on a polynomial of degree $t$, thus ensuring that the shares identify a unique secret. We refer to this property as the VSPS property, which will be defined more rigorously later.

In [GMW91] the shares are encrypted and then the VSPS property is proven via a "generic" zero-knowledge (ZK) proof of an NP-complete problem. The public knowledge of the encrypted shares also prevents bad players from contributing bad shares during reconstruction. This approach is made more efficient in [Fel87, Ped91a]

**Verifiable Secret Sharing**

**Sharing Phase**

1. Protocol for Dealer on input a secret $s$:
   - Randomly choose polynomials $f(x) = a_t x^t + ... + a_1 x + s$, and $r(x) = r_t x^t + ... + r_1 x + r_0$.
   - Compute and hand player $P_i$ the values $\alpha_i \stackrel{\text{def}}{=} f(i)$ and $\rho_i \stackrel{\text{def}}{=} r(i)$, for $1 \leq i \leq n$
   - Compute and broadcast the value $\mathcal{A}_i \stackrel{\text{def}}{=} \mathcal{H}(\alpha_i, \rho_i)$, for $1 \leq i \leq n$
2. Player $P_i$ verifies that $\mathcal{A}_i = \mathcal{H}(\alpha_i, \rho_i)$. If the equation does not hold then he broadcasts a complaint against the dealer.
3. If player $P_i$ broadcasted a complaint then the dealer broadcasts the values $\alpha_i, \rho_i$, s.t. $\mathcal{H}(\alpha_i, \rho_i) = \mathcal{A}_i$.
4. If the dealer does not follow some step he is disqualified, otherwise conclude that a secret has been shared.

**Reconstruction Phase**

1. Each player broadcasts the values $\alpha_i, \rho_i$.
2. Take $t+1$ broadcasted values for which $\mathcal{A}_i = \mathcal{H}(\alpha_i, \rho_i)$ and interpolate polynomials $\hat{f}(x)$ and $\hat{r}(x)$ of degree at most $t$ that pass through those points.
3. Compute $\hat{\alpha_i} = \hat{f}(i)$ and $\hat{\rho_i} = \hat{r}(i)$ and verify that $\mathcal{A}_i = \mathcal{H}(\hat{\alpha_i}, \hat{\rho_i})$ for all $i$. If yes, output $\hat{f}(0)$ else output 0.

Figure 1: New-VSS: - Sharing and Reconstruction Protocols

where the dealer publicly commits to the polynomial using some form of "homomorphic" commitment scheme. These commitments in return provide for a simpler proof of the VSPS property.

In [BGW88, CCD88, Rab94] the model assumes a computationally unbounded adversary, disabling the use of encryption. In this case the ZK proof is done via a cut-and-choose approach. Correction of bad shares during recover is done via error-correcting codes [BGW88, CCD88] or via a mechanism of mutual authentication [Rab94].

Is there a trend developing in all these solutions which explains why our solution is so simple? The answer is yes. The above mentioned results achieve more than just having the dealer commit to a single value. Indeed the dealer commits to a polynomial of degree $t$, where the intended secret is the free term of this polynomial. This additional commitment apparently complicates the protocol, and adds computations, and is not necessary in order to achieve the sole goal of verifiable secret sharing. Indeed our protocol shows that it is possible to commit to a single value without committing to the full polynomial. We will refer to the above protocols with the new name of *Verifiable Secret and Polynomial Sharing* (VSPS).

**Definition 1** *We say that $\pi$ is a* Verifiable Secret and Polynomial Sharing *protocol (VSPS) if the following properties hold for any adversary $\mathcal{A}$:*
*1. The protocol is a Verifiable Secret Sharing*
*2.* **VSPS property** *If the value set by the VSS is $\sigma$ then there exists a polynomial $f(x)$ of degree at most $t$, such that $f(0) = \sigma$ and player $P_i$ knows the value $f(i)$.*

In Section 4.2.1 we will provide a method to enhance our VSS scheme by adding the VSPS property.

As we will see later VSPS protocols are important as a tool for multiparty computation, due to their structural homomorphic properties. However, they are an overkill for a single VSS. And indeed there are several applications, such as storing important information for back-up in a distributed fashion on insecure devices, where there is a need only for VSS without a requirement to compute on the shares.

## 3 Simplification to Secure Multiparty Computations

We consider the problem of *secure multiparty computation* [Yao82, GMW87, BGW88, CCD88]. There are $n$ players $P_1, \ldots, P_n$. Player $P_i$ holds an input $x_i$ and the players want to compute a function $F(x_1, \ldots, x_n)$ in a *secure* manner, which intuitively means that the adversary cannot disrupt the computation, i.e. the value computed is correct, furthermore the adversary does not learn any information about the inputs of the good players (except for what is revealed by the function value).

MODEL AND DEFINITIONS. We consider a synchronous model with private channels and broadcast (e.g.[RB89, Bea89]). The parties engage in a distributed computation, following a protocol $\pi$, in order to evaluate $F(x_1, ..., x_n)$. We assume that there is an adversary $\mathcal{A}$ that corrupts up to $t$ players and coordinates their actions in an arbitrary manner. The adversary we consider is *static* i.e. it decides which players to corrupt at the beginning of the computation. Also our adversary is computationally unbounded. We follow formal definitions of secure multiparty computations that have appeared in several papers [MR91, Bea91, CFGN96, Can95].

In this section we will describe two simplifications to the [BGW88] protocol, and in particular to the multiplication protocol. We first describe an algebraic simplification followed by a simplified zero-knowledge proof for a specific property.

### 3.1 Algebraic Simplification for Multiplication Protocol

In the following we shall present a simple method for computing the multiplication of two secrets which are distributed among a set of players.

Given two secrets $\alpha$ and $\beta$ shared by polynomials $f_\alpha(x)$ and $f_\beta(x)$ respectively of degree $t$, the players would like to compute the product $\alpha\beta$. In their seminal paper Ben-Or et al. [BGW88] note that it isn't sufficient for each player to locally multiply his shares of both secrets, as this generates a polynomial whose constant term is the desired one, i.e. $\alpha\beta$, but it is of degree $2t$ and is not a random polynomial. To overcome this they introduced a degree reduction and randomization protocols. We will show how to achieve both the degree reduction and the randomization in a single step. This building block can be substituted for the multiplication step in the protocol of [BGW88], as it works in the same model of computation. The computation in this section is described under the assumption that all players act properly (as has been said, methods for how to remove this assumption appear in the next section).

Denote by $f_\alpha(i)$ and $f_\beta(i)$ the shares of player $P_i$ on $f_\alpha(x)$ and $f_\beta(x)$ respectively.

The product of $f_\alpha(x)$ and $f_\beta(x)$ is $f_\alpha(x)f_\beta(x) = a_{2t}x^{2t} + ... + a_1x + \alpha\beta \stackrel{\text{def}}{=} f_{\alpha\beta}(x)$. For $1 \leq i \leq 2t+1$, $f_{\alpha\beta}(i) = f_\alpha(i)f_\beta(i)$. Thus we can write

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 2 & \cdots & 2^{2t} \\ \vdots & & & \\ 1 & 2t+1 & \cdots & 2t+1^{2t} \end{bmatrix} \begin{bmatrix} \alpha\beta \\ a_1 \\ \vdots \\ a_{2t} \end{bmatrix} = \begin{bmatrix} f_{\alpha\beta}(1) \\ f_{\alpha\beta}(2) \\ \vdots \\ f_{\alpha\beta}(2t+1) \end{bmatrix}$$

Denote the above matrix by $A$. This is a $2t+1$ by $2t+1$ Van der Monde matrix, hence non-singular and has an inverse. Let the first row of the inverse matrix, $A^{-1}$, be $(\lambda_1, ..., \lambda_{2t+1})$, note that these are known constants. Then the previous equation implies that $\alpha\beta = \lambda_1 f_{\alpha\beta}(1) + ... + \lambda_{2t+1} f_{\alpha\beta}(2t+1)$.

Given polynomials $h_1(x), ..., h_{2t+1}(x)$ all of degree $t$ which satisfy that $h_i(0) = f_{\alpha\beta}(i)$ for $1 \leq i \leq 2t+1$, define $H(x) \stackrel{\text{def}}{=} \sum_{i=1}^{2t+1} \lambda_i h_i(x)$. Note that $H(0)$ is exactly $\lambda_1 f_{\alpha\beta}(1) + ... + \lambda_{2t+1} f_{\alpha\beta}(2t+1)$ and hence $\alpha\beta$. Furthermore, $H(j) = \sum_{i=1}^{2t+1} \lambda_i h_i(j)$.

