# Revisiting the Distributed Key Generation for Discrete-Log Based Cryptosystems

Rosario Gennaro
IBM T.J.Watson Research

Stanislaw Jarecki
Stanford University

Hugo Krawczyk
IBM T.J.Watson Research

Tal Rabin
IBM T.J.Watson Research

**Abstract**

A Distributed Key Generation (DKG) protocol is an essential component of any threshold cryptosystem. It is used to initialize the cryptosystem and generate its private and public keys, and it is used as a subprotocol, for example to generate a one-time key pair which is a part of any threshold El-Gamal-like signature scheme. Gennaro et al. showed [GJKR99] that a widely-known non-interactive DKG protocol suggested by Pedersen does not guarantee a uniformly random distribution of generated secret keys even in the static adversary model. Furthermore, Gennaro et al. proposed to replace this protocol with one that guarantees a uniform distribution of the generated key but requires an extra round of (broadcast) communication.

We investigate the question whether some discrete-log based threshold cryptosystems remain secure when implemented using the more efficient DKG protocol of Pedersen, in spite of the fact that the adversary can skew the distribution of the secret key generated by this protocol. We answer this question in the positive. We show that threshold versions of some schemes whose security reduces to the hardness of the discrete logarithm problem, remain secure when implemented with Pedersen DKG. We exemplify this claim with a threshold Schnorr signature scheme.

However, the resulting scheme has less efficient security reduction (in the random oracle model) to the hardness of the discrete logarithm problem than the same scheme implemented with the computationally more expensive DKG protocol of Gennaro et al. Thus our results imply a trade-off in the design of threshold versions of certain discrete-log based schemes between the round complexity of a protocol and the size of the modulus.

**Keywords:** Threshold Cryptography. Distributed Key Generation. Discrete Logarithm. Exact Security. Random Oracle Model.

## 1 Introduction

**Distributed Key Generation in Threshold Cryptosystems.** Distributed key generation is a main component of threshold cryptosystems ([Des87, DF89]). It allows a set of $n$ servers, a.k.a. "players", to jointly generate a pair of public and private keys in a such a way that the public key is output in the clear while the private key is shared by the $n$ servers via a threshold secret-sharing scheme [Sha79]. Unless an adversary compromises

more than a specified threshold, e.g., $t < n/2$, out of the $n$ servers, the generated private key remains secret and its secret-sharing can be subsequently used by these servers to jointly compute signatures or decryptions, or perform any other functionality required of the secret key in a cryptosystem. For discrete–log based threshold schemes, distributed key generation amounts to a distributed generation (i.e. without a trusted dealer) of a Shamir secret sharing [Sha79] of a random value $x$, and making public the value $y = g^x$. Following [GJKR99], we refer to such a protocol as DKG.

Torben Pedersen in [Ped91b] proposed a simple DKG protocol. Pedersen's protocol was then used in many discrete-log based threshold cryptosystems, e.g., [CMI93, Har94, LHL94, GJKR96, HJJ+97, PK96, CGS97, SG98, CG99]. It is important to point out that the DKG protocol in some of these threshold cryptosystems is used as a subprotocol every time a signature or decryption needs to be computed. For example, in all threshold implementations of ElGamal-like signatures schemes (e.g., [CMI93, Har94, LHL94, GJKR96, HJJ+97, PK96]), the servers need to generate a secret-sharing of a temporary secret $k$ and a public value $r = g^k$ every time they generate a signature. They accomplish this with an instance of the Pedersen's DKG protocol.

**Requirement of Uniform Distribution of the Private Key.** All the above threshold cryptosystems implicitly assume that the DKG protocol generates the private and public keys with uniform distribution over their prescribed domain. In a *centralized*, i.e. standard, version of any discrete–log based scheme the secret key $x$ is generated uniformly at random in a group $Z_q$ for prime $q$. Similarly, the proposed *threshold* versions of these schemes assume that the distributed key generation protocol they employ generates the private key uniformly in $Z_q$. (Alternatively, the key in both cases is chosen uniformly in a group $Z_{p-1}$ for prime $p$.)

Indeed, it seems necessary that the distribution of the secret key must be the same in the threshold and the centralized case if one attempts to argue the security of the threshold cryptosystem by reducing it to the security of the underlying centralized cryptosystem. For example, to argue that a threshold DSS scheme is as secure as the centralized (standard) DSS scheme, one needs to show that a forgery of a signature under the public key generated by the threshold DSS scheme implies the ability to forge signatures in an attack on a standard DSS scheme, i.e., to forge a signature under the public key generated uniformly at random. It is not clear how to argue this if the public keys generated by the threshold DSS scheme and the standard DSS scheme are chosen from different probability distributions.

However, Gennaro et al. in [GJKR99] showed that Pedersen's DKG protocol fails to generate the secret key (and the public key) with uniform distribution. They showed that an adversary who compromises even two out of $n$ servers can skew the distribution of the generated secret key. While it is not clear if the adversary can control the distribution of the private key in a way that helps him break the cryptosystem that uses this key, the adversary's ability to skew this distribution from uniform means that the above security reduction argument does not hold. Since such reduction was the proof technique employed (most often implicitly) to argue security of existing discrete-log based threshold cryptosystems, Gennaro et al. proposed a new DKG protocol which fixes this problem by generating the private key with a uniform distribution.[1] When the DKG protocol of Gennaro et al. is substituted for

[1] Another DKG protocol which also fixes the problem of Pedersen's protocol, and which moreover can be

Pedersen's DKG in a threshold cryptosystem whose security proof assumes uniform distribution of the generated secret – which is the case of all the discrete-log based cryptosystems mentioned above – this substitution renders the threshold cryptosystem provably secure.

**Problem: Cost of the DKG Protocol.** Unfortunately, the DKG protocol of Gennaro et al. is twice more expensive than the original DKG protocol by Pedersen. While Pedersen's protocol is non-interactive in the absence of faults, the protocol of Gennaro et al. requires two rounds of communication. Moreover, each communication round involves a reliable broadcast, which is a costly operation in a realistic setting like the internet (see e.g. [CP02]). The new DKG protocol also requires about twice more computation from each server.

The computation and communication cost of the DKG protocol is important because for threshold cryptosystems which have the most efficient threshold versions, for example for threshold version of Schnorr's signature scheme, the cost of the DKG protocol is a primary factor in the cost of the scheme. If a threshold scheme is designed to handle on-line requests, and/or the number of requests is large, an increase in the communication and computation costs of the scheme by a factor of two implies a considerable expense. Reducing the cost of this scheme is furthermore worthwhile because with off-line preprocessing it is more efficient than a threshold RSA scheme (see e.g. [Sho00] and the efficiency discussions therein).

**Our Contribution: Dealing with Non-Uniform Distribution on Keys.** We show that a certain type of discrete-log based threshold schemes can remain secure even if it uses Pedersen's DKG protocol as a subprotocol, notwithstanding the fact that this protocol indeed does not generate secrets with uniform distribution. Namely, we show how to prove secure a threshold version of Schnorr's signature scheme [Sch89] instantiated with Pedersen's DKG protocol.

