J. Cryptology (2007) 20: 51–83
DOI: 10.1007/s00145-006-0347-3

Journal of
CRYPTOLOGY

Secure Distributed Key Generation for Discrete-Log Based Cryptosystems*

Rosario Gennaro
IBM T.J. Watson Research Center,
P.O. Box 704, Yorktown Heights, NY 10598, U.S.A.
rosario@watson.ibm.com

Stanisław Jarecki
School of Information and Computer Science,
University of California, Irvine, CA 92697-3425, U.S.A.
stasio@ics.uci.edu

Hugo Krawczyk and Tal Rabin
IBM T.J. Watson Research Center,
P.O. Box 704, Yorktown Heights, NY 10598, U.S.A.
{hugo,talr}@watson.ibm.com

Communicated by Matthew Franklin

Received 9 September 2003 and revised 2 August 2005
Online publication 19 May 2006

Abstract. A Distributed Key Generation (DKG) protocol is an essential component of threshold cryptosystems required to initialize the cryptosystem securely and generate its private and public keys. In the case of discrete-log-based (dlog-based) threshold signature schemes (ElGamal and its derivatives), the DKG protocol is further used in the distributed signature generation phase to generate one-time signature randomizers ($r = g^k$).

In this paper we show that a widely used dlog-based DKG protocol suggested by Pedersen does not guarantee a uniformly random distribution of generated keys: we describe an efficient active attacker controlling a small number of parties which successfully biases the values of the generated keys away from uniform. We then present a new DKG protocol for the setting of dlog-based cryptosystems which we prove to satisfy the security requirements from DKG protocols and, in particular, it ensures a uniform distribution of the generated keys. The new protocol can be used as a secure replacement for the many applications of Pedersen's protocol.

Motivated by the fact that the new DKG protocol incurs additional communication cost relative to Pedersen's original protocol, we investigate whether the latter can be used in specific applications which require relaxed security properties from the DKG

* The results presented in this paper appeared in preliminary form in [GJKR2] and [GJKR3]. Work by the third author was partially conducted while at the Department of Electrical Engineering, Technion, Haifa.

protocol. We answer this question affirmatively by showing that Pedersen's protocol suffices for the secure implementation of certain threshold cryptosystems whose security can be reduced to the hardness of the discrete logarithm problem. In particular, we show Pedersen's DKG to be sufficient for the construction of a threshold Schnorr signature scheme. Finally, we observe an interesting trade-off between security (reductions), computation, and communication that arises when comparing Pedersen's DKG protocol with ours.

Key words. Threshold cryptography, Distributed key generation, Discrete logarithm, VSS.

1. Introduction

Distributed key generation is a main component of threshold cryptosystems. It allows a set of n servers to generate jointly a pair of public and private keys according to the distribution defined by the underlying cryptosystem without ever having to compute, reconstruct, or store the secret key in any single location and without assuming any trusted party (dealer). While the public key is output in the clear, the private key is maintained as a (virtual) secret shared via a threshold scheme. In particular, no attacker can learn anything about the key as long as it does not break into a specified number of servers. This shared private key can be later used by a threshold cryptosystem, e.g., to compute signatures or decryptions, without ever being reconstructed in a single location. For discrete-log-based (dlog-based) schemes, distributed key generation amounts to generating a secret sharing of a random, uniformly distributed value x and making public the value $y = g^x$. We refer to such a protocol as DKG.

A DKG protocol may be run in the presence of a malicious adversary who corrupts a fraction (or threshold) of the parties and forces them to follow an arbitrary protocol of its choice. Informally, we say that a DKG protocol is secure if the output of the non-corrupted parties is correct (i.e. the shares held by the good parties define a unique uniformly distributed value x and the public value y satisfies $y = g^x$), and the adversary learns no information about the chosen secret x beyond what is learned from the public value y.

Pedersen's DKG *Protocol.* Solutions to the shared generation of private keys for dlog-based threshold cryptosystems [DF] have been known and used for a long time. Indeed, the first DKG scheme was proposed by Pedersen in [P1]. It then appeared, with various modifications, in several papers on threshold cryptography, e.g., [CMI], [Har], [LHL], [GJKR1], [HJJ$^+$], [PK], and [SG], and distributed cryptographic applications that rely on it, e.g., [CGS]. Moreover, a secure DKG protocol is an important building block in other distributed protocols for tasks different than the generation of keys. One example is the generation of the randomizers in dlog-based signature schemes (for example the r value in an (r, s) DSS signature pair as in [GJKR1]). Another example is the generation of the refreshing polynomial in proactive secret sharing and proactive signature schemes [HJKY], [HJJ$^+$], [FGMY].

The basic idea in Pedersen's DKG protocol [P1] (as well as in the subsequent variants) is to have n parallel executions of Feldman's verifiable secret sharing (VSS) protocol [Fel] in which each party P_i acts as a dealer of a random secret z_i that it picks. The secret

value x is taken to be the sum of the properly shared z_i's. Since Feldman's VSS has the additional property of revealing $y_i = g^{z_i}$, the public value y is the product of the y_i's that correspond to those properly shared z_i's.

The Insecurity of Pedersen's DKG. In this paper we show that, in spite of its use in many protocols, Pedersen's DKG cannot guarantee the correctness of the output distribution in the presence of an adversary. Specifically, we show a strategy for an adversary to manipulate the distribution of the resulting secret x to something quite different from the uniform distribution.[1] This flaw stresses a well-known basic principle for the design of cryptographic protocols, namely, that secure components can turn insecure when composed to generate new protocols. We note that this ability of the attacker to bias the output distribution represents a flaw in several aspects of the protocol's security. It clearly violates the basic correctness requirement about the output distribution of the protocol; but it also weakens the secrecy property of the solution. Indeed, the attacker acquires in this way some a priori knowledge on the secret which does not exist when the secret is chosen truly at random. Moreover, these attacks translate into flaws in the attempted proofs of these protocols; in particular, they invalidate the standard simulation arguments (à la zero-knowledge) as used to prove the secrecy of these protocols must fail. For example, to argue that a threshold DSS scheme [GJKR1] is as secure as the standard (i.e., centralized) DSS scheme, one needs to be able to *simulate*, for every given value of a public key y in the centralized scheme, a run of the threshold scheme which generates the *same* public key y. This requires that the threshold scheme will be able to generate keys with the exact same distribution as in the centralized setting. However, this is exactly what our attacker against Pedersen's DKG is able to prevent. However, see below for applications where Pedersen's DKG suffices to guarantee security.