The polynomial $H(x)$, used for the sharing of $\alpha\beta$ is automatically of degree $t$. It is random because the $\lambda_i$ are non-zero (easy to check by inspection) and there are $n-t$ polynomials $h_i(x)$ chosen by good players, and hence at random. Thus, the sharing of $\alpha\beta$ by a random polynomial of degree $t$ can be achieved directly following Protocol Simple-Mult in Figure 2.

Simple-Mult

**Input of Player $P_i$:** The values $f_\alpha(i)$ and $f_\beta(i)$

1. Player $P_i$ shares the value $f_\alpha(i)f_\beta(i)$ by choosing a random polynomial $h_i(x)$ of degree $t$, such that $h_i(0) = f_\alpha(i)f_\beta(i)$. He gives player $P_j$ the value $h_i(j)$ for $1 \le j \le 2t+1$.
2. Each player $P_j$ computes his share of $\alpha\beta$ via a random polynomial $H$, i.e. the value $H(j)$, by locally computing the linear combination $H(j) = \sum_{i=1}^{2t+1} \lambda_i h_i(j)$.

Figure 2: Simplified Multiplication Protocol with honest players

**Theorem 2** *Protocol* Simple-Mult *is a secure multiplication protocol in the presence of a passive adversary computationally unbounded.*

In order to tolerate an active adversary there is a need to verify the actions of the players. [BGW88] uses a computationally expensive protocol to do this (which could be combined with Simple-Mult). However, we were able to simplify this protocol as well, and greatly improve its efficiency. The description of our simplification appears in Appendix D.

# 4 Computations with a Polynomial Time Adversary

In this section we describe how to carry out multiparty computations in the presence of a computationally bounded adversary. It is well known that in this model there exist VSS protocols due to Feldman [Fel87] and Pedersen [Ped91a] which are quite efficient and require limited interaction. We will show that is possible to use these kind of VSS protocols, including our New-VSS, to perform multiparty computations efficiently.

The basic idea is to use a homomorphic commitment (see Section 4.1) to commit to the sharing of the inputs during the VSS. The computation will then follow the [BGW88] paradigm. Additions are computed locally by just summing up the shares of the secret values being added. For multiplication we run a robust version of the simplified multiplication protocol Simple-Mult presented above. But we will use the public commitments over the inputs to enforce correct behavior on the part of the players.

This idea originated in [CCD88] in the information-theoretic model, where such "commitments" were achieved by a second layer of input sharings. In the cryptographic model we use homomorphic commitments to generate the same effect. Some of these techniques have been independently devised by [CDM97], yet they use them in the context of span programs.

In the following sections we will concentrate on the multiplication protocol. Given two secrets $\alpha$ and $\beta$ shared via some form of VSS, which generated some representation of the secrets, we want to compute a sharing of $\gamma = \alpha\beta$ resulting in the same representation. By representation we mean either the commitment to the coefficients or the commitment to the points of the polynomial. Player $P_i$ holds shares $\alpha_i, \beta_i$ of $\alpha$ and $\beta$ (resp.). In order to get a robust version of the multiplication protocol described in Section 3 we need to enforce that $P_i$ shares the product $\alpha_i\beta_i$ via a polynomial of degree $t$.

## 4.1 Homomorphic Commitments

The approach we follow requires the usage of *homomorphic commitments*. Denote by $\mathcal{H}(\alpha, \rho)$ a commitment to $\alpha$ with randomness $\rho$. We shall say that it is a homomorphic commitment if it has the following property: given $A_1 = \mathcal{H}(\alpha_1, \rho_1)$ and $A_2 = \mathcal{H}(\alpha_2, \rho_2)$ it holds for some $\rho$ that: $A_1 \cdot A_2 = \mathcal{H}(\alpha_1 + \alpha_2, \rho)$

In our protocols we also need a ZK proof for the following: A prover $P$ publishes three commitments: $A = \mathcal{H}(\alpha, \rho)$, $B = \mathcal{H}(\beta, \sigma)$ and $C = \mathcal{H}(\alpha\beta, \tau)$ and wants to prove in ZK to a verifier $V$ that $C$ is a commitment to the product of the committed values in $A$ and $B$ (see Appendix F).

POLYNOMIAL EVALUATIONS. Assuming a polynomial $f(x) = a_t x^t + ... + a_1 x + a_0$, the following two operations can be carried out:

- if the coefficients of the polynomial are committed to using the above scheme, then directly from these commitments we can compute commitments to the value $f(i)$, for $1 \leq i \leq n$, in the following we will call this procedure "evaluation in the exponent".
- and reversely, given commitments to $f(i)$, for $1 \leq i \leq n$, it is possible to compute commitments to the coefficients of the polynomial, in the following we will call this procedure "interpolation in the exponent".

Both of these computations are possible as there is a linear relation between the coefficients and the evaluated points thus, due to the homomorphic properties of the commitment, the computation can be carried out in the exponent.

Homomorphic commitments based on general computational assumptions have been recently introduced and studied by Cramer and Damgard [CD97]. The ZK proof in Appendix F is also due to them. For simplicity of exposition we will use a specific commitment scheme due to Pedersen described below. However the reader should keep in mind that any of the commitments in [CD97] will do.

Let $p$ and $q$ be primes such that $p = \mu q + 1$, where $g$ is an element of order $q$ in $Z_p^*$ and $h \stackrel{\text{def}}{=} g^z \bmod p$. The value $z$ is unknown to the dealer and players.

**Discrete Log Assumption:** We assume that it is infeasible to compute discrete logarithms in the subgroup of $Z_p^*$ generated by $g$.

A commitment to a string $\alpha \in Z_q$ using a random $\rho \in_R Z_q$ is the value $A = g^\alpha h^\rho \bmod p$. It is proven in [Ped91a] that this commitment is information-theoretic secure in terms of privacy and can be opened in two different ways only by somebody who can compute $z$.

## 4.2 Multiparty Computation Using our VSS

When we introduced our VSS protocol we said that it gained in efficency because it did not satisfy the VSPS property, i.e. the guarantee that there exists an underlying polynomial. We further said that this property is needed for the multiparty computations of [BGW88]. Thus, if we want to use our protocol for computations we will first need to reintroduce the VSPS property into our VSS. Yet, we add the VSPS in such a manner that our VSS with VSPS enjoys a novel property which is that the verification of the existence of a secret is disjoint from the verification of the VSPS property. This split will enable us to expedite our computations along the fast-track paradigm (see Section 5). We start by showing how to verify the VSPS property followed by the presentation of the robust multiplication gate.

### 4.2.1 Checking the VSPS Property

The original description of our VSS protocol simply assumed a commitment scheme, but for the multiparty computations we will implement this commitment with the homomorphic commitment of Pedersen. Now, the dealer will share his secret $\alpha \in Z_q$ in the following manner. He will choose polynomials $f(x) = a_t x^t + ... + a_1 x + \alpha$, and $r(x) = r_t x^t + ... + r_0$. The dealer will compute and give player $P_i$ the values $\alpha_i \stackrel{\text{def}}{=} f(i)$ and $\rho_i \stackrel{\text{def}}{=} r(i)$. The commitment will be done by $\mathcal{A}_i = \mathcal{H}(\alpha_i, \rho_i) \stackrel{\text{def}}{=} g^{\alpha_i} h^{\rho_i} \bmod p$. For reasons that will become apparent later we extend the VSS protocol by having the dealer commit also to the secret itself, which is $f(x)$ evaluated at 0, by publishing $\mathcal{A}_0 = g^\alpha h^{r_0}$. The reconstruction phase is, in essence, as before; player $P_i$ broadcasts $\alpha_i$ and $\rho_i$. We accept only those values that match the published commitment $\mathcal{A}_i$. The polynomials $\hat{f}$ and $\hat{r}$ are interpolated from the accepted values and a check is carried out that, for all $i = 0, \ldots, n$, $\mathcal{A}_i = \mathcal{H}(\hat{f}(i), \hat{r}(i))$. If this check succeeds then $\alpha \stackrel{\text{def}}{=} hat f(0)$ otherwise $\alpha \stackrel{\text{def}}{=} 0$.

We denote with DL-VSS the above implementation of New-VSS. Although it looks similar to Pedersen's VSS it differs from it because in DL-VSS the public commitments are to the points of the polynomial, while in Pedersen's VSS the commitments are to the coefficient. For this same reason however DL-VSS does not have the VSPS property i.e. it does not insure that the shares lie on a polynomial of degree $t$.

The first method that comes to mind to verify the VSPS property, is to interpolate in the exponent the polynomial from $t+1$ values, and then to evaluate in the exponent the remaining points, and see if they match. Yet, this solution is highly expensive in computation. We present a more efficient randomized solution.