We show that this threshold scheme is secure by exhibiting a *direct* reduction of its security to the hardness of the underlying computational problem. In other words, rather than reducing the security of a threshold signature scheme to the security of the centralized version of this signature scheme, we prove its security "from scratch". Our proofs work because, as it turns out, even though the adversary has some control over the generated public key, we can still embed an instance of the underlying hard computational problem into the part of the public key which is contributed by the uncorrupted players in the Pedersen's DKG protocol. We then show how to translate a successful forgery under the resulting public key into solving the embedded instance of a hard problem.

In this way we avoid the limitations of the existing proof techniques which argue security of threshold signature schemes by exhibiting a reduction to the centralized version of the same signature scheme, which, as we argued above, seems hard to do for threshold schemes which use Pedersen's DKG protocol. Indeed, for this reason our methodology does not apply to threshold version of schemes for which no known reduction to some underlying hard problem exists. Thus for example it will be difficult to use our techniques to imply security of threshold DSS or ElGamal signatures implemented with Pedersen's DKG.

However, it is very likely that our methodology can be applied to showing security of other threshold discrete-log based cryptosystems using the less expensive Pedersen's DKG

used to construct in a generic fashion secure threshold versions of discrete-log based schemes, was proposed by Frankel et al. in [FMY99]. The DKG protocols proposed by Gennaro et al. and Frankel et al. are similar in spirit but the proposal of Gennaro et al. is more efficient.

protocol, if the security of the original centralized versions of these schemes can be reduced to the discrete logarithm assumption, the computational Diffie-Hellman, or some other hardness assumption.

**Implications of our Result: Cost vs. Exact Security** The security proof we exhibit for the threshold Schnorr signature scheme implemented with Pedersen's DKG has one important drawback: the security reduction to the underlying hard problem is less efficient than the security reduction that exist for the centralized version of this scheme. If $q_H$ is the number of hash function queries made by the adversary who breaks the threshold implementation of this signature scheme with probability $\epsilon$, then the discrete logarithm problem can be solved in comparable time with probability only $\epsilon^2/(q_H)^2$. This is a factor of $q_H$ degradation of security compared to the centralized version of this scheme, for which a successful forgery with probability $\epsilon$ implies computation of the discrete log in comparable time with probability $\epsilon^2/q_H$ (by the result of [PS96]). In comparison, the same threshold scheme implemented with the DKG protocol of Gennaro et al. has the same exact security as the centralized scheme.

Since the difficulty of computing discrete logarithms in a $q$-order subgroup of field $F_p$ grows as $\exp(|q|/2)$ and $\exp(|p|^{1/3})$, the $q_H$ degradation in the security reduction implies twice longer $q$ and $2^3 = 8$ times longer $p$, while $q_H^2$ degradation implies three times longer $q$ and $3^3 = 27$ times longer $p$.[2] Because the cost of exponentiation grows at least as $O(|q| \cdot |p|^{1.6})$, the proportion between the computational cost of a threshold Schnorr scheme implemented with Pedersen DKG and the computational cost of a threshold Schnorr scheme implemented with the [GJKR99] DKG is $3^{1+3*1.6}/2^{1+3*1.6} \approx 10$, if the two schemes are to achieve the same security guarantee based on the discrete logarithm assumption. Moreover, the computational cost of either scheme is comparable to the communication delay incurred by one broadcast round which, in the recent implementation of [CP02], takes $100ms$ in the LAN setting and about $1s$ in the internet setting, for a group of 5-7 players. Comparing the two costs (see Section 5 for the detailed comparison), we conclude that even though it requires one more round of reliable broadcast, the threshold Schnorr protocol implemented with DKG of [GJKR99] is still more efficient than the same protocol implemented with Pedersen DKG, *if* the two schemes are to achieve provably same guarantee of security based on the discrete-log assumption.

We point out, however, that the mere existence of a polynomial reduction from some scheme to a discrete logarithm problem can be a *heuristic* argument suggesting that the security of the two problems is similar. If one believes in this heuristics, then both Schnorr's signatures, and the threshold Schnorr signatures built using [GJKR99] DKG, *and* the threshold Schnorr signatures built using Pedersen DKG, can be implemented using the same field in which the discrete logarithm problem is believed to be hard for modern computers, e.g. has the $2^{80}$ security bound. In that case, the threshold signature scheme using Pedersen DKG will be twice more efficient because the cost of broadcast will dominate the delay incurred by running the threshold scheme.

**Organization:** In Section 2 we summarize the communication and adversarial models and the definition of security for the threshold cryptosystems we consider. In Section 3 we recall

[2] See a more detailed explanation in Section 5.

Pedersen's DKG protocol and we give some intuition for why this protocol is good enough for proving security of certain threshold schemes. In Section 4 we recall a simple threshold version of Schnorr's signature scheme using Pedersen's DKG protocol, and we prove its security. In Section 6 we conclude with the discussion of costs vs. security tradeoffs, other potential applications of the method presented here, and some open problems.

# 2 Preliminaries

## 2.1 Computation, Communication, and Adversarial Model

We work in the standard model for threshold signature schemes. For a comprehensive overview of the models for threshold cryptography we refer the reader to [Jar01].

The computation proceeds among a set of $n$ *players* $P_1, \ldots, P_n$ modelled by probabilistic polynomial-time Turing machines, and an adversary $\mathcal{A}$, also modelled as a PPT TM, who submits the messages of his choice to be signed. We assume that the players are connected by a complete network of private (i.e. untappable) and authenticated point-to-point channels because link encryption and authentication can be achieved in this model by standard cryptographic techniques. In addition, the players have access to a dedicated broadcast channel, which can be implemented for example by [CP02].

We assume for simplicity that the initial input of all players is a specification of a discrete-log scheme chosen by a trusted third party, i.e. a triple $(p, q, g)$ where $p$ and $q$ are primes and $g$ is a generator of subgroup $G_q$ of order $q$ in $Z_p^*$. Such triple can be publicly chosen without a trusted party via Bach's algorithm [Bac85].

We consider threshold signature schemes secure in adaptive chosen-message attack [GMR88]. The computation proceeds by the $n$ players first executing a DKG protocol to compute a secret-sharing of some private key $x \in Z_q$ and a corresponding public key $y = g^x \bmod p$. Then every time the adversary requests a signature on some message of his choice, the players perform the threshold signature protocol on that message.

In addition to requesting signatures on adaptively chosen messages, the adversary can corrupt up to $t$ of the $n$ players in the network, for any value of $t < n/2$, which is the best achievable threshold. The adversary can cause the corrupted players to arbitrarily divert from the specified protocol. We assume that the adversary is *static*, i.e. that he chooses the corrupted players at the beginning of the protocol.