A New DKG *Protocol with Guaranteed Uniform Output Distribution*. We present a new dlog DKG protocol that enjoys a full proof of security. We first present the formal requirements for a secure solution of the DKG problem, then present a particular DKG protocol and rigorously prove that it satisfies the security requirements. In particular, we show that the output distribution of private and public keys is as required, and prove the secrecy requirement from the protocol via a full simulation argument. Our solution is based on ideas similar to Pedersen's DKG (in particular, it also uses Feldman's VSS as a main component), but we are careful about designing an initial *commitment phase* where each party commits to its initial choice z_i in a way that prevents the attacker from later biasing the output distribution of the protocol. For this commitment phase we use another protocol of Pedersen, i.e., Pedersen's VSS protocol as presented in [P2].

Our new DKG protocol preserves most of the efficiency and simplicity of the original DKG solution from [P1]. There is however an increase in the round complexity: while Pedersen's protocol has one round of communication in the absence of faults, our new protocol requires two rounds of communication. Moreover, each communication round involves a reliable broadcast, which is a costly operation in a realistic setting like the

[1] In a related paper, Langford [Lan] showed that in a careless implementation of Pedersen's DKG the attacker could influence the choice of the secret key. She [Lan] attempts to solve the problem, yet the suggested solution is susceptible to our attacks.

Internet (see, e.g., [CP]). The new DKG protocol also requires about double the computation from each server. The computation and communication cost of the DKG protocol is particularly important in cases in which the protocol is repeatedly invoked as in the case of ElGamal-like signatures and proactive randomization.

Dealing with Non-Uniform Distribution on Keys. Motivated by the increase in performance cost incurred by our new DKG protocol relative to the original Pedersen's solution, we investigate the question of whether the latter can be used in applications with relaxed requirements on the DKG protocol. In particular, these must be applications where deviation from uniformity is not a show-stopper. Interestingly, we show this to be possible by first proving that Pedersen's DKG is suitable for applications in which the only security requirement is that the attacker cannot compute the generated secret key x (namely, the attacker cannot learn the discrete logarithm of the DKG's public output y). Then we show that certain class of threshold schemes can be built using a DKG protocol that satisfies this property. These are threshold variants of (centralized) schemes whose security can be proven to be equivalent to the hardness of computing discrete logarithms. Concretely, we show how to prove secure a threshold version of Schnorr's signature scheme [S1] 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 computing discrete logarithms (thus avoiding the simulation step between the centralized and threshold scheme as discussed above which cannot be carried using Pedersen's solution). Technically, this proof extends the random-oracle-based proof technique from [PS] to the threshold setting.

Which Protocol (and Proof Technique) to Prefer. As stated, there are some threshold schemes that can be proven secure even when they are instantiated with Pedersen's DKG protocol. Of course they can also be proven secure using our new DKG protocol. So which proof and protocol to prefer? A first observation deals with the use of the *random oracle* model. In this model the hash function used in the scheme is modeled as a *random function* in the security proof. This is clearly a mathematical abstraction that does not hold in reality, and indeed it has been shown that proofs in the random oracle model do not necessarily yield real security proofs [CGH]. Yet, a proof in the random oracle model can be considered a strong heuristic argument in favor of the security of the scheme. When using Pedersen's DKG our security proof works by direct reduction to the hardness of the discrete-log problem, however it does so in the random oracle model. On the other hand, when using our new DKG scheme the security proof works by reduction to the security of the centralized scheme, and dispenses with the random oracle model (note, however, that the random oracle model may be used in the proof of the centralized scheme).

The other main difference between the two protocols and their security proofs is *efficiency*. The security proof we exhibit for the threshold Schnorr signature scheme implemented with Pedersen's DKG has a significant drawback: the security reduction to the underlying hard problem is less efficient than the security reduction that exists for the centralized version of this scheme. On the other hand, if one implements threshold Schnorr signatures with our new DKG protocol, the reduction from the threshold scheme to the centralized one is tight. Thus, in the latter case, if one carries out the reduction all

the way to the discrete-log problem, the threshold scheme's reduction is as efficient as the centralized one. It is well known (see Section 6) that a less-efficient security reduction implies the need to choose a larger security parameter, with a consequent increase in the computational complexity of implementing the scheme. Thus our results suggest an interesting trade-off between the increase in round complexity incurred by our DKG protocol and the increase in computational complexity imposed by the security proof of Pedersen's DKG.

Other Applications. As we pointed out above, DKG protocols can be used to generate randomizers in threshold versions of ElGamal-like signature schemes. This kind of signature usually consists of a pair (r, s) where $r = g^k$ for a random value $k \in Z_q$; in this case the DKG protocol is applied for the distributed computation of the value r. Yet another application of DKG protocols arises in the setting of *proactive security*, for example when implementing proactive secret sharing [HJKY] or signature schemes [HJJ+]. These schemes were introduced to cope with "mobile adversaries" [OY] who may corrupt more than t servers during the lifetime of the secret. In this setting time is divided into *periods* and the security of the system is preserved as long as the attacker does not simultaneously control more than t parties in any single time period. The main tool for implementing such schemes is the use of proactive secret sharing in which secrets are "refreshed" at the beginning of each period such that the resultant new shares are independent of the shares in the previous period, except for the fact that they reconstruct the same secret. Technically, this refresh is achieved by the parties running a DKG protocol to create a random sharing of the value 0, and using these shares to randomize (via addition) the shares of the secret key x held by the parties in the previous period.

Organization. Section 2 outlines the adversarial model used throughout the paper and recalls some central secret sharing tools. Section 3 describes Pedersen's DKG protocol and our attack on it. In Section 4 we develop a formal definition of security for a (generic) DKG protocol, and present the new DKG protocol together with a rigorous proof of its security. Section 5 shows how the original Pedersen's DKG can be used to implement specific threshold signature schemes securely. We conclude with a detailed comparison of efficiency trade-offs between the new and Pedersen's DKG protocols in Section 6.

2. Preliminaries

2.1. *Communication and Adversary Models*

Communication Model

We assume that our computation model is composed of a set of *n parties* (or servers) $P_1, \ldots, P_n$ that can be modeled by polynomial-time randomized Turing machines. They are connected by a complete network of private (i.e., untappable and authenticated) point-to-point channels. In addition, the parties have access to a dedicated broadcast channel.