If the $\mathcal{A}_0, \ldots, \mathcal{A}_n$ determine a unique pair of $t$-degree polynomials $(f, r)$ such that $\mathcal{A}_i = g^{f(i)} h^{r(i)}$, then $\mathcal{A}_0, \ldots, \mathcal{A}_t$ should define $(f, r)$ and so should $\mathcal{A}_{t+1}, \ldots, \mathcal{A}_{2t+1}$. Denote by $f^{(1)}(x) = a_{1,t}x^t + \ldots + a_{1,0}$, $r^{(1)}(x) = r_{1,t}x^t + \ldots + r_{1,0}$ and $f^{(2)}(x) = a_{2,t}x^t + \ldots + a_{2,0}$, $r^{(2)}(x) = r_{2,t}x^t + \ldots + r_{2,0}$ the polynomials defined by the first and second sets respectively. The idea of the check is to prove that for a random value $\delta \in Z_q$ we have

$$g^{f^{(1)}(\delta)} h^{r^{(1)}(\delta)} = g^{f^{(2)}(\delta)} h^{r^{(2)}(\delta)} \tag{1}$$

as $h = g^z$ this implies that $f^{(1)}(\delta) + zr^{(1)}(\delta) = f^{(2)}(\delta) + zr^{(2)}(\delta)$. But since $\delta$ is chosen at random that means that with probability $1 - \frac{t}{q}$ we have

$$f^{(1)}(x) + zr^{(1)}(x) = f^{(2)}(x) + zr^{(2)}(x) \tag{2}$$

For large $q$ the probability of error can be made negligible.

Recall that our final goal is to prove that $f^{(1)}(x) = f^{(2)}(x)$ and $r^{(1)}(x) = r^{(2)}(x)$. Suppose that the dealer distributed shares such that $f^{(1)}(x) \neq f^{(2)}(x)$ and $r^{(1)}(x) \neq r^{(2)}(x)$, but such that Equation (2) holds. Then it is easy to see that the dealer can compute $z$ which contradicts the assumptions.

Thus, the whole test reduces to a local check by each player of Equation (1) for a random $\delta \in Z_q$ chosen by the player. The left side of the equation can be computed as follows:

$$g^{f^{(1)}(\delta)} h^{r^{(1)}(\delta)} = g^{\sum_{j=0}^{t} a_{1,j}\delta^j} h^{\sum_{j=0}^{t} r_{1,j}\delta^j} =$$
$$g^{\sum_{j=0}^{t}\sum_{i=0}^{t} f(i)\lambda_{ji}\delta^j} h^{\sum_{j=0}^{t}\sum_{i=0}^{t} r(i)\lambda_{ji}\delta^j} = \prod_{i=1}^{t+1} (g^{f(i)} h^{r(i)})^{\Delta_i} = \prod_{i=1}^{t+1} \mathcal{A}_i^{\Delta_i}$$

where $\Delta_i = \sum_{j=0}^{t} \lambda_{ji}\delta^j$ for appropriate Lagrange coefficients $\lambda_{ji}$. Similarly compute the right-hand side of Equation (1). We denote with VSPS-Check the above method for verifying the VSPS property.

### 4.2.2 The Robust Multiplication Gate with our VSS

Let us assume that we are given two secrets $\alpha$ and $\beta$ shared via our DL-VSS protocol with polynomials $f_\alpha(x), f_\beta(x)$ (resp). Player $P_i$ has shares $\alpha_i \stackrel{\text{def}}{=} f_\alpha(i)$ and $\beta_i \stackrel{\text{def}}{=} f_\beta(i)$ in addition to $\rho_i \stackrel{\text{def}}{=} r(i)$ and $\sigma_i \stackrel{\text{def}}{=} s(i)$ where $r(x), s(x)$ are two random polynomials of degree $t$. The values $\mathcal{A}_i = \mathcal{H}(\alpha_i, \rho_i) = g^{\alpha_i} h^{\rho_i}$ and $\mathcal{B}_i = \mathcal{H}(\beta_i, \sigma_i) = g^{\beta_i} h^{\sigma_i}$ are public. We assume that the VSPS property of these two sharings has been checked.

The basic idea of the robust multiplication protocol is the following: each player $P_i$ shares $c_i = \lambda_i \alpha_i \beta_i$ via our DL-VSS protocol, where $\lambda_i$ is the coefficient defined in Section 3.1. If $c_{ij}$ and $\tau_{ij}$ are the values $P_i$ sends to $P_j$, then $P_i$ publishes $\mathcal{C}_{ij} = \mathcal{H}(c_{ij}, \tau_{ij}) = g^{c_{ij}} h^{\tau_{ij}}$.

After the sharing the players check the VSPS property for $P_i$'s sharing. Notice that $P_i$ broadcasted the value $C_{i0} = g^{\lambda_i \alpha_i \beta_i} h^{\tau_{i0}}$. $P_i$ uses this value to prove in zero-knowledge that he shared $\lambda_i \alpha_i \beta_i$ with respect to $\mathcal{A}_i$ and $\mathcal{B}_i$ using the protocol in Appendix F. For any player who does not follow the protocol, all his private information is made public through reconstruction. It is important to note that our representation of the secret as a commitment to the points on the polynomial lends naturally to the ZK proof, as the values are already in the format needed for the proof.

Now we are at the starting point of the multiplication operation described in Section 3.1 with the additional property that we know that all the sharings are correct. Thus, each player locally sums the shares which he has received from all the other players in order to compute $\gamma_i = \sum_{j=1}^{2t+1} c_{ji}$ and $\tau_i = \sum_{j=1}^{2t+1} \tau_{ji}$. Furthermore, the public information corresponding to this new share is generated: $\mathcal{C}_i = \mathcal{H}(\gamma_i, \tau_i) = g^{\gamma_i} h^{\tau_i} = \prod_{j=1}^{2t+1} \mathcal{C}_{ji}$. The full protocol appears in Figure 3 and is denoted Mult.

**Mult: Robust Multiplication**

**Input of player $P_i$:** values $\alpha_i = f_\alpha(i)$, $\beta_i = f_\beta(i)$, $\rho_i = r(i)$, $\sigma_i = s(i)$.
**Public input:** $\mathcal{A}_i = \mathcal{H}(\alpha_i, \rho_i) = g^{\alpha_i} h^{\rho_i}$, $\mathcal{B}_i = \mathcal{H}(\beta_i, \sigma_i) = g^{\beta_i} h^{\sigma_i}$ for $0 \leq i \leq n$

1. Each player $P_i$ shares $\lambda_i \alpha_i \beta_i$ using the DL-VSS protocol. That is set $c_{ij} = f_{\alpha\beta,i}(j)$, $\tau_{ij} = u_i(j)$ where $f_{\alpha\beta,i}$, $u_i$ are random polynomials of degree $t$ such that $f_{\alpha\beta,i}(0) = \lambda_i \alpha_i \beta_i$.
   **Secret information of $P_i$:** share $c_{ji}, \tau_{ji}$ of $\lambda_j \alpha_j \beta_j$
   **Public information:** $\mathcal{C}_{ij} = g^{c_{ij}} h^{\tau_{ij}}$ for $1 \leq i, j \leq n$ $\quad \mathcal{C}_{i0} = g^{c_{i0}} h^{\tau_{i0}}$ for $1 \leq i \leq n$
2. Players run a VSPS-Check on $P_i$'s sharing. If a sharing fails the test then expose the secret through the VSS reconstruction.
3. $P_i$ proves in zero-knowledge that $C_{i0}$ is a commitment to the product of $\lambda_i \alpha_i \beta_i$ using the ZK proof from Appendix F. Expose the values of the players who fail the proof.
4. Player $P_i$ computes $\gamma_i = \sum_{j=1}^{2t+1} c_{ji}$ which is a share of $\gamma = \alpha\beta$ via a random polynomial of degree $t$. Compute also $\tau_i = \sum_{j=1}^{2t+1} \tau_{ji}$ and $\mathcal{C}_j = \mathcal{H}(\gamma_j, \tau_j) = g^{\gamma_j} h^{\tau_j} = \prod_{l=1}^{2t+1} \mathcal{C}_{lj}$, for $1 \leq j \leq n$.
   **Secret information of $P_i$:** share $\gamma_i$
   **Public information:** $\mathcal{C}_i$ for $1 \leq i \leq n$

Figure 3: Robust multiplication protocol using DL-VSS

**Theorem 3** *Under the the discrete log assumption protocol* Mult *is a secure multiplication protocol in the presence of a computationally bounded active adversary.*
Plugging the above multiplication protocol into the [BGW88] construction one gets that for any function $F$ there exists a secure multiparty computation protocol. We note that this protocol is quite efficient in terms of computation and communication required by each player.