We assume a realistic *partially synchronous* communication model, i.e. that computation proceeds in synchronized rounds and that the messages are received by their recipients within some specified time bound. To guarantee this round synchronization, and for simplicity of discussion, we assume that the players are equipped with synchronized clocks. Notice that this model messages sent from the uncorrupted players to the corrupted ones can still be delivered relatively fast, in which case, in every round of communication, the adversary can wait for the messages of the uncorrupted players to arrive, then decide on his computation and communication for that round, and still get his messages delivered to the honest parties on time. Therefore we should always assume the worst case that the adversary speaks last in every communication round. In the cryptographic protocols literature this is also known as a *rushing* adversary.

## 2.2 Notation, Assumptions, Security Definitions, Proof Methodology

**Negligible Probability.** We call function $f$ *negligible* if for every polynomial $P(\cdot)$, $f(k) \leq (1/P(k))$ for all sufficiently large $k$. We say that some event occurs with a *negligible probability* if the probability of this event is a negligible function of the security parameter $k$.

**Assumption 1 (Discrete Log Assumption)** *Let $PRIMES(k)$ be the set of poly$(k)$-bit primes $p$ such that there exists a $k$-bit prime $q$ dividing $p-1$. For every probabilistic polynomial time algorithm $I$, for every $p$ in $PRIMES(k)$, probability* $\Pr[g \leftarrow G_q; x \leftarrow Z_q; I(p,q,g,g^x) = x]$ *is negligible, where $G_q$ is the subgroup of $Z_p^*$ of order $q$.*

**Notion of Security for a Threshold Signature Scheme** We call a threshold signature scheme *secure* with adversarial threshold $t$, if it is both *robust* and *unforgeable*. A threshold scheme is robust if in the presence of a threshold adversary, the threshold signature protocol produces a valid signature on the requested message, except for at most negligible probability. A scheme is unforgeable if, after an execution of the distributed key generation protocol which produces some public key, and after participating as a threshold adversary in an execution of a polynomial number of runs of the threshold signature protocol on the messages of adversary's choice, the adversary's probability of forgery, i.e. of producing a valid signature under the generated public key on some new message, is negligible.

**Simulation Proof Technique.** We will argue unforgeability of a threshold signature scheme using the standard technique of *simulation* of adversary's view in the distributed computation. Namely, to prove that the scheme is unforgeable based on some cryptographic assumption, say the above discrete log assumption, we will exhibit an efficient algorithm called a *simulator*, which on the input of a random instance of the hard problem, say an instance of the discrete logarithm problem, can simulate to the adversary his view of an execution of the threshold protocol in such a way that (1) this view is indistinguishable from a view of an actual random execution of the threshold protocol; and (2) if the adversary succeeds in forging a signature under the generated public key then the simulator can translate this forgery into solving the input instance of the hard problem. An existence of such a simulator algorithm concludes the proof because if the adversary has a non negligible probability of forgery against the threshold scheme then, by the simulation property (1), he will produce such a forgery during the simulation. But then, by the simulation property (2), the simulator will be able to solve the input instance of the supposedly hard problem.

**Random Oracle Model.** Our proofs of security are in so-called Random Oracle Model [BR93], i.e. we model a hash function like MD5 or SHA1 as ideal random oracles in the sense that if an adversary who makes some number of queries to the fixed hash function has some (non negligible) probability of success in, for example, computing a forgery against a threshold signature scheme, the same adversary should have the same probability of success if instead of a fixed hash function like MD5 or SHA1, he accesses a truly random function.

**Notation.** All arithmetic operations in this paper are performed in some finite group or field, in fact all operations are modulo some prime, either $q$ or $p$. We will often omit writing mod$p$ or mod$q$ to simplify the notation, but the proper modulus should always be clear from the context: operations on secret values chosen in $Z_q$, like $x$ and $k$, are modulo $q$, while all modular exponentiation operations and operations on public values like $y$ or $r$ are computed modulo $p$.

## 3 Pedersen's Distributed Key Generation Protocol

In Pedersen's DKG protocol [Ped91b], each player shares a randomly chosen secret $x_i$ using Feldman's verifiable secret sharing (VSS) protocol [Fel87]. The secret value $x$ is the sum of the properly shared $x_i$'s. Since Feldman's VSS has the additional property of revealing $y_i = g^{x_i}$, the public value $y = g^x$ is the product of the $y_i$'s that correspond to those properly shared $x_i$'s. We present this protocol, denoted Ped-DKG, in Figure 1. For completeness we recall Feldman's VSS protocol and discuss its security properties in Appendix A.

We will require that the reconstruction of the secret $x$ shared in a Ped-DKG protocol, proceeds via $n$ parallel reconstructions of the Fel-VSS protocol for all shared secrets $x_i$, and then $x$ is computed as $x_1 + \ldots x_n \bmod q$. Note that the secret can also be reconstructed by every player $P_i$ submitting his *polynomial* share of $x$, $\overline{x}_i = \sum_j \overline{x}_{ji}$. This is indeed a standard reconstruction procedure in threshold protocols. However, our proof of security requires the "additive" rather than "polynomial" secret reconstruction. Note that the two reconstruction procedures have similar cost both in the absence of faults and with faults.

**Distributed Key Generation Protocol** Ped-DKG

1. Each player $P_i$ chooses a random polynomial $f_i(z)$ over $Z_q$ of degree $t$:

   $$f_i(z) = a_{i0} + a_{i1}z + \ldots + a_{it}z^t$$

   $P_i$ broadcasts $X_{ik} = g^{a_{ik}} \bmod p$ for $k = 0, \ldots, t$. Denote $a_{i0}$ by $x_i$ and $X_{i0}$ by $y_i$. Each $P_i$ computes the shares $\overline{x}_{ij} = f_i(j) \bmod q$ for $j = 1, \ldots, n$ and sends $\overline{x}_{ij}$ secretly to player $P_j$.

2. Each $P_j$ verifies the shares he received from the other players by checking for $i = 1, \ldots, n$:

   $$g^{\overline{x}_{ij}} = \prod_{k=0}^{t} (X_{ik})^{j^k} \bmod p \quad (1)$$

   If the check fails for an index $i$, $P_j$ broadcasts a *complaint* against $P_i$.

3. If more than $t$ players complain against a player $P_i$, then that player is clearly faulty and he is disqualified. Otherwise $P_i$ reveals the share $\overline{x}_{ij}$ matching Eq. 1 for each complaining player $P_j$. If any of the revealed shares fails this equation, $P_i$ is disqualified. By convention, the secret shared by a disqualified player $P_i$ is always set to $x_i = 0$ and $y_i = 1$.

4. The public value $y$ is computed as $y = \prod y_i \bmod p$. The secret shared value $x$ itself is not computed by any party, but it is equal to $x = \sum x_i \bmod q$. Each player $P_i$ keeps all the values it received during these $n$ parallel Fel-VSS runs for the purpose of verification in subsequent secret reconstruction.

Figure 1: Pedersen's Distributed Key Generation protocol

The robustness property of the Ped-DKG protocol follows from robustness of Fel-VSS. The secret $x$ is uniquely defined at the end of the protocol and no malicious behavior of up to $t < n/2$ servers can prevent its reconstruction.