We assume a *partially synchronous* communication model: computation proceeds in synchronized rounds and 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 parties are equipped with synchronized clocks. Notice that, in this model, messages sent from the uncorrupted parties 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 parties 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.

We note that this communication model (and the resultant stronger advesarial model) is more realistic than the typically assumed *fully synchronous model* in which messages of a given round in the protocol are sent by all parties, and delivered to their intended recipients, *simultaneously.*

The Adversary

We assume that an adversary, $\mathcal{A}$, can corrupt up to t of the n parties in the network, for any value of $t < n/2$ (this is the best achievable threshold—or resilience—for solutions that provide both secrecy and robustness). We consider a malicious adversary that may cause corrupted parties to divert from the specified protocol in *any* way. We assume that the computational power of the adversary is adequately modeled by a probabilistic polynomial-time Turing machine. Our adversary is *static*, i.e., chooses the corrupted parties at the beginning of the protocol.[2] In addition, and as discussed above, the partially synchronous communication model assumed in this paper results in a stronger adversary; namely, one which chooses the messages sent by the corrupted parties at the end of each communication round after having seen the messages output by the uncorrupted players in that round. Yet, it is interesting to note that the attacks against known DKG protocols that we show in Section 3 hold also against the weaker adversarial model which assumes fully synchronized communication.

The Discrete-Log Assumption

A main computational assumption used throughout this paper is the infeasibility to compute discrete logarithms. In particular, we assume that the attacker against the protocols discussed here is unable to compute discrete logarithms modulo large primes. Specifically, we consider primes p for which there exists a large prime q dividing $p - 1$. The concept of "large" can be defined precisely by fixing a security parameter k and choosing p such that the lengths of both p and q grow polynomially with k. For any pair of prime numbers p and q as above we denote by G the subgroup of elements of order q in Z_p^*, and use g to denote a particular generator of G. Given an element $y \in G$ we can write $y = g^x \bmod p$ for $x \in [1..q]$; the integer x is called the discrete logarithm of y with respect to g. We use the notation $s \in_R S$ to mean "the element s is chosen with uniform probability from the set S."

[2] An extension of the results presented here to the setting of an *adaptive* adversary appears in [CGJ+]. See also Section 4.4.

Assumption 1 (Discrete-Log Assumption). *Let PRIMES*(k) *be the set of pairs* (p, q) *where* p *is a* $poly(k)$*-bit prime and* q *is a* k*-bit prime dividing* $p - 1$*. We assume that for every probabilistic polynomial-time Turing machine TM, for every polynomial* $P(\cdot)$*, and for sufficiently large* k*: if* $(p, q) \in_R PRIMES(k)$*,* g *is an element of order* q *in* Z_p^**, and* $x \in_R [1..q]$*, then* $\Pr(TM(p, q, g, g^x) = x) \leq 1/P(k)$.

The assumption can be strengthened by assuming that it holds for *every* prime pair (p, q), and not for a randomly chosen one.

2.2. *Secret Sharing Tools*

We recall three secret sharing protocols which we use throughout the paper.

Shamir's Secret Sharing

In Shamir's secret sharing protocol [S2] a dealer shares a secret $\sigma \in Z_q$ among the parties $P_1, \ldots, P_n$ in the following way. The dealer chooses at random a polynomial $f(z)$ over Z_q of degree t, such that $f(0) = \sigma$. He then secretly transmits to each party P_i a share $s_i = f(i) \bmod q$. It is clear that t or less parties have no information about the secret while $t + 1$ can easily reconstruct it by polynomial interpolation.

It is well known however that in the presence of a malicious adversary, Shamir's secret sharing protocol could fail. Indeed, it is possible for a dealer to share values which do not lie on a polynomial of degree t. Also dishonest parties may contribute incorrect shares at reconstruction time. *Verifiable secret sharing* (*VSS*) protocols [CGMA], described next, are intended to prevent this possibility.

Feldman's VSS

Feldman's VSS protocol [Fel] extends Shamir's secret sharing method in a way that allows the recipients of shares to verify that the shares they receive from the dealer are consistent (i.e., that any subset of $t + 1$ shares determines the same unique secret), and to filter out the incorrect shares submitted by the dishonest parties at reconstruction time. The protocol can tolerate up to $n/2$ malicious faults *including the dealer*. In the following we denote it with the name Feldman-VSS. Like in Shamir's scheme, the dealer generates a random t-degree polynomial $f(z) = \sum_k a_k z^k$ over Z_q, such that $f(0) = \sigma$, and transmits to each party P_i a share $s_i = f(i)$. The dealer also broadcasts verification values $A_k = g^{a_k} \bmod p$ for $k = 0, \ldots, t$. This will allow the parties to check that the values s_i really define a secret by checking that

$$g^{s_i} = \prod_{k=0}^{t} (A_k)^{i^k} \bmod p. \tag{1}$$

If a party P_i holds a share that does not satisfy (1) then he broadcasts a *complaint* against the dealer. The dealer reveals the share s_i matching (1) for each complaining party P_i. If any of the revealed shares fails this equation, the dealer is disqualified. This guarantees that either the dealer is disqualified or the honest parties hold at least $t + 1$ shares s_i matching (1). At reconstruction time, the same equation is used to detect any incorrect

shares submitted by the dishonest parties so that the remaining shares can be interpolated to reconstruct the shared secret σ.

Notice that the value of the secret is only computationally secure, e.g., the value $A_0 = g^{a_0} = g^{\sigma} \bmod p$ is leaked. However, it can be shown that an adversary that learns t or less shares cannot obtain any information on σ beyond what can be derived from g^{σ}. The proof of this fact uses a simulation argument which we sketch here. Given any t (or less) shares (known to the adversary) and $g^{\sigma} = A_0$, one can generate the distribution of the other public information in the protocol, i.e., values $A_1, \ldots, A_t$, as follows. Assume the known shares are $s_1, \ldots, s_t$. Thus we know $g^{f(i)} = g^{s_i}$, $i = 1, \ldots, t$, as well as $g^{f(0)} = g^{\sigma}$. This allows us to compute A_k for $k = 1, \ldots, t$ using the equation $A_k = g^{a_k} = \prod_{i=0}^{t} (g^{f(i)})^{\lambda_{ki}}$ where λ_{ki} are coefficients such that $a_k = \sum_{i=0}^{t} \lambda_{ki} f(i)$.[3]

Pedersen's VSS

Differently from Feldman-VSS, Pedersen-VSS provides perfect (information-theoretic) secrecy of the shared secret, namely, the view of the adversary is independent from the secret being shared. However, it assumes that the adversary (specifically the dealer) cannot solve the discrete-log problem. Indeed, a cheating dealer with such an ability could either successfully finish the sharing phase without a secret being properly shared and/or can prevent the subsequent reconstruction of the secret.