### 4.3 Efficiency Analysis

A protocol similar to Mult using Pedersen's VSS instead of our DL-VSS is presented in Appendix E and denoted Ped-mult. We omit from this extended abstract the complete computational analysis of Mult, Ped-Mult and the comparison between them. Here we only point out the major issues in this comparison.

- Our new VSS DL-VSS generates commitments to the points of the polynomial, and these are the values which are required as input for the ZK proof of proper conduct. Pedersen's VSS instead has commitments to the coefficients of the polynomial and thus is required in the multiplication protocol to compute these values via evaluation in the exponent.
- Pedersen's VSS takes advantage of the fact that the check of the VSPS property requires exponentiations to relatively small exponents. Our VSPS-Check instead requires full exponentiations in the group generated by $g$. However a close look at the cost analysis shows that only for very small $n$ there is an advantage of using Pedersen's VSS versus DL-VSS plus VSPS-Check. Relatively fast (in the growth of $n$) they have the same performance.
- However, the most attractive feature of using DL-VSS is that the verification of the existence of a secret and the verification of the VSPS property are separate computations. This will allow for the introduction of the fast-track paradigm described in Section 5 which will improve the overall performance of the protocol when there are no faults in the system.

# 5 Fast-track Computation

As we mentioned in the Introduction secure multiparty protocols pay a heavy cost in terms of communication/computation in order to guarantee robustness against malicious adversaries. Typically, however, one can expect malicious faults to happen quite rarely. We would like to build on the efficiency of private protocols, which operate under the assumption that no faults occur, while avoiding the trap of assuming that you can execute the private computation until a fault occurs and then re-compute. Indeed such a computation might turn out to be insecure, and expose secret information.

Thus, we borrow from other fields and introduce into the multiparty computation scenario the paradigm of fast-track computation. The idea is to avoid carrying out all the verification steps, but rather to identify "critical" verification points. Only at these critical points some verification will be carried out. Once the verification is carried out in a critical point we are guaranteed that the computation up to this point was correct. These critical points must be chosen in such a manner that if faults occur between two consecutive critical points $c_1$ and $c_2$, where $c_2$ is a later point in the protocol, then the faults will be detected at point $c_2$. Furthermore, recomputing the section from critical point $c_1$ to $c_2$ will not violate the security of the computation. Thus, if no faults occurred between $c_1$ and $c_2$ we "saved" all the verifications which should have been carried out between these two points.

The main result of this section is the following.

**Theorem 4** *For any function $F$ There exists a fast-track secure multiparty multiplication protocol* FT-Mult *that requires a factor of $n$ less computation than* Mult *when there are no faults in the system.*

It will become clear here why our DL-VSS protocol with VSPS-Check , which has a disjoint verification for the existence of a secret and for the VSPS property, falls nicely into the framework of fast-track. It allows to verify the existence of a valid secret at a low cost, and delay the expensive VSPS check to a later point, in which the property can be effectively verified for many secrets by a single check.

Furthermore in Appendix H we present fast-track Joint VSS protocols, which allow a set of players to generate a random secret unknown to all of them in a shared form via a VSS protocol.

## 5.1 Fast-track Robust Multiplication Protocol

In this section we describe FT-Mult . When computing a multiplication gate we do not check the VSPS property on every sharing of the values $\lambda_i\alpha_i\beta_i$ but rather we check *only the combined secret* which should be the result of the multiplication. Basically we run a single VSPS-Check protocol on the values $\mathcal{C}_1, \ldots, \mathcal{C}_n$. Thus, we reduce the number of VSPS checks by a factor of $n$ (assuming there are no faults). If the check fails then we know that there were faults and reiterate the computation of the gate using the Mult protocol.

The protocol works in the following manner: each player $P_i$ shares the product of his local shares, i.e. $\lambda_i\alpha_i\beta_i$ via our DL-VSS protocol. Using the commitment to the free term he proves (using the ZK proof in Appendix F) that he has in fact shared the proper value. Then the player computes the sum of the shares which he has received, and on the set of result of this computation the players check the VSPS property. The complete protocol appears in Appendix G.

# 6 Threshold Cryptography Applications

In recent years it has become evident that one of the most important applications of secure multiparty computation is *threshold cryptography* [Des87, Des94]. Consider for example the cryptographic function of signing which receives as input a secret key and a message, and generates the signature on the message. The signer holding the secret key can easily generate the signature. But if his computer is broken into, then the secrecy of his key is compromised. In other words, the storage of the secret key creates a single point of failure which we would like to eliminate. This can be achieved by sharing the secret key among several signing servers in a threshold fashion. Now the computation of the signature must be carried out in a distributed manner via a multiparty computation protocol among the signing servers.

Threshold cryptography is indeed the study of efficient multiparty computation protocols for cryptographic functions (e.g. signing or decrypting) in which each party has as input a share of the secret key that allows the computation of such function. Examples of threshold cryptography protocols can be found in [Boy89, Des87, DF91, DF89, CMI93, Har94, DDFY94, PK96, Lan95, GJKR96b, FGY96, GJKR96a, JY].

The above cited protocols use, in various ways, expensive VSS protocols and zero-knowledge proofs. Though some are more efficient than others there is still room and need for improvement. Our techniques can be readily applied to this scenario to obtain much more efficient protocols.

We would like to present a specific application of this paradigm. In the next section we will apply our techniques to the robust threshold DSS protocol of Gennaro e Tal [GJKR96b]. The improvements to that protocol will be twofold:

**fault-tolerance** the simplified multiplication protocol described in this paper brings the fault-tolerance of the scheme up to $\frac{n-1}{2}$ (from $\frac{n-1}{4}$) without an increase in communication or computational complexity.

**efficiency** Our new DSS protocol has a fast-track version which requires a factor of $n$ less computation (in terms of modular exponentiations) from each player.

SECURITY. Formal definitions of security for threshold signature protocols can be found in [GJKR96b]. We stress that our new protocol can be proven secure under the sole assumption of the unforgeability of DSS signatures. For the details see Appendix I.

## Acknowledgments

We would like to thank Hugo Krawczyk for countless suggestions on earlier versions of this paper and Ivan Damgard for suggesting the use of the [DPP96] provably secure commitments in our protocol New-VSS. We also thank: Ran Canetti, Ronald Cramer, Juan Garay, Amir Herzberg and Ueli Maurer for useful discussions.

## References

[BB89] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds. In *Proc. 8th ACM Symp. on Principles of Distributed Computati on*, pages 201–209. ACM, 1989.

[Bea89] D. Beaver. Multiparty Protocols Tolerating Half Faulty Processors. In G. Brassard, editor, *Advances in Cryptology — Crypto '89*, pages 560–572, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science No. 435.

[Bea91] D. Beaver. Foundations of secure interactive computing. In J. Feigenbaum, editor, *Advances in Cryptology — Crypto '91*, pages 377–391, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science No. 576.

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

[Bla79] G. R. Blakley. Safeguarding cryptographic keys. In *Proc. AFIPS 1979 National Computer Conference*, pages 313–317. AFIPS, 1979.

[Boy89] C. Boyd. Digital Multisignatures. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 241–246. Claredon Press, 1989.

[Can95] Ran Canetti. *Studies in Secure Multiparty Computation*. PhD thesis, weizmann Institute of Science, 1995.

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

[CD97] R. Cramer and I. Damgard. Zero-knowledge for finite field arithmetic or: Can zero-knowledge be for free? Manuscript, 1997.

[CDM97] R. Cramer, I. Damgard, and U. Maurer. Span programs and general multiparty computations. Manuscript, 1997.

[CFGN96] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proc.* 28*th Annual Symp. on the Theory of Computing*, pages 639–648. ACM, 1996.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *Proceeding 26th Annual Symposium on the Foundations of Computer Science*, pages 383–395. IEEE, 1985.

[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, 1993.

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

[Des87] Yvo Desmedt. Society and group oriented cryptography: A new concept. In C. Pomerance, editor, *Advances in Cryptology — Crypto '87*, pages 120–127, Berlin, 1987. Springer-Verlag. Lecture Notes in Computer Science No. 293.

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

[DF89] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In G. Brassard, editor, *Advances in Cryptology — Crypto '89*, pages 307–315, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science No. 435.

[DF91] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In J. Feigenbaum, editor, *Advances in Cryptology — Crypto '91*, pages 457–469, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science No. 576.

[DH76] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DPP96] I. Damgard, T.P. Pedersen and B. Pfitzmann. Statistical Secrecy and Multi-Bit Commitments. BRICS report series, RS-96-45, available from http://www.brics.dk

[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 Annual Symp. on Foundations of Computer Science*, pages 427–437. IEEE, 1987.

[FGY96] Y. Frankel, P. Gemmell, and M. Yung. Witness-based Cryptographic Program Checking and Robust Function Sharing. In *Proc.* 28*th Annual Symp. on the Theory of Computing*, pages 499–508. ACM, 1996.