**Secrecy Property of Ped-DKG.** The secrecy property of the Ped-DKG protocol is somewhat murky. Namely, it is not clear if the adversary learns anything more in some useful sense about the shared value $x$ than is revealed by its public counterpart $y = g^x$. As Gennaro et al. [GJKR99] point out, a standard formalization of a secrecy property fails, because the adversary controls to some extent the distribution of public keys $y$ output by this protocol, and hence it is impossible to simulate this protocol on input $y$ chosen uniformly at random in $G_q$.

As Gennaro et al. observe, the adversary can, for example, cause this protocol to output only even $y$'s as follows. Assume wlog that the adversary corrupts players $P1, \ldots, P_t$. Call those "bad" players and call the remaining players "good". Let $x_G = x_{t+1} + \ldots + x_n$ be the contribution of the good players to secret $x$, and let $x_B$ be the summary contribution of the bad players to $x$, i.e. $x_B = \sum_{P_i \in \{1,\ldots,t\}} x_i$. Then secret $x$ shared in the DKG protocol is $x = x_G + x_B$. Because the adversary can always speak last (see section 2), he can choose his contribution $x_B$ *after* seeing $y_G = \prod_{P_i \in \{t+1,\ldots,n\}} y_i = g^{x_G}$ in step 1 of the DKG protocol. Consequently, he is free to choose $x_B$ so that the resulting value $y = y_G * y_B = y_G * g^{x_B}$, has some property that he wants to achieve. For example, we can see that by trying out at random potential values for $x_B$, a polynomial-time adversary can fully control $O(\ln k)$ bits of $y$, where $k$ is the security parameter.

However, even though the adversary has some control over the resulting $y$, and hence the secrecy of the Ped-DKG protocol is unclear, it turns out that for the purpose of proving security of threshold cryptosystems which use the Ped-DKG protocol, the following "secrecy-like" property of this protocol is good enough: The public key $y$ it produces is a product of $y_G$ chosen with uniform distribution by the good players, and of value $y_B = g^{x_B}$, where $x_B$ is the contribution of the bad players into the computation. Because this contribution $x_B$ can be reconstructed by the good players via interpolation, we can show that under the discrete log assumption, for any polynomial-sized set of values $Y \subset G_q$, the adversary cannot choose his contribution $x_B$ in such a way that the resulting $y = y_G g^{x_B}$ belongs to set $Y$. If he did, then let $y_{max} \in Y$ be the value that he is most likely to hit. On input a random $y_T$ in $G_q$, a simulator playing on behalf of the good players chooses their contribution as $y_G = y_{max}/y_T$. Since $y_T$ is random in $G_q$, so is $y_G$. Moreover, by the same argument that one uses to argue secrecy of Feldman's VSS protocol (see Appendix A), the simulator can perfectly simulate the adversary's view of the contribution of the good players to any value $y_G$ which is uniformly distributed in $G_q$. Then, if the adversary creates $x_B$ s.t. $y = y_G g^{x_B} = y_{max}$ then $x_B = dlog_g(y_T)$, and hence the simulator would solve the discrete log problem on a random instance $y_T$.

In other words, even though the simulator cannot control the chosen value $y$, the adversary cannot control it too much either. In particular, he has only negligible chance of making $y$ fall in any polynomially-sized set. In the next section we will see how this property enables simulation of certain threshold signature schemes that use the Ped-DKG protocol to create the commitment in the three-round zero-knowledge identification scheme.

# 4 Threshold Schnorr Signature Scheme using Ped-DKG

## 4.1 Schnorr's signature scheme

We first recall Schnorr's signature scheme [Sch89]. As before, let $p, q$ be primes and let $g$ be a generator of subgroup $G_q$ of order $q$ in $Z_p^*$. Let $H$ be a hash function, $H : \{0,1\}^* \rightarrow Z_q$, which we will model as an ideal random function. The private key is $x$, chosen at random in $Z_q$. The public key is $y = g^x \bmod p$. A signature on message $m$ is computed as follows. The signer picks a one-time secret $k$ at random in $Z_q$, and computes the signature on $m$ as a pair $(c, s)$ where $s = k + cx \bmod q$, $c = H(m, r)$, and $r = g^k \bmod p$. Signature $(c, s)$ can be publicly verified by computing $r = g^s y^{-c} \bmod p$ and then checking if $c = H(m, r)$.

This signature scheme follows a methodology introduced by Fiat and Shamir [FS86], which converts any three-round *commit-challenge-response* zero-knowledge identification scheme where the challenge is a public coin into a signature scheme. This is done by replacing the random challenge chosen by the verifier with an output of a random function $H$ computed on the message and the prover's commitment. In this case the prover's commitment is $r = g^k$, the challenge is $c = H(m, r)$, and the prover's response is $s = k + cx$. In the random oracle model, the unforgeability of this scheme under a chosen-message attack [GMR88] reduces to the discrete log assumption, as proven by [PS96].

We recall this proof here because it helps in understanding of the proof of security of the *threshold* Schnorr signature scheme below. The simulator, on input $y$, can produce Schnorr's signatures on any $m$ by picking $s$ and $c$ at random in $Z_q$, computing $r = g^s y^{-c}$ and setting $H(m, r) = c$. This simulator can also translate the adversary's forgery into computing $dlog_g y$ as follows. It runs the adversary until the adversary outputs a forgery $(c, s)$ on some message $m$. Note that because $H$ is a random function, except for negligible probability, the adversary must ask to $H$ a query $(m, r)$ where $r = g^s y^{-c}$, because otherwise it could not have guessed the value of $c = H(m, r)$. The simulator then rewinds the adversary, runs it again by giving the same answers to queries to $H$ until the query $(m, r)$, which it now answers with new randomness $c'$. If the adversary forges a signature on $m$ in this run, then, except for negligible probability, it produces $s'$ s.t. $r = g^{s'} y^{-c'}$, and hence the simulator can now compute $dlog_g y = (s - s')/(c' - c)$. One can show that if the adversary's probability of forgery is $\epsilon$, this simulation succeeds with probability $\epsilon^2/4q_H$: $O(\epsilon)$ probability that the adversary forges in the first run times the $O(\epsilon/q_H)$ probability that it will forge on the second run *and* that it will choose to forge on the same $(m, r)$ query out of its $q_H$ queries to $H$. We refer to [PS96] for the full proof.

## 4.2 Threshold version of Schnorr's scheme using Pedersen's DKG

The threshold version of Schnorr's scheme using Pedersen's DKG protocol Ped-DKG is a very simple protocol. It works in the straightforward way as all standard threshold dlog-based protocols, e.g. [GJKR96], except that secret reconstruction is done on additive rather than polynomial shares.[3]

To initialize the threshold signature scheme, first the Ped-DKG protocol is executed for distributed key generation, i.e. it outputs secret-sharing of a private key $x$ and a public

[3]Replacing computation on polynomial shares with computation on additive shares often seems necessary for the proof of security of a threshold protocol to go through. This technique was used before to handle an adaptive adversary in [FMY99, CGJ+99] or to handle the no-erasure and concurrent adversary in [JL00].