The scheme uses parameters p, q, and g as introduced before and an additional element h in the subgroup of Z_p^* generated by g. It is assumed that the adversary cannot compute $\log_g h$.

The dealer generates two random t-degree polynomials $f(z)$, $f'(z)$ over Z_q, such that $f(0) = \sigma$, the secret being shared. He transmits to each party P_i his share (s_i, s_i') where $s_i = f(i)$ and $s_i' = f'(i)$. The dealer also broadcasts values $C_k = g^{a_k} h^{b_k} \bmod p$ for $k = 0, \ldots, t$, where $f(z) = \sum_k a_k z^k$ and $f'(z) = \sum_k b_k z^k$. This will allow the parties to check that the values s_i, s_i' really define a secret by checking that

$$g^{s_i} h^{s_i'} = \prod_{k=0}^{t} (C_k)^{i^k} \bmod p. \tag{2}$$

If a party P_i holds a share that does not satisfy (2) then he *complains* against the dealer. The dealer reveals the share (s_i, s_i') matching (2) for each complaining party P_i. If any of the revealed shares fails this equation, the dealer is disqualified.

Some of the main properties of Pedersen-VSS are summarized in the next lemma and used in the analysis of our DKG solution in the next subsection.

Lemma 1 [P2]. *Under the Discrete-Log Assumption,* Pedersen-VSS *satisfies the following properties in the presence of a polynomially bounded adversary that corrupts at most t parties*:

1. *If the dealer is not disqualified during the protocol then all honest parties hold shares that interpolate to a unique polynomial of degree t. In particular, any* $t + 1$ *of these shares suffice to reconstruct* (*via interpolation*) *the secret s efficiently.*

[3] If $[f(0), \ldots, f(t)]^{\mathrm{T}} = A \cdot [a_0, \ldots, a_t]^{\mathrm{T}}$, where A is a $(t+1)$ by $(t+1)$ matrix with i^k in row $i = 0, \ldots, t$ and column $k = 0, \ldots, t$, then $[a_0, \ldots, a_t]^{\mathrm{T}} = A^{-1} \cdot [f(0), \ldots, f(t)]^T$ and λ_{ki}'s are entries of A^{-1}.

2. *The protocol produces information* (*the public values* C_k *and private values* s'_i) *that can be used at reconstruction time to test for the correctness of each share*; *thus*, *reconstruction is possible*, *even in the presence of malicious parties*, *from any subset of shares containing at least* $t + 1$ *correct shares*.
3. *The view of the adversary is independent of the value of the secret* s, *and therefore the secrecy of* s *is unconditional*.

3. The Insecurity of Pedersen's DKG Protocol

We recall that a DKG protocol is run by the parties on an empty input and it should produce as a public output, a random value y uniformly distributed in G. Each party P_i should also hold as a private output a value x_i which is a share of a t-out-of-n sharing of x, the discrete log in base g of y. We give a formal definition of DKG security in Section 4, but for now the above informal DKG requirements will suffice to show the problem with previous DKG proposals.

Based on Feldman's VSS protocol, Pedersen [P1] proposed the first DKG protocol. It requires the execution of n parallel invocations of Feldman-VSS as follows. Each party P_i selects a random secret $z_i \in Z_q$ and shares it among the n parties using Feldman-VSS. This defines the set *QUAL* of parties that shared their secrets properly. The random secret x is set to be the sum of the properly shared secrets and each party can compute his share of x by locally summing up the shares he received. The value y can be computed as the product of the public values $y_i = g^{z_i} \bmod p$ generated by the proper executions of the Feldman-VSS protocols. Similarly, the verification values $A_1, \ldots, A_t$ necessary for robust reconstruction of x in Feldman-VSS, can be computed as products of the corresponding verification values generated by each properly executed VSS protocol.

In Fig. 1 we present a simplified version of the protocol proposed in [P1], which we call JF-DKG (for "Joint Feldman DKG"). By concentrating on the core of the protocol we are able to emphasize the central weakness in its design. We then show how this crucial flaw applies also to several variants of this core protocol (including the full protocol from [P1] and other modifications [HJKY], [HJJ$^+$]).

An Attack Against JF-DKG

The JF-DKG protocol enables an adversary to influence the distribution of the result to a non-uniform distribution. It can be seen, from the above description of the protocol (Fig. 1), that the determining factor for what the value x will be, is the definition of the set *QUAL*. The attack utilizes the fact that the decision whether a party is in *QUAL* or not, even given the fully synchronous communication model, occurs after the adversary has seen the values y_i of all parties. The values y_i are made public in Step 1 and the disqualification of parties occurs in Step 3. Using this timing discrepancy, the attacker can affect the distribution of the pair (x, y).

More specifically the attack works as follows. Assume the adversary wants to bias the distribution towards keys y whose last bit is 0. He corrupts two parties, say P_1 and P_2. In Step 1, P_1, as a dealer, follows the protocol. At the end of Step 1 the adversary computes $\alpha = \prod_{i=1}^{n} y_i$ and $\beta = \prod_{i=2}^{n} y_i$. If α ends with 0 then the adversary will do nothing, and P_1 will not be disqualified. Otherwise, α ends with 1, the adversary forces the disqualification of P_1. This is achieved by having P_2 complain against P_1 in Step 2,

Protocol JF-DKG

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

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

   P_i broadcasts $A_{ik} = g^{a_{ik}} \bmod p$ for $k = 0, \ldots, t$. Denote a_{i0} by z_i and A_{i0} by y_i. Each P_i computes the shares $s_{ij} = f_i(j) \bmod q$ for $j = 1, \ldots, n$ and sends s_{ij} secretly to party P_j.
2. Each P_j verifies the shares he received from the other parties by checking for $i = 1, \ldots, n$:

$$g^{s_{ij}} = \prod_{k=0}^{t} (A_{ik})^{j^k} \bmod p. \tag{3}$$

   If the check fails for an index i, P_j broadcasts a *complaint* against P_i.
3. Party P_i (as a dealer) reveals the share s_{ij} matching (3) for each complaining party P_j. If any of the revealed shares fails this equation, P_i is disqualified. We define the set $QUAL$ to be the set of non-disqualified parties.
4. The public value y is computed as $y = \prod_{i \in QUAL} y_i \bmod p$. The public verification values are computed as $A_k = \prod_{i \in QUAL} A_{ik} \bmod p$ for $k = 1, \ldots, t$. Each party P_j sets his share of the secret as $x_j = \sum_{i \in QUAL} s_{ij} \bmod q$. The secret shared value x itself is not computed by any party, but it is equal to $x = \sum_{i \in QUAL} z_i \bmod q$.