[FM] P. Feldman and S. Micali. A Definition of Verifiable Secret Sharing. An adaptation from [FM88].

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

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

[GJKR96a] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of RSA functions. In N. Koblitz, editor, *Advances in Cryptology — Crypto '96*, pages 157–172, Berlin, 1996. Springer-Verlag. Lecture Notes in Computer Science No. 1109.

[GJKR96b] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In Ueli Maurer, editor, *Advances in Cryptology — Eurocrypt '96*, pages 354–371, Berlin, 1996. Springer-Verlag. Lecture Notes in Computer Science No. 1070.

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

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game. In *Proc.* 19*th Annual Symp. on the Theory of Computing*, pages 218–229. ACM, 1987.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems . *Journal of the ACM*, 38(1):691–729, 1991.

[Har94] L. Harn. Group oriented (t,n) digital signature scheme. *IEEE Proc.-Comput.Digit.Tech*, 141(5):307–313, Sept 1994.

[JY] Markus Jakobsson and Moti Yung. Distributed "magic ink" signatures. To appear in EuroCrypt97.

[Lan95] S. Langford. Threshold dss signatures without a trusted party. In D. Coppersmith, editor, *Advances in Cryptology — Crypto '95*, pages 397–409, Berlin, 1995. Springer-Verlag. Lecture Notes in Computer Science No. 963.

[MR91] S. Micali and P. Rogaway. Secure computation. In J. Feigenbaum, editor, *Advances in Cryptology — Crypto '91*, pages 392–404, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science No. 576.

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

[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. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Advances in Cryptology — Crypto '91*, pages 129–140, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science No. 576.

[Ped91b] T. Pedersen. A threshold cryptosystem without a trusted party. In D. Davies, editor, *Advances in Cryptology — Eurocrypt '91*, pages 522–526, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science No. 547.

[Rab94] T. Rabin. Robust Sharing of Secrets When the Dealer is Honest or Faulty. *Journal of the ACM*, 41(6):1089–1109, 1994.

[RB89] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In *Proc. 21st Annual Symp. on the Theory of Computing*, pages 73–85. ACM, 1989.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communication of the ACM, 21(2):120–126, 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.

[TW88] M. Tompa and H. Woll. How to share a secret with cheaters. *Journal of Cryptology*, 1(2):133–138, 1988.

[Yao82] A.C. Yao. Protocols for secure computations. In Proceedings of FOCS'82, pages 160–164, Chicago, 1982. IEEE.

# A Shamir's Secret Sharing

Assume the dealer has a secret $s$ which is a number in $Z_p$ where $p$ is a prime. The dealer wants to "share" this number among $n$ players $P_1, \ldots, P_n$ so that $t$ of them have *no* information about the secret while $t+1$ of them can reconstruct it. Shamir's protocol [Sha79] is described in Figure 4.

It is important to notice that the protocol works only under the assumption that no faults occur in the system. Otherwise, for example, there is no assurance that the dealer shared values which define a polynomial of degree at most $t$. And during reconstruction time the bad players may compromise the recovering of $s$ by contributing values $\hat{\alpha}_i$ different than the ones originally received from the dealer.

# B Verifiable Secret Sharing

Informally a VSS protocol achieves secret sharing in the presence of malicious faults. In other words what we want is that at the end of the sharing phase the good players are guaranteed that indeed a secret has been shared, in the sense that they will be able to reconstruct it at the end of the recover phase, regardless of the actions of a faulty dealer or players.

**Shamir's Secret Sharing**

**Sharing Phase** Protocol for Dealer on input a secret $s$:

- Choose $a_t, \ldots, a_1 \in_R Z_p$ and define the polynomial $f(x) = a_t x^t + \ldots + a_1 x + s$
- Compute and hand to player $P_i$ the value $\alpha_i \stackrel{\text{def}}{=} f(i) \bmod p$, for $1 \leq i \leq n$.

**Reconstruction Phase**

1. Each player broadcasts the value $\alpha_i$.
2. Take $t+1$ broadcasted values and interpolate a polynomial $f(x)$ of degree at most $t$.
3. Output $s = f(0) \bmod p$.

Figure 4: Sharing and Reconstruction Protocols

Another way of thinking of VSS is as a "recoverable commitment". In typical commitment schemes when Alice commits to a secret value $s$ to Bob, Bob has a guarantee that indeed there is a unique committed secret although he knows nothing about $s$. This is due to the secrecy and binding properties of commitment schemes. However nothing prevents Alice from never opening the commitment at a later time. VSS protocols have the same functionality of commitments with the added feature that at a later time it is always possible for the good players to reconstruct the value the dealer committed to.

The following definition of VSS is from [FM88, FM].

We have $n$ players $P_1, \ldots, P_n$ and a distinguished player $D$, the dealer. The dealer and the players are connected by private communication channels and they also have access to a broadcast channel. There is a static adversary $\mathcal{A}$ that can corrupt up to $t$ of the players including the dealer.

Let $\pi$ be a protocol consisting of two phases `Share,Reconstruct` in which all players have as common input the description of a set of possible secrets, $S$. The dealer has an extra input the secret $s$ in $S$. At the end of `Share` each player $P_i$ is instructed to output a Boolean value $ver_i$. At the end of `Reconstruct` each player is instructed to output a value in $S$.

We say that $\pi$ is a Verifiable Secret Sharing protocol (VSS) if the following properties hold for any adversary $\mathcal{A}$

**Unanimity** If any good player $P_i$ output $ver_i = 1$ at the end of `Share`, then $ver_j = 1$ for all other good players $P_j$

**Acceptance of good secrets** If the dealer is good, then $ver_i = 1$ for every good $P_i$

**Verifiability** If a good player $P_i$ outputs $ver_i = 1$ at the end of `Share` then there exists a value $\sigma$ in the set of possible secrets, $S$, such that the event that all good players output $\sigma$ at the end of `Reconstruct` is fixed at the end of `Share`. Moreover if the dealer is good then $\sigma = s$ the original secret input of the dealer.

**Unpredictability** If the secret $s$ is randomly chosen from a set of cardinality $q$, and the dealer is good, then the adversary $\mathcal{A}$ cannot guess at the end of `Share` the value $s$ with probability better than $\frac{1}{q}$ by a non-negligible additive factor.

The final condition can be strengthened by requiring that the view of the adversary is simulatable by a simulator that has no knowledge of $s$. Which means that the adversary gains no knowledge at all from the execution of the VSS protocol.

## C Proof of Theorem 1

### Sketch of Proof

UNANIMITY. The decision to disqualify or accept the sharing is done based on public information viewed by all players, hence all good players reach the same decision.

ACCEPTANCE OF GOOD SECRETS. If the dealer is good then all his public actions will be seen as proper and all honest players will decide that a secret has been shared.

VERIFIABILITY. This property is achieved via the collision resistance of $\mathcal{H}$. Assume w.l.o.g. that at least $P_1, ... P_{t+1}$ are honest. Let $f(x)$, $r(x)$ be the polynomials of degree $t$ determined by values $\alpha_i$ and $\rho_i$, for $1 \leq i \leq t+1$. If $\mathcal{A}_i = \mathcal{H}(f(i), r(i))\ \forall i$ then define $\sigma \stackrel{\text{def}}{=} f(0)$. Otherwise, $\sigma \stackrel{\text{def}}{=} 0$. The dealer committed himself to the values $\mathcal{A}_1, \ldots, \mathcal{A}_n$ by broadcasting them. The values $\alpha_i, \rho_i$ for $1 \leq i \leq t+1$ are set at the end of the sharing phase, and hence $f(x)$ is set. Thus, $\sigma$ is well defined at the end of the sharing phase. It remains to be shown that at the end of the reconstruction phase the players output the value $\sigma$. Assume by contradiction that they reconstruct $\hat{\sigma} \neq \sigma$ by choosing $t+1$ values $\alpha_{i_1}, \ldots, \alpha_{i_{t+1}}$ given out by players such that $\mathcal{H}(\alpha_{i_j}, \rho_{i_j}) = \mathcal{A}_{i_j}$. This means that the $t$-degree polynomials $\hat{f}(x), \hat{r}(x)$ interpolated by the $\alpha_{i_j}$ and $\rho_{i_j}$ (resp.) have the property that $\mathcal{H}(\hat{f}(i), \hat{r}(i)) = \mathcal{A}_i$ but $\hat{f}(x) \neq f(x)$ (as they differ in the free term), thus there must be an index $j$ such that $\hat{f}(j) \neq f(j)$. The pairs $(\hat{f}(j), \hat{r}(j))$ and $(f(j), r(j))$ are a collision for $\mathcal{H}$, which is known to either the dealer or player $P_j$, which contradicts the hypothesis.