**Threshold Signature Protocol** TSch

**Inputs:** Message $m$ to be signed, plus the secret-sharing of $x$ generated by the initial Ped-DKG protocol. In particular, each player $P_i$ holds an additive share $x_i$ of $x$ while values $y = g^x$ and $y_i = g^{x_i}$ for every $P_i$ are public.
**Outputs:** Schnorr's signature $(c, s)$ on $m$.

1. Players perform an instance of the Ped-DKG protocol (Figure 1). Denote the outputs of this run of Ped-DKG as follows. Each player $P_i$ holds an additive share $k_i$ of the secret-shared secret $k$. Each of these additive shares is itself secret-shared with Fel-VSS. We denote the generated public values $r = g^k$ and $r_i = g^{k_i}$ for each $P_i$.
2. Each player locally computes the challenge $c = H(m, r)$.
3. Players perform the reconstruction phase of Feldman's secret-sharing of value $s = k + cx$ as follows. Each player $P_i$ broadcasts its additive share $s_i = k_i + cx_i$. Each share is verified by checking if $g^{s_i} = r_i y_i^c$. Otherwise $x_i$ and $k_i$ are reconstructed and $s_i$ is computed publicly. The protocol outputs signature $(c, s)$ where $s = s_1 + \ldots + s_n$.

Figure 2: Threshold Signature Protocol for Schnorr's Signature Scheme

value, the public key $y = g^x$. Then the threshold Schnorr signature protocol TSch proceeds as follows on input a message $m$ (see Figure 2). First the players run an instance of the Ped-DKG protocol to generate a secret-sharing of the one time secret $k$ and the public value $r = g^k$. Then each player locally computes the challenge $c = H(m, r)$, and each player $P_i$ broadcasts its *additive* share $s_i = k_i + cx_i$, where $k_i, x_i$ are $P_i$'s additive shares of $k$ and $x$ respectively. Notice that these shares can be publicly verified by checking if $g^{s_i} = K_{i0}(X_{i0})^c$, where $X_{i0} = y_i = g^{x_i}$ and $K_{i0} = r_i = g^{k_i}$ are verification values broadcasted by player $P_i$, during the initial key-generation Ped-DKG protocol, and the Ped-DKG protocol that generates $r$ and the sharing of $k$, respectively. If verification fails for some $P_i$, the players reconstruct the Feldman secret-sharing of both $x_i$ and $k_i$ and compute $s_i$ publicly. Finally, $s$ is computed as $s = s_1 + \ldots + s_n$. Therefore secret $s$ can be efficiently reconstructed as in the Fel-VSS protocol.

**Efficiency Considerations.** Note that without faults, the above protocol requires only one round of broadcast during the Ped-DKG protocol of Step (1). Recall that broadcast is the most important factor in the delay incurred by threshold protocols in the internet setting. This is true if the verifier requesting the signature communicates directly with every player $P_i$. When $P_i$ receives and validates a message $m$ to be signed, it triggers the Ped-DKG protocol of Step 1 and broadcasts along it the message $m$. This allows the players to detect if the verifier submitted inconsistent requests to them. If no faults occur, there is only one more round of broadcast, in Step 3 of TSch, but this can be avoided if every player $P_i$ sends to the verifier the value $c$ and its share $s_i$. The verifier can check if signature $(c, s) = (c, s_1 + \ldots + s_n)$ is valid, and the protocol falls back to the robust reconstruction only if the signature does not verify. The same protocol implemented with the DKG protocol of Gennaro et al. has two rounds of broadcast without faults. Note also that Step (1) of

the protocol can be performed off-line, and thus with preprocessing the threshold Schnorr signature is non-interactive (without faults).

**Security of Threshold Signature Scheme (Ped-DKG,TSch).** The robustness of the (Ped-DKG,TSch) threshold signature scheme follows straightforwardly from the robustness of the Ped-DKG protocol. The interesting part is the proof of unforgeability.

**Theorem 1** *Under the discrete log assumption, the threshold signature scheme (*Ped-DKG, TSch*) is unforgeable in the static adversarial model with adversarial threshold $t < n/2$, in the random oracle model.*

**Proof:** Assume that there exists an adversary which breaks the unforgeability property of this signature scheme. We'll construct a simulator SIM which, using this adversary, will compute a discrete logarithm on input a random challenge value $y_T$ in $G_q$. In the course of this simulation, SIM answers adversary's queries to oracle $H$ at random, except in a crucial case specified below. Let $B$ be the set of corrupted (i.e. "bad") players and $G$ be the set of good players. Assume wlog that $P_n \in G$. Let $q_H, q_S$ be the upper bound on the number of $H$ queries and the number of the signature queries, respectively, that the adversary makes.

To compute discrete logarithm of a target value $y_T$, the simulator SIM embeds it in the value $y_n$ contributed by player $P_n$ to the public key generated in the initial Ped-DKG protocol. In other words, the simulator follows the Ped-DKG protocol on behalf of players $P_i \in G \setminus \{n\}$ as prescribed, but for player $P_n$ the simulator *simulates* the adversarial view of the Fel-VSS protocol performed as a part of Ped-DKG by player $P_n$ so that public value $y_n$ broadcasted by $P_n$ is equal to the target value $y_T$. (In Appendix A we recall how such simulation of Fel-VSS should be done.)

After simulating the initial key generation protocol in this way, SIM, for every message $m$ submitted by the adversary for a signature, simulates an execution of TSch on this message as follows. SIM chooses at random in $Z_q$ values $c$ and $s_n$, computes $r_n = g^{s_n} y_n^{-c}$, and in Step (1) of TSch it follows the protocol on behalf of players $P_i \in G \setminus \{n\}$ as prescribed, but for $P_n$ it again simulates the Fel-VSS protocol in this step so that the value $r_n$ broadcasted by that player is what the simulator wants, i.e. $r_n = g^{s_n} y_n^{-c}$.

The simulator's goal is to make sure that whatever value $r$ is computed in this step, the output of the $H$ oracle on input $(m, r)$ can be set to $c$. In that case the simulator can simulate the rest of the TSch protocol. The value $H(m, r)$ will be computed as $c$ in Step (2), and in Step (3) the simulator can follow the protocol on behalf of players $P_i \in G \setminus \{n\}$ as prescribed, while for $P_n$ it can publish the previously chosen value $s_n$, and since $g^{s_n} = r_n y_n^c$, this value passes. Moreover, this is a valid signature because $g^s = g^{\sum s_i} = \prod(g^{s_i}) = \prod(r_i y_i^c) = (\prod r_i)(\prod y_i)^c = r y^c$, and $c = H(m, r)$.