Fig. 1. Pedersen's solution for distributed generation of secret keys.

and P_1 broadcasting an inconsistent share. This action sets the public value y to β which ends with 0 with probability $1/2$. Thus effectively the attacker has forced strings ending in 0 to appear with probability $3/4$ rather than $1/2$.

Note: It is worth remarking that the above adversarial actions are possible even if we limit the attacker to work on a fully synchronized communication model where all messages corresponding to a given communication round are generated and delivered from and to all parties simultaneously. In particular, the added power to the adversary provided by our partially synchronous model (see Section 2.1) is not necessary for these attacks to succeed.

Why the Simulation Fails

An attempt to prove this protocol secure would use a simulation argument. Following is an explanation of why such a simulator would fail. Consider a simulator $\mathcal{S}$ which receives the value y and needs to "hit" this value. That is, $\mathcal{S}$ needs to generate a transcript which is indistinguishable from an actual run of the protocol that outputs y as the public key, and where the adversary controls up to t parties, say $P_1, \ldots, P_t$. The simulator has enough information to compute the values $z_1, \ldots, z_t$ that the adversary has shared in Step 1. Now $\mathcal{S}$ needs to commit itself to the values shared by the good parties. However, the attack described in the paragraph above can be easily extended to a strategy that allows the adversary to decide in Steps 2–3 on the set Q of faulty parties whose values will be considered in the final computation (i.e., $QUAL = Q \cup \{t+1, \ldots, n\}$). Consequently, in Step 1, the simulator $\mathcal{S}$ does not know how to pick the good parties' values $y_{t+1}, \ldots, y_n$ so that $(\prod_{i \in Q} y_i) \cdot (y_{t+1} \cdot \cdots \cdot y_n) = y \bmod p$, as $\mathcal{S}$ still does not know the set Q. Since the number of possible sets Q that the adversary can choose is exponential in t, then $\mathcal{S}$ has no effective strategy to simulate this computation in polynomial time.

Other Insecure Variants of the JF-DKG *Protocol*

The many variants and extensions of the JF-DKG protocol which have appeared in the literature are also insecure. They suffer from the same drawback of JF-DKG discussed above. The variants include: signatures on shares, commitments to y_i, committing encryption on broadcast channel, committing encryption with reconstruction, and "stop, kill, and rewind." A discussion of these variants and their flaws is given in the Appendix.

What Can We Prove about JF-DKG?

The results in this section show that JF-DKG cannot be used as a generic secure DKG protocol, in particular one in which the generated key is to be chosen with uniform distribution. Yet, one can wonder whether there are applications of DKG for which the security requirements can be relaxed to allow the use of the JF-DKG protocol in a secure way. Interestingly, we show this to be the case for some natural uses of DKG. Specifically, we show in Section 5 how to apply JF-DKG in order to achieve a provably secure threshold Schnorr signature scheme. As we will see, for the security of this application the full strength of DKG is not required but only that the attacker cannot compute the discrete logarithm of the resultant (public) key y, a property that we show to hold for JF-DKG.

4. The New DKG Protocol

As shown in the previous section, the JF-DKG protocol cannot be considered a secure DKG protocol in general since it cannot guarantee that the output y from the protocol (as well as the virtually shared secret x) be distributed according to the uniform distribution as required by dlog-based cryptosystems. Moreover, beyond this generic weakness of the protocol, there are specific applications of DKG where the lack of the uniformity guarantee spoils the security of the scheme or, at least, invalidates the known proofs of security. To illustrate the latter point, we examine the use of the DKG protocol in the context of the threshold DSS signature scheme described in [GJKR1]. Here DKG is used for the generation of the (shared) private signature key x and the corresponding public key $y = g^x$, as well as for the generation of pairs $(k, r = g^k)$ which are part of the computation of each individual signature.

The proof of security of the threshold scheme [GJKR1] is carried by reducing its security to the security of the regular (centralized) DSS scheme.[4] Namely, given an attacker against the threshold signatures one constructs an attacker against the centralized scheme. Note that for this strategy to succeed one needs to be able to create a virtual (simulated) threshold scenario in which the generated public key value y is the *same* as the *given* public key y from the centralized scheme. However, if one uses the JF-DKG protocol in the implementation of the threshold scheme this "hitting" of a specific y cannot succeed: this is something our attacker against JF-DKG from Section 3 can *always* prevent. A similar problem arises in showing how to transform successful forgeries in the

[4] We stress that this is the best possible security guarantee in this case since centralized DSS signatures are *secure by "self-assumption"* rather than by reduction to an established hard problem such as computing discrete logarithms.

threshold case into forgeries against the centralized scheme. In this case it is the values of $r = g^k$ provided by the signatures in the centralized scheme that need to be "hit" in the threshold simulation. Yet, once again, the attacker against JF-DKG can always succeed in spoiling such a simulation.

Note that the above is not a cryptanalytical argument showing the insecurity of threshold DSS when implemented with JF-DKG but rather a strong argument against the viability of known proof techniques to succeed in proving the security of such schemes. In other words, either threshold DSS with JF-DKG is truly insecure, or new proof techniques will be required to prove the contrary. Fortunately, as we show next provably secure threshold DSS schemes (as well as others) can be implemented by using our New-DKG protocol. On the other hand, we show in Section 5 that in some applications the security requirements from the DKG protocol can be relaxed enough to allow for the secure use of JF-DKG.