UNPREDICTABILITY. If the dealer is good the adversary sees $t$ points on a polynomial of degree $t$ plus all the values $\mathcal{A}_i$. But as we assume that $\mathcal{H}$ has the secrecy property the $\mathcal{A}_i$'s give no information about the other points. Hence, $\mathcal{A}$ has no information about the secret. In other words it is possible to simulate the view of the adversary with $t$ random values as the shares and $n$ random values as the $\mathcal{A}_i$'s.

## D Computing Multiplication with Faults

The underlying assumption for the computation in the previous section is that each player $P_i$ shared a polynomial $h_i(x)$ such that $h_i(0) = f_\alpha(i) f_\beta(i)$. We present a simple method for verifying that $P_i$ has shared the proper value. To reduce the complexity of exposition we change the notation, saying that player $P_i$ has values $\alpha$ and $\beta$ and he needs to share a polynomial whose constant term is $\alpha\beta$. We take as a starting point that the values $\alpha, \beta$ have been shared properly using polynomials $f_\alpha(x), f_\beta(x)$ resp. (see [BGW88] for proof). Thus, we need to prove that the three polynomials satisfy the property that $h(0) = f_\alpha(0) f_\beta(0)$.

We are able to present a simpler proof for this property based on a combination of two ideas. The first idea is, as in the multiplication step, that instead of reducing the degree of a polynomial and randomizing it through computation it can be directly shared as a random polynomial of degree $t$. And the second is that the prover is present and can help the players out during the proof stage. More specifically, previous proofs assumed that the players need to reconstruct the polynomials while correcting errors. Under this assumption a set of $3t+1$ players can *interpolate* a polynomial of degree at most $t$. But if the dealer exposes the polynomial directly and the players only need to verify their points, then a set of $3t+1$ players can *check* their values and insure the validity of a polynomial of degree (at most) $2t$.

Thus, we shall have player $P_i$ share $h(x)$ of degree $t$ and prove that $h(0) = f_\alpha(0)f_\beta(0)$ in the following manner. First, $P_i$ will prove that $h(x)$ is of degree $t$. Then, $P_i$ will share an additional polynomial $r(x)$ of degree $2t-1$, there is no need to verify that it is of the right degree, because one of two things can happen: information of $P_i$ will be revealed is $P_i$'s, or the proof will not go through. To complete the proof $P_i$ will broadcast the polynomial $R(x) = xr(x) + f_\alpha(x)f_\beta(x) - h(x)$. This is a random polynomial of degree $2t$ and hence reveals no information about the coefficients of $f_\alpha(x)f_\beta(x)$ or $h(x)$. Each player $P_j$ checks that $R_i(0) = 0$ which indicates that $h(x)$ as as its constant term the product $\alpha\beta$. Furthermore, $P_j$ verifies that $R(j) = jr(j) + f_\alpha(j)f_\beta(j) - h(j)$, to ensure that his share of $h(x)$ is in fact on the polynomial, if there is no match he requests that his values be made public.

This is much more efficient than the proof in [BGW88] that uses error-correction in quite a complicated way to enforce the condition that $h(0) = f_\alpha(0)f_\beta(0)$. ■

# E The Multiplication Gate with Pedersen's VSS

In this section we show how to carry out the multiplication gate using Pedersen's VSS [Ped91a]. A dealer for a secret $\alpha \in Z_q$ chooses a random polynomial $f_\alpha(x) = a_t x^t + \ldots + a_0$ (with $a_0 = \alpha$) and a random polynomial $r(x) = r_t x^t + \ldots + r_0$ where $a_i, r_i \in Z_q$. The dealer gives to player $P_i$ the values $\alpha_i = f_\alpha(i) \bmod q$ and $\rho_i = r(i) \bmod q$. He then publishes the following values $A_0, \ldots, A_t$ where $A_j = g^{a_j} h^{r_j} \bmod p$. The $A_i$'s are basically commitments to the coefficients of the polynomials. Each player checks that his share lies on the committed polynomial by checking that

$$g^{\alpha_i} h^{\rho_i} = \prod_{j=0}^{t} A_j^{i^j}$$

Let us now deal with a multiplication gate. Assume that the two secrets $\alpha$ and $\beta$ are currently shared using Pedersen's VSS.
That is $\alpha$ is shared via polynomials $f_\alpha(x) = a_t x^t + \ldots + a_0$ (with $a_0 = \alpha$) and $r(x) = r_t x^t + \ldots + r_0$; each player $P_i$ holds the values $\alpha_i = f_\alpha(i) \bmod q$ and $\rho_i = r(i) \bmod q$. The values $A_j = g^{a_j} h^{r_j} \bmod p$ (for $j = 0, \ldots, t$) are public.
Similarly $\beta$ is shared via polynomials $f_\beta(x) = b_t x^t + \ldots + b_0$ (with $b_0 = \beta$) and $s(x) = s_t x^t + \ldots + s_0$; each player $P_i$ holds the values $\beta_i = f_\beta(i) \bmod q$ and $\sigma_i = s(i) \bmod q$. The values $B_j = g^{b_j} h^{s_j} \bmod p$ (for $j = 0, \ldots, t$) are public.

We use the simplified multiplication protocol shown in Section 3.1. Each player $P_i$ shares the value $\lambda_i \alpha_i \beta_i$ via Pedersen's VSS. This will assure that the value is shared via a polynomial of degree $t$. A side effect of the VSS sharing is that $P_i$ publishes the value $g^{\lambda_i \alpha_i \beta_i} h^\tau$ for some random value $\tau$. We will use this public value to check that $P_i$ shared the correct value $\alpha_i \beta_i$. This is done by first generating from the commitment to the coefficients of the polynomial of $\alpha$ ($\beta$) a commitment to the interpolated values, i.e. $g^{\alpha_i} h^{\rho_i}$ ($g^{\beta_i} h^{\sigma_i}$), via interpolation in the exponents. Then player $P_i$ proves in ZK that the value he shared is the product of the values contained in these two commitments. A protocol for this task is described in Appendix F.

The full protocol is described in Figure 5.

# F ZK Proof for multiplication of committed values

In both the Mult and FT-Mult protocols a crucial tool to prove that a player is performing correctly is a ZK proof of the following statement.

**Ped-Mult: Multiplication based on Pedersen's VSS**

Input of player $P_i$: values $\alpha_i = f_\alpha(i)$, $\beta_i = f_\beta(i)$, $\rho_i = r(i)$, $\sigma_i = s(i)$.
Public input $A_j = g^{a_j} h^{r_j}$, $B_j = g^{b_j} h^{s_j}$

1. Each player $P_i$ shares $\lambda_i \alpha_i \beta_i$
   using Pedersen's VSS protocol. That is let $f_i(x) = f_{it}x^t + \ldots + f_{i0}$ and $u_i(x) = u_{it}x^t + \ldots + u_{i0}$ two random polynomials of degree $t$ such that $f_i(0) = \lambda_i \alpha_i \beta_i$. Player $P_i$ gives to player $P_j$ the values $c_{ij} = f_i(j)$, $\tau_{ij} = u_i(j)$. Player $P_i$ publishes $C_{ij} = g^{f_{ij}} h^{u_{ij}}$ for $j = 0, \ldots, t$.

   Secret information of $P_i$: share $c_{ji}, \tau_{ji}$ of $\lambda_j \alpha_j \beta_j$
   Public information: $C_{ij} = g^{f_{ij}} h^{u_{ij}}$

2. The players verify each other sharing. The players who fail the verification of the VSS protocol are exposed.
3. Players compute $\mathcal{A}_i = g^{\alpha_i} h^{\rho_i} = \prod_{j=0}^{t} A_j^{i^j}$ and $\mathcal{B}_i = g^{\beta_i} h^{\sigma_i} = \prod_{j=0}^{t} B_j^{i^j}$ Require $P_i$ to prove in zero-knowledge that $C_{i0} = g^{\lambda_i \alpha_i \beta_i} h^{u_{i0}}$ is of the correct form with respect to $\mathcal{A}_i$ and $\mathcal{B}_i$. (see Appendix F.) Expose the values of the players who fail the check.
4. Player $P_i$ computes $\gamma_i = \sum_{j=1}^{2t+1} c_{ji}$ which is a share of $\gamma = \alpha\beta$ via a random polynomial of degree $t$. Compute also $\tau_i = \sum_{j=1}^{2t+1} \tau_{ji}$.
5. Player $P_i$ computes $C_j = \prod_{i=1}^{2t+1} C_{ij}$, for $1 \leq j \leq n$.