To extract with non-negligible probability the value $c$ as the output of $H$ on $(m, r)$ where $r$ is computed in Step (1) of TSch, the simulator SIM, after computing values $r_i \in G$ in the simulation of Step (1) as described above, but before sending any messages in this simulation to the adversary, flips a coin $b$. If $b = 0$, then during the simulation of Step (1) SIM answers all the (new) queries the adversary makes to $H$ at random. If the value $r$ computed in Step (1) is such that $H$ was already asked on $(m, r)$, the simulator fails. Otherwise, SIM sets $H(m, r)$ as $c$ and succeeds. If $b = 1$, then the simulator chooses at random an index $i \in \{1, \ldots, q_H\}$ and if the adversary makes any $(m, r^{(j)})$ queries to $H$ *after* the simulator publishes all values $y_i \in G$ but *before* the adversary shares his values $x_i \in B$

(these are sub-steps of the simulation of Step (1)), the simulator answers $i$-th such query with $c$. If value $r$ output in this Step (1) is equal to $r^{(i)}$, this is simulator's success because $H(m, r) = c$. Otherwise the simulator fails.

We try to upper-bound the simulator's probability of success. We use the intuition outlined earlier in the discussion of security of the Ped-DKG protocol in Section 3. If the adversary chooses his contribution $r_B = \prod_{i \in B} r_i$ to $r$ so that $H$ was not asked on $r$, the simulator succeeds with probability 1/2. Let $\epsilon_{hit}$ be the probability which with the adversary, after seeing $r_G = \prod i \in G r_i$, chooses his contribution $r_B$ so that it "hits" some value $r = r_G r_B$ on which oracle $H$ has been queried *before* the simulation of Step (1) starts. Therefore, if $b = 1$ and the output $r$ of Step (1) is such that $H$ has been queried $(m, r)$ either before the simulation of this step or during this simulation, the probability that $r = r^{(i)}$ chosen by the simulator is upper bounded by $1/q_H - \epsilon_{hit}$, the probability that SIM guesses the right index $i$ minus the probability that the adversary hits some value $r$ on which the oracle $H$ was queried before the simulation. Assuming that $\epsilon_{hit}$ is small, the probability that the simulator successfully passes the simulation of this run of the TSch protocol can be upper-bounded as $\epsilon_{ss} = 1/(2q_H) - \epsilon_{hit}$. Therefore, if SIM repeats this simulation $c/\epsilon_{ss}$ times, the probability that it fails is at most $e^{-c}$. In this way, with probability $(1 - 1/e^c)^{q_S}$, the simulator successfully goes through the simulation of each of the $q_S$ instances of the TSch protocol. Setting $c = \ln 2q_S$, this probability is more than a half.

Since the adversary's view in this simulation is the same as in the protocol execution, then assuming that the adversary forges with not negligible probability $\epsilon$, the simulator will get some forged signature $(c, s)$ on some message $m$ with probability $\epsilon/2$. By applying the same "forking lemma" argument as [PS96] used to prove the security of the standard Schnorr signature (see the beginning of this section), we can argue the following. SIM, with probability at least $\epsilon/4$ gets one forgery *and* if he re-winds this adversary to the point when the adversary asks query $H(m, r)$ where $r = g^s y^{-c}$, and then continues to simulate from that point on with fresh randomness (in particular answering this query to $H$ with fresh randomness $c'$), then SIM has about $(\epsilon/4)/q_H$ chance of getting a second forgery on the same message $m$ but relative to a different random function $H$. In this case SIM gets two pairs $(c, s)$ and $(c', s')$ s.t. $r = g^s y^{-c} = g^{s'} y^{-c'}$ and thus can compute $dlog_g y$ and hence also $dlog_g y_n = dlog_g y_T$ because SIM knows all $x_i = dlog_g y_i$ for $P_i \neq P_n$.

Therefore, if the assumed threshold signature forger runs in time $T$ and succeeds in forgery with probability $\epsilon$, then if probability $p$ is small, SIM computes the discrete logarithm in time $2c/\epsilon_{ss} * T = 2 \ln(2q_S)/\epsilon_{ss} * T$ with probability at least about $\epsilon^2/(16q_H)$. If $T$ is big enough so that $\epsilon \approx 1$ then the simulator's computation is longer than the adversary's computation by the factor $\alpha = 16q_H * 2 \ln 2q_S/\epsilon_{ss}$. If $\epsilon_{hit} < 1/(4q_H)$, then $\epsilon_{ss} = 1/(2q_H) - \epsilon_{hit} > 1/(4q_H)$, and hence $\alpha < 2^7 q_H^2 \ln(2q_S)$. Taking $q_S < 2^{30}$, we get $\alpha < 2^9 q_H^2$ factor of security degradation.

Now, if $\epsilon_{hit}$ is not small enough then we can solve discrete logarithm otherwise, following the argument in Section 3. Recall that $\epsilon_{hit}$ is the probability which with the adversary, after seeing $r_G = \prod i \in G r_i$, chooses his contribution $r_B$ so that it "hits" some value $r = r_G r_B$ on which oracle $H$ has been queried *before* the simulation of Step (1) starts. Note that there are at most $q_H$ of such values. Let $r_{max}$ be the value that the adversary is most likely to hit. On input a random $y$ in $G_q$, the simulator chooses the contribution of the good players as $r_G = r_{max}/y$. Since $y$ is random in $G_q$, so is $r_G$. The simulator can simulate an execution of Ped-DKG so that the good players' contribution is $r_G$ by the secrecy property of Fel-VSS.

Then, if the adversary does hit the value $r_{max}$, which happens with probability $\epsilon_{hit}/q_H$, and supplies $k_B$ s.t. $r = r_G r_B = r_G g^{k_B} = r_{max}$ then $k_B = dlog_g(y)$, and hence the simulator solves the discrete log problem on a random instance $y$ with probability $\epsilon_{hit}/q_H$. Hence if $\epsilon_{hit} > 1/(4q_H)$, we can reduce this adversary to an attack on the discrete logarithm scheme with similar factor $4q_H^2$ of security degradation. *qed.*

# 5 Security vs. Efficiency Implications

We showed that even though Pedersen's DKG protocol Ped-DKG does not generate secret keys with uniform distribution, it generates them randomly enough for us to show security for a threshold Schnorr signature protocol TSch implemented with Ped-DKG. This is an improvement because with previously known proof methods, discrete-log based schemes had to use the DKG protocol of Gennaro et al which requires two rounds of broadcast instead of one round incurred by Pedersen's DKG. This is especially important for a scheme like Schnorr's, whose main cost lies in the DKG subprotocol that it uses, and which is also the most efficient discrete-log based threshold signature scheme, so reducing its commmunication cost by half is worthwhile.[4]

However, the security reduction we are able to show for the TSch threshold Schnorr signature scheme has a $q_H^2$ factor of security degradation compared to the security of the discrete log problem [DLP]. This is a factor of $q_H$ degradation over provable security of the security of the centralized version of Schnorr signatures. Recall that Pointcheval-Stern [PS96] show a reduction from Schnorr signatures to the DLP which has a $q_H$ factor of security degradation. Because the security reduction from the threshold Schnorr scheme that uses the DKG protocol of Gennaro et al (let's denote this threshold Schnorr scheme gjkr-TSch) to the centralized Schnorr signatures is tight, it follows that there is also a $q_H$ factor in the degradation of (provable) security between the TSch scheme and the gjkr-TSch scheme.