4.1. *Requirements for a Secure* DKG *Protocol*

In order to be able to provide a rigorous proof of security for our proposed DKG protocol we first develop a formal definition of security for such protocols. As we mentioned in the Introduction, distributed generation of keys in a dlog-based scheme amounts to generating a secret sharing of a random, uniformly distributed value $x \in Z_q$ and making public the value $y = g^x \bmod p$ (where p, q, g are as defined above). Specifically, in a dlog-based scheme the distributed protocol DKG performed by n parties $P_1, \ldots, P_n$ generates private outputs $x_1, \ldots, x_n$, called *the shares*, and a public output y. The protocol is called *t-secure* (or secure with threshold t) if in the presence of an attacker that corrupts at most t parties the following requirements for correctness and secrecy are satisfied:

Correctness.

- (C1) All subsets of $t + 1$ shares provided by honest parties define the same unique secret key x.
- (C2) All honest parties have the same value of public key $y = g^x \bmod p$, where x is the unique secret guaranteed by (C1).
- (C3) x is uniformly distributed in Z_q (and hence y is uniformly distributed in the subgroup generated by g).

Secrecy. No information on x can be learned by the adversary except for what is implied by the value $y = g^x \bmod p$.

More formally, we state this condition in terms of simulatability: for every (probabilistic polynomial-time) adversary $\mathcal{A}$ that corrupts up to t parties, there exists a (probabilistic polynomial-time) simulator $\mathcal{S}$, such that on input an element y in the subgroup of Z_p^* generated by g, produces an output distribution which is polynomially indistinguishable from $\mathcal{A}$'s view of a run of the DKG protocol that ends with y as its public key output.

The above is a minimal set of requirements needed in all known applications of such a protocol. In many applications a stronger version of (C1) is desirable, which reflects two additional aspects: (1) It requires the existence of an *efficient procedure* to build the secret x out of $t + 1$ shares; and (2) it requires this procedure to be *robust*, i.e.,

the reconstruction of x should be possible also in the presence of malicious parties that try to foil the computation. We note that these added properties are useful not only in applications that require explicit reconstruction of the secret, but also in applications (such as threshold cryptosystems) that use the secret x in a distributed manner (without ever reconstructing it) to compute some cryptographic function, e.g., a signature. Thus, we formulate (C1′) as follows:

(C1′) There is an efficient procedure that on input the n shares submitted by the parties and the public information produced by the DKG protocol, outputs the unique value x, even if up to t shares are submitted by faulty parties.

Remark. The above conditions (C1), (C2), and (C1′) can of course be relaxed to allow for a negligible (in the security parameter) probability of error. Similarly condition (C2) can be relaxed to allow a distribution for x which has negligible statistical distance from the uniform distribution. In the following when we say that a DKG protocol is secure we mean that it satisfies these relaxed versions of the above conditions.

4.2. *The New Scheme*

Our solution enjoys the same flavor and simplicity as the JF-DKG protocol presented in Fig. 1, i.e., each party shares a random value and the random secret is generated by summing up these values. However, we use a different sharing for the random values z_i, one that is information-theoretically secure, so that no information is revealed about the z_i's. This will prevent the adversary from modifying the set of qualified parties based on the values of the z_i's. We then need a method to extract the public key y from these sharings.

In more detail, we start by running a commitment stage where each party P_i commits to a t-degree polynomial $f_i(z)$ whose constant coefficient is the random value, z_i, contributed by P_i to the jointly generated secret x. We require the following properties from this commitment stage: First, the attacker cannot force a commitment by a (corrupted) party P_j to depend on the commitment(s) of any set of honest parties. Second, for any party P_i that is not disqualified during this stage, there is a unique polynomial f_i committed to by P_i and this polynomial is recoverable by the honest parties (this may be needed if party P_i misbehaves at a later stage of the protocol). Finally, for each honest party P_i and non-disqualified party P_j, P_i holds the value $f_i(j)$ at the end of the commitment stage.

To realize the above commitment stage we use the information-theoretic VSS protocol due to Pedersen [P2] and which we denote by Pedersen-VSS (see Section 2.2). We show that at the end of the commitment stage the value of the secret x is determined and no later misbehavior by any party can change it (indeed, if a non-disqualified party misbehaves later in the protocol his value z_i is publicly reconstructed by the honest parties). Most importantly, this guarantees that no bias in the output x or y of the protocol is possible, and it allows us to present a full proof of security based on a careful simulation argument.

After the value x is fixed we enable the parties to compute $g^x \bmod p$ efficiently and securely. This is done by having party P_i run a Feldman-VSS with the same polynomial $f_i(z)$ he committed to before. Thus, P_i does not distribute shares in this step, but only publishes the public verification values of Feldman-VSS which are validated by party

P_j using the point $f_i(j)$ he received before. Feldman-VSS reveals g^{z_i} and by the fact that we are using the same polynomial it assures us that it is the z_i committed to by P_i at the beginning (this way the adversary cannot change his contribution at this point). Notice that if a party P_i misbehaves at this point (for example by refusing to carry on the Feldman-VSS or carrying it out incorrectly), the honest parties can recover the polynomial f_i. This way, P_i's contribution to the secret key x will still be included (if we did not include it, we would allow the adversary to bias the distribution of x).

4.3. *Secure* DKG *Protocol*

Our secure solution to the distributed generation of keys follows the above ideas and is presented in detail in Fig. 2. We denote this protocol as New-DKG. The security properties of this solution are stated in the next theorem.

Theorem 1. *Under the Discrete-Log Assumption, Protocol* New-DKG *from Fig.* 2 *is a secure protocol for distributed key generation in dlog-based cryptosystems, namely, it satisfies the correctness and secrecy requirements of Section* 4.1 *with threshold* t*, for any* $t < n/2$.

Proof of Correctness. We first note that all honest parties in the protocol compute the same set *QUAL* since the determination of which parties are to be disqualified depends on public broadcast information which is known to all (honest) parties.