   Secret information of $P_i$: share $\gamma_i$
   Public information: $C_i$ for $1 \leq i \leq n$

Figure 5: Robust multiplication protocol using Pedersen's VSS

The prover $P$ publishes three commitments: $A = g^\alpha h^\rho$, $B = g^\beta h^\sigma$ and $C = g^{\alpha\beta} h^\tau$. He wants to prove in ZK to a verifier $V$ that he knows how to open such commitments and the opening of $C$ that he knows is really the product of the values he committed to in $A$ and $B$.

The following ZK proof is adapted from a more general one invented by Cramer and Damgard [CD97]. The basic idea is for the prover to prove that he knows that $C$ can be written as $B^\alpha h^{\tau - \sigma\alpha}$.

1. $P$ chooses $d, s, x, s_1, s_2 \in_R Z_q$. He sends to $V$ the messages $M = g^d h^s$, $M_1 = g^x h^{s_1}$, $M_2 = B^x h^{s_2}$.
2. $V$ chooses a challenge $e \in_R Z_q$ and sends it to $P$
3. $P$ replies with the following values: $y = d + e\beta$, $w = s + e\sigma$, $z = x + e\alpha$, $w_1 = s_1 + e\rho$, $w_2 = s_2 + e(\tau - \sigma\alpha)$.
4. $V$ checks that: $g^y h^w = MB^e$, $g^z h^{w_1} = M_1 A^e$ and $B^z h^{w_2} = M_2 C^e$.

The above protocol is only ZK against an honest verifier but can be transformed in a ZK proof against any verifier by standard techniques, i.e. by having the verifier commit to the challenge as a first round.

Notice that the protocol involves only a constant number of exponentiations (i.e. $O(k)$ multiplications).

**Remark:** In our protocol we can exploit the fact that the verifier only sends a random challenge to the prover. Indeed this allows us to run a *single* proof from $P_i$ to *all* the other players. The proof would go as follows: 1) all the other players commit to a random number in $Z_q$; 2) the prover sends the first message; 3) all the players would decommit and the challenge will be computed as the sum of the decommitted values. If the original commitment is non-malleable this is secure.

# G Fast-track Multiplication

Protocol appears in Figure 6.

**FT-Mult: Fast-track Multiplication**

Input of player $P_i$: values $\alpha_i = f_\alpha(i)$, $\beta_i = f_\beta(i)$, $\rho_i = r(i)$, $\sigma_i = s(i)$.
Public input $\mathcal{A}_i = \mathcal{H}(\alpha_i, \rho_i) = g^{\alpha_i} h^{\rho_i}$, $\mathcal{B}_i = \mathcal{H}(\beta_i, \sigma_i) = g^{\beta_i} h^{\sigma_i}$ for $0 \le i \le n$

1. Each player $P_i$ shares $\lambda_i \alpha_i \beta_i$ using the VSS protocol. That is set $c_{ij} = f_{\alpha\beta,i}(j)$, $\tau_{ij} = u_i(j)$ where $f_{\alpha\beta,i}$, $u_i$ are random polynomials of degree $t$ such that $f_{\alpha\beta,i}(0) = \lambda_i \alpha_i \beta_i$.

   | |
   |---|
   | Secret information of $P_i$: share $c_{ji}, \tau_{ji}$ of $\lambda_j \alpha_j \beta_j$ |
   | Public information: $\mathcal{C}_{ij} = g^{c_{ij}} h^{\tau_{ij}}$ for $1 \le i, j \le n$ |
   | $\mathcal{C}_{i0} = g^{c_{i0}} h^{\tau_{i0}}$ for $1 \le i \le n$ |

2. $P_i$ proves in zero-knowledge that $C_{i0}$ is a commitment to the product of $\lambda_i \alpha_i \beta_i$ using the ZK proof from Appendix F. Expose the values of the players who fail the proof.
3. Player $P_i$ computes $\gamma_i = \sum_{j=1}^{2t+1} c_{ji}$ which is a share of $\gamma = \alpha\beta$ via a random polynomial of degree $t$, and $\tau_i = \sum_{j=1}^{2t+1} \tau_{ji}$.
4. Player $P_i$ computes and broadcasts $\mathcal{C}_i = \mathcal{H}(\gamma_i, \tau_i) = g^{\gamma_i} h^{\tau_i} = \prod_{j=1}^{2t+1} \mathcal{C}_{ji}$.
5. Players run a VSPS-Check on $\mathcal{C}_i$ for $1 \le i \le n$. If the test fails STOP and run Multfrom Step 2.

   | |
   |---|
   | Secret information of $P_i$: share $\gamma_i$ |
   | Public information: $\mathcal{C}_i$ for $1 \le i \le n$ |

Figure 6: Fast-track multiplication protocol

# H Fast-track Joint Random VSS protocols

A crucial tool in several cryptographic protocols is a scheme to generate a a random value unknown to all the players which will be shared with the VSPS property. A method to achieve this was introduced by Pedersen [Ped91b]. Each player shares a random value with a VSPS protocol, then these secrets are summed to generate the random secret. Each player checks all the other sharings and then locally sums the shares received by the other players. It is easy to see that such a sum is a share (with the VSPS property) of a randomly distributed secret.

In the following we will denote with Joint-Uncond-VSS a joint VSS that is obtained by the above paradigm with the underlying VSPS protocol being either Pedersen's VSS or our DL-VSS combined with VSPS-Check .

However we observe that if we use DL-VSS as the underlying VSS protocol, we can create a fast track version of this protocol by deferring the verification of the VSPS property only to the combined values. Indeed it is not important if individual sharings do not have the VSPS property, as we are only interested that the final secret will have the property. If the resulting sharing fails the VSPS-Check protocol then we know there are faults in the system and only then we check each individual sharing. The full protocol which we call FT-Joint-DL-VSS is described in Figure 7.

**Fast-Track Joint VSS**

1. Player $P_i$ chooses a random value $r_i$ and shares it using the DL-VSS protocol in Section 2. Denote by $\alpha_{i,j}, \rho_{i,j}$ the shares player $P_i$ gives to player $P_j$. The value $\mathcal{A}_{i,j} = g^{\alpha_{i,j}} h^{\rho_{i,j}}$ is public.
2. The players verify the VSPS property of the sum of the shared secrets by running VSPS-Check on $\mathcal{A}_1, \ldots, \mathcal{A}_n$ where
$$\mathcal{A}_j = \prod_i \mathcal{A}_{i,j}$$
3. If the output of VSPS-Check$= 1$ then player $P_j$ computes his shares $\alpha_j, \rho_j$ of the random secret $r = \sum_i r_i$ by setting $\alpha_j = \sum_i \alpha_{i,j}$, $\rho_j = \sum_i \rho_{i,j}$ otherwise the players run VSPS-Check on each individual sharing from step 1. The values $\alpha_j, \rho_j$ are set to the sum of the shares from the sharings that pass the VSPS-Check protocol.

Figure 7: FT-Joint-DL-VSS

EFFICIENCY GAIN. If there are no faults in the system the protocol FT-Joint-DL-VSS is a factor of $n$ faster than the corresponding Joint-Uncond-VSS since the expensive procedure VSPS-Check is performed only once instead of $n$ times.

# I DSS Threshold Signatures

## I.1 The Digital Signature Standard

The Digital Signature Standard (DSS) [fST91] is a signature scheme based on the El-Gamal [ElG85] and Schnorr's [Sch91] signature schemes. In our description of the DSS protocol we follow the notation introduced in [Lan95].

KEY GENERATION. A DSS key is composed of public information $p, q, g$, a public key $y$ and a secret key $x$, where: $p$ is a prime number of length $l$ where $l$ is a multiple of 64 and $512 \leq l \leq 1024$. $q$ is a 160-bit prime divisor of $p-1$. $g$ is an element of order $q$ in $Z_p^*$. The triple $(p, q, g)$ is public. $x$ is the secret key of the signer, a random number $1 \leq x < q$. $y = g^x \bmod p$ 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 $r = (g^{k^{-1}} \bmod p) \bmod q$ and $s = k(m + xr) \bmod q$ The pair $(r, s)$ is a signature of $m$.

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

Our DSS protocol uses in a crucial way Joint VSS protocols, which allow a set of players to generate a random secret unknown to all of them in a shared form via a VSS protocol. We describe such protocols and a clever way to fast-track them in Appendix H

## I.2 Yet another VSS

In our basic VSS consider yet another implementation of $\mathcal{H}$ directly based on modular exponentiation. That is the dealer shares the secret $\alpha \in Z_q$ with the polynomial $f_\alpha(x) = a_t x^t + ... + a_1 x + \alpha$, gives player $P_i$ the value $\alpha_i = f_\alpha(i)$ and publishes $\mathcal{A}_i = \mathcal{H}(\alpha_i) \stackrel{\text{def}}{=} g^{\alpha_i} \bmod p$. The dealer also publishes $A_0 = g^\alpha$. The reconstruction is as before. Each player $P_i$ broadcasts $\alpha_i$. We accept only those that match the published $\mathcal{A}_i$. We extrapolate the polynomial $\hat{f}_\alpha$ check that, for all $i = 0, \ldots, n$, $\mathcal{A}_i = g^{\hat{f}_\alpha(i)}$. If this check succeeds then $\alpha = \hat{f}(0)$ otherwise $\alpha = 0$.