The degradation in the provable security can be interpreted in two ways. One can ignore it and take the mere existence of a polynomial reduction from some scheme to the DLP as a good coin, and claim that since the two problems are shown to be polynomially related, one can securely implement the scheme in question over a field in which DLP is believed to be hard. This is, however, only a heuristic argument. Formally, existence of a security reduction from some scheme to DLP with a degradation factor $f$ implies that if one takes $b$ as a target security bound – i.e. if one wants to claim that the constructed scheme is secure against an adversary performing about $b$ operations – then one needs to use a group in which DLP is believed to be hard against an adversary performing about $b \cdot f$ operations. Therefore, the less efficient reduction for the TSch scheme means that the TSch scheme should be implemented over a larger field to guarantee the same $b$ security bound. In fact, the increase of the computation time resulting from the fact that the TSch scheme needs to work over a larger field to guarantee the same security as the gjkr-TSch scheme outweighs the benefits resulting from the fact that the TSch scheme requires only one round of broadcast while gjkr-TSch needs two.

[4] In fact, the on-line part of the computation in threshold Schnorr signature is more efficient than in threshold RSA. On the other hand, threshold RSA can be fully non-interactive, so its overall cost, as opposed to on-line cost, is smaller.

More specifically, if we take $b = 2^{80}$ as the target security bound, and assume that $q_H \approx 2^{80}$ as well, then the [PS96] results imply that Schnorr's signatures can be securely run in a group with at least $2^{80} \cdot 2^{80} = 2^{160}$ DLP security. Because the security reduction from the gjkr-TSch scheme to the Schnorr signatures is tight, this implies that the gjkr-TSch scheme is secure in the same $2^{160}$-DLP group. On the other hand, the security reduction of Theorem 1 implies that the TSch scheme is secure in a group with $2^{80} \cdot 2^{160} = 2^{240}$ security of the DLP. In other words, threshold Schnorr with [GJKR99] DKG implies factor $\alpha = 160/80 = 2$ growth in the DLP security parameter, while threshold Schnorr using Pedersen DKG implies factor $\alpha = 240/80 = 3$ growth in the DLP security parameter.

Recall that the cost of exponentiation modulo $p$ with an $|q|$-bit exponent grows at best like $O(|q| \cdot |p|^{1.6})$. If we consider the DLP security in the classic number field setting, the factor $\alpha$ growth in the security parameter implies factor $\alpha^3$ growth in the size of modulus $p$ and factor $\alpha$ growth in the size of the modulus $q$. Therefore the overall cost of exponentiation grows like $\alpha \cdot (\alpha^3)^{1.6} = \alpha^{5.8}$. For elliptic curves, the sizes of both $p$ and $q$ grow by factor $\alpha$, and hence the overall cost of exponentiation grows only by factor $\alpha^{2.6}$.

As a reference points we take the performance table of [Wei00], where on a Celeron 850 MHz an exponentiation in a number field with $|p| = 1024$ and $|q| = 160$ takes 2 ms and an exponentiation over a 168-bit elliptic curve takes 5 ms. We assume (after Lenstra and Verhaul [LV01]) that in both settings DLP has security parameter 80. If $n$ is the number of players and $t \approx .5 \cdot n$, then in the TSch threshold Schnorr protocol, each player makes $1.5 \cdot n$ long exponentiations, while in the threshold Schnorr implemented with the [GJKR99] DKG protocol (let's denote this protocol as gjkr-TSch) each player performs $2.5 \cdot n$ long exponentiations.

Taking it all together, for a threshold system with $n = 7$ players, in the standard number field setting, each player's computation in the TSch protocol with $2^{80}$ security guarantees would take about $11 \cdot 3^{5.8} \cdot 2ms = 12.8s$, while in gjkr-TSch each player's computation takes only about $18 \cdot 2^{5.8} \cdot 2ms = 2s$. In this setting, the gjkr-TSch scheme is a winner, because. taking the SINTRA implementation of broadcast [CP02] as a reference point, a round of reliable broadcast between about 7 players would take about $1s$ on the internet and only about $100ms$ on LAN, and therefore the computation cost incurred by TSch outweighs the communication delay caused by an extra round of broadcast incurred by gjkr-TSch.

The TSch scheme schemes maybe slightly faster than gjkr-TSch in the elliptic curve setting with players distributed over the internet. There the player's computation is $18 \cdot 2^{2.6} \cdot 5ms = .5s$ for gjkr-TSch, and $11 \cdot 3^{2.6} \cdot 5ms = .95s$, and so the $.45s$ difference in computation is outweighed by the approximately $1s$ difference in delay caused by the extra broadcast round. On the other hand, the gjkr-TSch scheme still wins with TSch if the $n$ players are connected via a LAN, where the extra round of broadcast costs only about $.1s$.

On the other hand, if one takes our reduction as a heuristic argument that the TSch threshold Schnorr signature scheme with security parameter 80 can be achieved in fields where DLP also has security parameter 80, then the TSch scheme would win with gjkr-TSch: The computation time per player in TSch would be only about $11 \cdot 5ms = 55ms$ compared with $18 \cdot 5ms = 90ms$ for gjkr-TSch. Moreover the TSch scheme, having only one round of broadcast would incur a communication delay of about $100ms$ in the LAN setting or $1s$ in the internet setting, while the gjkr-TSch scheme would take, respectively, $200ms$ and $2s$.

## 6 Open Problems

We point out that it is very likely that our methodology can be applied to showing security of other threshold discrete-log based cryptosystems implemented with the less expensive Pedersen's DKG protocol, if there is a security reduction to the original centralized versions of these schemes from the discrete logarithm assumption, or from some other computational assumption in the discrete-log setting, e.g. the computational Diffie-Hellman assumption. It is clear, for example, that our methodology applies to a threshold version of Chaum-Pedersen undeniable signature protocol [CP92], which is secure under the Diffie-Hellman assumption. Other candidates for this methodology include, but are not limited to, the threshold version of Cramer-Shoup encryption scheme given in [CG99] or the "Modified ElGamal" signature scheme of [PS96].

## References


[Bac85] E. Bach. Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms ACM Distiguished Dissertation (1984). MIT Press, Cambridge, MA, 1985.

[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 Computation*, pages 201–209, 1989.

[BR93] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[CG99] R. Canetti and S. Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In *Eurocrypt '99*, pages 90–106, 1999. LNCS No. 1592.

[CGJ+99] Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *Proc. CRYPTO 99*, pages 98–115. Springer-Verlag, 1999. LNCS No. 1666.

[CGS97] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Eurocrypt '97*, pages 103–118, 1997. LNCS No. 1233.

[CP92] D. Chaum and T. Pederson. Wallet databases with observers. In *Crypto '92*, LNCS No. 740, pages 89-105, 1992.

[CP02] C. Cachin, and J. A. Poritz Secure Intrusion-tolerant Replication on the Internet. In *Proc. Intl. Conference on Dependable Systems and Networks (DNS-2002)*, Washington DC, USA, IEEE, 2002. (see also *http://eprint.iacr.org/*)

[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.