(C1) At the end of Step 2 of the protocol it holds that if $i \in QUAL$ then party P_i has successfully performed the dealing of z_i under Pedersen-VSS. From part 1 of Lemma 1 we know that all honest parties hold shares (s_{ij}) which interpolate to a unique polynomial with constant coefficient equal to z_i. Thus, for any set $\mathcal{R}$ of $t+1$ correct shares, $z_i = \sum_{j\in\mathcal{R}} \gamma_j \cdot s_{ij} \bmod q$ where γ_j are appropriate Lagrange interpolation coefficients for the set $\mathcal{R}$. Since each honest party P_j computes its share x_j of x as $x_j = \sum_{i\in QUAL} s_{ij}$, then we have that for the set of shares $\mathcal{R}$,

$$x = \sum_{i\in QUAL} z_i = \sum_{i\in QUAL}\left(\sum_{j\in\mathcal{R}} \gamma_j \cdot s_{ij}\right) = \sum_{j\in\mathcal{R}} \gamma_j \cdot \left(\sum_{i\in QUAL} s_{ij}\right) = \sum_{j\in\mathcal{R}} \gamma_j x_j.$$

Since this holds for *any* set of $t+1$ correct shares then x is uniquely defined.

(C1′) The above argument in (C1) shows that the secret x can be efficiently reconstructed, via interpolation, out of any $t+1$ correct shares. We need to show that we can tell apart correct shares from incorrect ones. For this we show that for each share x_j, the value g^{x_j} can be computed from publicly available information broadcast in Step 4(a):

$$g^{x_j} = g^{\sum_{i\in QUAL} s_{ij}} = \prod_{i\in QUAL} g^{s_{ij}} = \prod_{i\in QUAL}\prod_{k=0}^{t} (A_{ik})^{j^k} \bmod p,$$

where the last equality follows from (5). Thus the publicly available value g^{x_j} makes it possible to verify the correctness of share x_j at reconstruction time.

Protocol New-DKG

Generating x:

1. Each party P_i performs a Pedersen-VSS of a random value z_i as a dealer:
   (a) P_i chooses two random polynomials $f_i(z)$, $f_i'(z)$ over Z_q of degree t:

$$f_i(z) = a_{i0} + a_{i1}z + \cdots + a_{it}z^t, \qquad f_i'(z) = b_{i0} + b_{i1}z + \cdots + b_{it}z^t.$$

   Let $z_i = a_{i0} = f_i(0)$. P_i broadcasts $C_{ik} = g^{a_{ik}}h^{b_{ik}} \bmod p$ for $k = 0, \ldots, t$. P_i computes the shares $s_{ij} = f_i(j), s'_{ij} = f_i'(j) \bmod q$ for $j = 1, \ldots, n$ and sends s_{ij}, s'_{ij} to party P_j.
   (b) Each party P_j verifies the shares he received from the other parties. For each $i = 1, \ldots, n$, P_j checks if

$$g^{s_{ij}}h^{s'_{ij}} = \prod_{k=0}^{t}(C_{ik})^{j^k} \bmod p. \tag{4}$$

   If the check fails for an index i, P_j broadcasts a *complaint* against P_i.
   (c) Each party P_i who, as a dealer, received a complaint from party P_j broadcasts the values s_{ij}, s'_{ij} that satisfy (4).
   (d) Each party marks as *disqualified* any party that either
   - received more than t complaints in Step 1(b), or
   - answered a complaint in Step 1(c) with values that falsify (4).
2. Each party then builds the set of non-disqualified parties *QUAL*. (We show in the analysis that all honest parties build the same set *QUAL* and hence, for simplicity, we denote it with a unique global name.)
3. The distributed secret value x is not explicitly computed by any party, but it equals $x = \sum_{i\in QUAL} z_i \bmod q$. Each party P_i sets his share of the secret as $x_i = \sum_{j\in QUAL} s_{ji} \bmod q$ and the value $x'_i = \sum_{j\in QUAL} s'_{ji} \bmod q$.

Extracting $y = g^x \bmod p$:

4. Each party $i \in QUAL$ exposes $y_i = g^{z_i} \bmod p$ via Feldman-VSS:
   (a) Each party P_i, $i \in QUAL$, broadcasts $A_{ik} = g^{a_{ik}} \bmod p$ for $k = 0, \ldots, t$.
   (b) Each party P_j verifies the values broadcast by the other parties in *QUAL*, namely, for each $i \in QUAL$, P_j checks if

$$g^{s_{ij}} = \prod_{k=0}^{t}(A_{ik})^{j^k} \bmod p. \tag{5}$$

   If the check fails for an index i, P_j *complains* against P_i by broadcasting the values s_{ij}, s'_{ij} that satisfy (4) but do not satisfy (5).
   (c) For parties P_i who receive at least one valid complaint, i.e., values which satisfy (4) and not (5), the other parties run the reconstruction phase of Pedersen-VSS to compute z_i, $f_i(z)$, A_{ik} for $k = 0, \ldots, t$ in the clear. For all parties in *QUAL*, set $y_i = A_{i0} = g^{z_i} \bmod p$. Compute $y = \prod_{i\in QUAL} y_i \bmod p$.

Fig. 2. Secure distributed key generation in dlog-based systems.

(C2) The value y is computed (by the honest parties) as $y = \prod_{i\in QUAL} y_i \bmod p$, where the values of y_i are derived from information broadcast in the protocol and thus known to all honest parties. We need to show that indeed $y = g^x$ where $x = \sum_{i\in QUAL} z_i$. We will show that for $i \in QUAL$, $y_i = g^{z_i}$, and then $y = \prod_{i\in QUAL} y_i = \prod_{i\in QUAL} g^{z_i} = g^{\Sigma_{i\in QUAL} z_i} = g^x$. For parties $i \in QUAL$ against whom a valid complaint has been issued in Step 4(b), value z_i is publicly reconstructed and y_i set to $g^{z_i} \bmod p$ (the correct

reconstruction of z_i is guaranteed by Lemma 1 (part 2)). Now we need to show that for P_i, $i \in QUAL$, against whom a valid complaint has not been issued, the value y_i is set to A_{i0}. Values A_{ik}, $k = 0, \ldots, t$, broadcast by party P_i in Step 4(a) define a t-degree polynomial $\hat{f}_i(z)$ in Z_q. Since we assume that no valid complaint was issued against P_i then (5) is satisfied for all honest parties, and thus $\hat{f}_i(z)$ and $f_i(z)$ have at least $t+1$ points in common, given by the shares s_{ij} held by the uncorrupted parties P_j. Hence they are equal, and in particular $A_{i0} = g^{f_i(0)} = g^{z_i}$.