We name the above protocol FVSS. Although it looks similar to Feldman's VSS [Fel87] it differs from it because in FVSS the public commitments are to the points of the polynomial, while in Pedersen's VSS the commitments are to the coefficient. For this same reason however DL-VSS does not have the VSPS property i.e. it does not insure that the shares lie on a polynomial of degree $t$. However it is easy to see that such property can be checked via a randomized test similar to the one described in Section 4.2.1.

As in Feldman's VSS, FVSS reveals the value $g^\alpha \bmod p$. In general this can be a problem in terms of security. However for the specific application of threshold DSS it is OK to reveal such a value, since it will turn out to be part of the output of the protocol.

A joint version of FVSS can be obtained as in Section H. We will denote with Joint-VSS a joint VSS protocol in which the underlying VSS scheme is either Feldman's VSS or our FVSS with VSPS-Check . We denote with FT-Joint-FVSS the fast-track version of it that can be obtained with FVSS as the underlying VSS.

## I.3 Our Protocol for Threshold DSS signatures

KEY GENERATION. As noted first in [Ped91b], for any discrete-log based scheme, the distributed key generation protocol can be implemented with Joint-VSS. Recall that as a result of this protocol player $P_i$ holds a secret input $x_i$ which is his share of the secret key $x$. The values $g^x$ and $g^{x_i}$ are public.

OUTLINE OF SIGNATURE PROTOCOL. The protocol follows the same structure of the one in [GJKR96b]. First the players generate distributively a random value $k$ by running a Joint-Uncond-VSS protocol. It is necessary that this protocol be unconditionally secure as we do want to reveal $g^k$, which is information not revealed by a DSS signature. To compute $r = g^{k^{-1}} \bmod p \bmod q$ without revealing $k$, the players use a variation of a protocol to compute inverses due to Bar-Ilan and Beaver [BB89]. The idea here is to generate a random value $a$ distributively through a Joint-VSS protocol. Recall that this reveals $g^a$. Compute a sharing $\mu_1, \ldots, \mu_n$ of the value $\mu = ka$ via a multiplication protocol Mult. Notice that although $a$ is shared with a Feldman-based protocol the Mult protocol still works (one just needs to adapt the ZK proof to a special case in which one of the committed values is not information-theoretically secure). Reconstruct $\mu$ by revealing the shares $\mu_i$ (bad players are caught because they cannot contribute bad shares which do not match the commitment). Then, the value $r$ can be publicly computed as $(g^a)^{\mu^{-1}}$. For the generation of the signature's value $s$, the players have to compute a multiplication protocol Mult and a linear combination over the shared values $k$ and $x$ (here once again one has to notice that $x$ is shared via a Feldman-based VSS).

The protocol is described in full in Figure 8.

**Theorem 5** DSS-Thresh *is a secure threshold signature protocol for DSS*

IMPROVEMENTS. What did we gain with respect to the protocol in [GJKR96b]? First of all the use of the simplified multiplication approach allows us to bring the fault-tolerance up to $t = n/2$. This is a dramatic improvement over the fault-tolerance of $t = n/4$ in [GJKR96b]. This does *not* come at the expenses of extra complexity. A close look at the protocol reveals that each player performs 4 VSS's as a dealer and it also

DSS-Thresh

**Private input to player $P_i$:** A share $x_i$ of the secret key $x$.
**Public Input:** The values $g^x, g^{x_1}, \ldots, g^{x_n}$ and the message $m$.

1. **Generate $k$.** The players generate a secret value $k$, uniformly distributed in $Z_q$, by running Joint-Uncond-VSS with two polynomials of degree $t$, $f_k(x)$ and $f_\rho(x)$ such that $f_k(0) = k$ and $f_\rho(0) = \rho$.

   Secret information of $P_i$ : shares $k_i = f(i)$ and $\rho_i = r(i)$
   Public information $g^k h^\rho, g^{k_i} h^{\rho_i},\ 1 \le i \le n$.

2. **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-VSS.

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

   (b) Perform protocol Mult to get shares $\mu_i$, of $\mu = ka \bmod q$ that lie on a polynomial of degree $t$. This also produces random values $\sigma_i$ that lie on a polynomial of degree $t$.

   Secret information of $P_i$ : shares $\mu_i$ and $\sigma_i$
   Public information: $g^\mu h^\sigma, g^{\mu_i} h^{\sigma_i},\ 1 \le i \le n$

   (c) Player $P_i$ broadcasts $\mu_i$, $\sigma_i$. Discard those that do not match $g^{\mu_i} h^{\sigma_i}$. Interpolate the remaining ones to reconstruct $\mu = ka$. Each player $P_i$ computes locally $r \stackrel{\text{def}}{=} (g^a)^{\mu^{-1}} \bmod p \bmod q$.

   Public information: $r$

3. **Generate $s = k(m + xr) \bmod q$**
   (a) Perform a protocol Mult to get shares $s_i$ of $s = k(m + xr) \bmod q$ that lie on a polynomial of degree $t$. This also produces random values $\tau_i$ that lie on a polynomial of degree $t$.

   Private Information of Player $P_i$: shares $s_i$ and $\tau_i$.
   Public information: $g^{s_i} h^{\tau_i}$, $1 \le i \le n$

   (b) Player $P_i$ broadcasts $s_i, \tau_i$. Discard those that do not match $g^{s_i} h^{\tau_i}$. Let $s$ be the free term of the polynomial interpolating the accepted $s_i$'s.

4. **Check and Output.** Output $(r, s)$ as a signature on $m$.

Figure 8: DSS Distributed signature generation

participates to $4(n-1)$ VSS's dealt by other players as a participant. This is the same as in [GJKR96b], but we have an increase in fault-tolerance. This is due to our improved and simplified multiplication protocols. Basically the VSS's used in [GJKR96b] to randomize polynomials of degree $2t$ are replaced in our protocol by VSS's that *at the same time* reduce the degree and randomize the polynomial.

Another nice property of our protocol (which the one in [GJKR96b] does not have) is the possibility of creating a fast-track version as we will see in the next section.

ON-LINE/OFF-LINE BEHAVIOR. It is worth noting that the on-line/off-line behavior of DSS is preserved even under our new protocols. Indeed the value $r$ can be precomputed off-line first. Then $r$ can be used for the computation of $s$ on-line. In order to avoid computing modular exponentiations during the on-line computation of $s$ (because of the VSS's of the values $k_i x_i$) one must precompute the sharings of the values $k_i x_i$ as well.

## I.4 Fast Track version

It is possible to create a fast-track version of the protocol considered above. When run in fast-track mode the protocol will improve its speed by a factor of $n$ if there are no faults in the system. However if a malicious fault happen the protocol has to be resetted and ran in the fully fault-tolerant mode.

OUTLINE. The basic idea of the protocol is to use our DL-VSS and FVSS protocols (instead of Pedersen's and Feldman's VSS) for the joint VSS used during signature generation. This is because using thos protocols will allows us to fast-track the joint VSS's by postponing the VSPS check to the combined secret. Also the FT-Mult protocol is used instead of Mult . This means that the VSPS check is done on the resulting sharing of the product rather than on the single sharings of the players. If a malicious fault is discovered it is important to notice that the fully fault-tolerant protocol starts from the round the fault manifested itself.

USING THE PUBLIC KEY. An additional improvement to the efficiency of the fast-track version can be obtained by performing a weaker multiplication protocol during the computation of $s$. We will not require the players to prove they are sharing the proper value during the multiplication protocol. This may mess up the result of the computation of $s$. But now we can use the public key $y = g^x$ to check that the signature is correct and if it is not just run the fully fault-tolerant multiplication protocol in the last round.

**Remark.** In [GJKR96b] a very simple and efficient protocol is presented for the case of no malicious faults. Players carry out simple secret sharings. One could be tempted to use this protocol for the fast-track case and then do the fully fault-tolerant protocol only if the signature does not match. However we were not able to prove that the first run of the protocol does not reveal information to the adversary. For the same reason the weaker multiplication protocol can be used only at the last round and not during the computation of $r$.

IMPROVEMENTS. The net result is that if there are no malicious faults the players have to perform only *one* VSPS check per round instead of the $n-1$ per round required by the fully fault-tolerant protocol. Thus, we have a reduction of the overall complexity of the protocol by a factor of $n$.