[Des87] Yvo Desmedt. Society and group oriented cryptography: A new concept. *Crypto'87*, pages 120–127, 1987. LNCS No. 293.

[DF89] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Crypto '89*, pages 307–315, 1989. LNCS No. 435.

[Fel87] P. Feldman. A Practical Scheme for Non-Interactive Verifiable Secret Sharing. In *Proc. 28th FOCS*, pages 427–437. IEEE, 1987.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Crypto'86*, pages 186–194, 1986. LNCS No. 263.

[FMY99] Y. Frankel, P. D. MacKenzie, and M. Yung. Adaptively-secure distributed Public Key systems. In *Algorithms – ESA'99, 7th Annual European Symposium, Prague*, pages 4–27, 1999. LNCS No. 1643

[GJKR96] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *Information and Computation* 164, pp.54–84, 2001.

[GJKR99] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. The (in)security of distributed key generation in dlog-based cryptosystems. In *Eurocrypt '99*, pages 295–310, 1999. LNCS No. 1592.

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

[GRR98] Rosario Gennaro, Michael Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proc. 17th ACM Symp. on Principles of Distributed Computation*. ACM, 1998.

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

[HJJ+97] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *1997 ACM Conference on Computers and Communication Security*, 1997.

[Jar01] S. Jarecki. Efficient Threshold Cryptosystems. *MIT PhD Thesis*, June 2001, *theory.lcs.mit.edu/~cis/cis-theses.html*.

[JL00] Stanisław Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptosystems without erasures. In *Eurocrypt'00*, pages 221-242, 2000. LNCS. No. 1807

[LHL94] C.-H. Li, T. Hwang, and N.-Y. Lee. $(t, n)$ threshold signature schemes based on discrete logarithm. In *Eurocrypt '94*, pages 191–200, 1994. LNCS No. 950.

[LV01] A. K. Lenstra and E. R. Verheul Selecting Cryptographic Key Sizes. In *Journal of Cryptology*, vol. 14(4), 2001, pages 255-293.

[Ped91a] Torben Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Crypto '91*, pages 129–140. 1991.

[Ped91b] Torben Pedersen. A threshold cryptosystem without a trusted party. In *Eurocrypt '91*, pages 522–526, 1991. LNCS No. 547.

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

[PS96] D. Pointcheval, and J. Stern, Security Proofs for Signature Schemes. *Eurocrypt'96*, pages 387–398, 1996. LNCS No. 1070.

[Sha79] A. Shamir. How to Share a Secret. *CACM*, 22:612–613, 1979.

[Sch89] P. Schnorr. Efficient identification and signatures for smart cards. - *Crypto '89*, pages 235–251, 1989. LNCS No. 435

[Sho00] Victor Shoup. Practical threshold signatures. In *Eiurocrypt '00*, pages 207–220. Springer-Verlag, 2000.

[SG98] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *Eurocrypt '98*, pages 1–16, 1998. LNCS No. 1403.

[Wei00] Wei Dai. Benchmarks for the Crypto++ 4.0 library performance. Available at *http://www.eskimo.com/∼weidai/cryptlib.html*

# A Feldman's Verifiable Secret Sharing (VSS) Protocol

For completeness, we recall the Fel-VSS verifiable secret-sharing protocol of Feldman [Fel87] and we describe its security properties.[5] To share secret $x$ in $Z_q$, the dealer generates a random $t$-degree polynomial $f(z)$ over $Z_q$, s.t. $f(0) = x$, transmits to each player $P_i$ his *polynomial share* $\overline{x}_i = f(i)$ of secret $x$, and broadcasts values $X_k = g^{a_k} \bmod p$ for $k = 0, \ldots, t$, where $f(z) = \sum_k a_k z^k \bmod q$. We call the set of broadcasted values $\{X_0, \ldots, X_t\}$ *Feldman verification values*. Each player $P_i$ uses them to verify its share by checking that

$$g^{\overline{x}_i} = \prod_{k=0}^{t} (X_k)^{i^k} \bmod p \tag{2}$$

If a player $P_i$ holds a share that does not satisfy Eq. (2) then he broadcasts a *complaint* against the dealer. If more than $t$ players complain then the dealer is clearly faulty and is disqualified. Otherwise, the dealer reveals the share $\overline{x}_i$ matching Eq. (2) for each complaining player $P_i$. By convention, if the dealer is disqualified then $x = 0$.

We call a resulting distributed data structure, a set of polynomial shares $\overline{x}_i$ held by each player $P_i$ and the public set of Feldman verification values $\{X_0, \ldots, X_t\}$ a *Feldman secret sharing* of value $x$.

[5] The original protocol of [Fel87] is preceded by a stage in which the dealer permutes the indices of the players at random, which makes the protocol adaptively secure. Furthermore, the original presentation works for any homomorphic commitment function. Here we use a particular instantiation of Fel-VSS using exponentiation in a prime field as the commitment function.

Reconstruction of a secret $x$ from this secret sharing can proceed as follows: The dealer can publish $x$ and if $g^x = y$ then $x$ is accepted as the reconstructed secret. Otherwise, the other players have to reconstruct $x$ by having each player $P_i$ broadcast its share $\overline{x}_i$ of $x$. Everyone can verify each broadcasted share via equation 2 and the resulting secret is computed by polynomial interpolation in $Z_q$ of the shares that pass this verification (see e.g. [Sha79, Fel87]). This protocol is unconditionally robust if the adversarial threshold is $t < n/2$. Namely, after an execution of the Fel-VSS sharing protocol, there always exist a unique secret $x$ in $Z_q$ that will be output in the reconstruction protocol even if $t$ out of $n$ players behave in an arbitrarily malicious way [Fel87].

**Simulation of Feldman's VSS.** Moreover, the Fel-VSS protocol has the following secrecy property: It hides all information about the shared secret $x$ *except* for what is revealed by the public value $X_0 = g^x \bmod p$. This property is exhibited by a following simulation argument that shows that all other values seen by the adversary can be efficiently computed from $X_0$, and hence the protocol does not leak any more information on $x$ than $X_0$. (We will refer to this simulation procedure in the security proofs of threshold schemes which use Fel-VSS as a subprotocol.) Assume wlog that the adversary corrupts players $P_1, \ldots, P_t$ and that the dealer is some player $P_i$, $t < i \leq n$. The simulator, on input a random $X_0 \in G_q$, computes the rest of the adversary's view of Fel-VSS protocol by choosing random values $\overline{x}_i$, $i = 1, \ldots, t$, in $Z_q$, and computing the verification values $X_1, \ldots, X_t$ by "interpolation in the exponent" as follows. For each $i = 1, \ldots, t$, $X_i = X_0^{\lambda_{i0}} \prod_{j=1}^{t} (g^{\lambda_{ij}\overline{x}_j})$ where $\lambda_{i0}, \ldots, \lambda_{it}$ are constants s.t. coefficient $a_i = \sum_{j=0}^{t} \lambda_{ij} f(j)$. The produced view has the same probability distribution as an adversarial view of a run of Fel-VSS in which the dealer shares value $x = dlog_g X_0$.