(C3) The secret x is defined as $x = \sum_{i \in QUAL} z_i$. Note that as long as there is one value z_i in this sum that is chosen at random and independently from other values in the sum, we are guaranteed to have uniform distribution of x. Also note that the secret x and the components z_i in the sum are already determined at the end of Step 2 of New-DKG (since neither the values z_i nor the set $QUAL$ change later). Let P_i be a non-corrupted party; in particular, $i \in QUAL$. At the end of Step 1 of the protocol z_i exists only as a value dealt by P_i using Pedersen-VSS. By virtue of part 3 of Lemma 1 the view (and thus actions) of the adversary are independent of this value z_i and hence the secret x is uniformly distributed (as z_i is).

Proof of Secrecy. We provide a simulator $\mathcal{S}$ for the New-DKG protocol in Fig. 3. In the description and analysis of the simulator we assume, without loss of generality, that the adversary compromises parties $P_1, \ldots, P_{t'}$, where $t' \leq t$. We denote the indices of the parties controlled by the adversary by $\mathcal{B} = \{1, \ldots, t'\}$, and the indices of the parties controlled by the simulator by $\mathcal{G} = \{t'+1, \ldots, n\}$.

An informal description of the simulator is as follows. $\mathcal{S}$ will run on behalf of the good parties, the first part of the protocol (Steps 1–3) following exactly the instructions. For $n - t' - 1$ of the good parties he will also follow Step 4. However, for one of the good parties (say P_n) he will have to change the broadcasted value to "hit" the desired public key y. At this point $\mathcal{S}$ knows all the values A_{i0} for $i \in QUAL$, since if P_i is good he chose A_{i0}, otherwise he knows enough points on the polynomials shared by the bad party. Thus he will choose A_{n0} so that $\prod_{i \in QUAL} A_{i0} = y$. Now he has to choose the other values A_{nk}'s so that the shares held by the bad parties will match them. He can do this via a simple interpolation in the exponent. Again details are in Fig. 3.

We now proceed to show that the view of the adversary $\mathcal{A}$ that interacts with $\mathcal{S}$ on input y is the same as the view of $\mathcal{A}$ that interacts with the honest parties in a regular run of the protocol that outputs the given y as the public key.

In a regular run of protocol New-DKG, $\mathcal{A}$ sees the following probability distribution of data produced by the uncorrupted parties:

- Values $f_i(j)$, $f'_i(j)$, $i \in \mathcal{G}$, $j \in \mathcal{B}$, uniformly chosen in Z_q (and denoted as s_{ij}, s'_{ij}, resp.).
- Values C_{ik}, A_{ik}, $i \in \mathcal{G}$, $k = 0, \ldots, t$, that correspond to (exponents of) coefficients of randomly chosen polynomials and for which (4) and (5) are satisfied for all $j \in \mathcal{B}$.

Since here we are interested in runs of New-DKG that end with the value y as the public key output of the protocol, we note that the above distribution of values is induced by the choice (of the good parties) of polynomials $f_i(z)$, $f'_i(z)$, $i \in \mathcal{G}$, uniformly distributed in

Algorithm of Simulator $\mathcal{S}$

We denote by $\mathcal{B}$ the set of parties controlled by the adversary, and by $\mathcal{G}$ the set of honest parties (run by the simulator). Without loss of generality, $\mathcal{B} = \{1, \ldots, t'\}$ and $\mathcal{G} = \{t'+1, \ldots, n\}$, $t' \leq t$.

Input: public key y

1. Perform Steps 1–3 on behalf of the uncorrupted parties $P_{t'+1}, \ldots, P_n$ exactly as in protocol New-DKG. This includes receiving and processing the information sent privately and publicly from corrupted parties to honest ones. At the end of Step 2 the following hold:
   - The set $QUAL$ is well-defined. Note that $\mathcal{G} \subseteq QUAL$ and that polynomials $f_i(z)$, $f'_i(z)$ for $i \in \mathcal{G}$ are chosen at random.
   - The adversary's view consists of polynomials $f_i(z)$, $f'_i(z)$ for $i \in \mathcal{B}$, the shares $(s_{ij}, s'_{ij}) = (f_i(j), f'_i(j))$ for $i \in QUAL$, $j \in \mathcal{B}$, and all the public values C_{ik} for $i \in QUAL$, $k = 0, \ldots, t$.
   - $\mathcal{S}$ knows all polynomials $f_i(z)$, $f'_i(z)$ for $i \in QUAL$ (note that for $i \in QUAL \cap \mathcal{B}$ the honest parties, and hence $\mathcal{S}$, receive enough consistent shares from the adversary that allow $\mathcal{S}$ to compute all these parties' polynomials). In particular, $\mathcal{S}$ knows all the shares s_{ij}, s'_{ij}, the coefficients a_{ik}, b_{ik}, and the public values C_{ik}.
2. Perform the following computations:
   - Compute $A_{ik} = g^{a_{ik}}$ for $i \in QUAL \backslash \{n\}$, $k = 0, \ldots, t$.
   - Set $A^*_{n0} = y \cdot \prod_{i \in (QUAL \backslash \{n\})} (A_{i0})^{-1} \bmod p$.
   - Assign $s^*_{nj} = s_{nj} = f_n(j)$ for $j = 1, \ldots, t$.
   - Compute $A^*_{nk} = (A^*_{n0})^{\lambda_{k0}} \cdot \prod_{i=1}^{t} (g^{s^*_{ni}})^{\lambda_{ki}}$ for $k = 1, \ldots, t$, where λ_{ki}'s are the Lagrange interpolation coefficients.

   (a) Broadcast A_{ik} for $i \in \mathcal{G} \backslash \{n\}$, and A^*_{nk} for $k = 0, \ldots, t$.

   (b) Perform for each uncorrupted party the verifications of (5) on the values A_{ik}, $i \in \mathcal{B}$, broadcast by the parties controlled by the adversary. If the verification fails for some $i \in \mathcal{B}$, $j \in \mathcal{G}$, broadcast a complaint (s_{ij}, s'_{ij}). (Notice that the corrupted parties can publish a valid complaint only against one another.)

   (c) Perform Step 4(c) of the protocol on behalf of the uncorrupted parties, i.e., perform reconstruction phase of Pedersen-VSS to compute z_i and y_i in the clear for every P_i against whom a valid accusation was broadcast in the previous step.

Fig. 3. Simulator for the shared key generation protocol New-DKG.

the family of t-degree polynomials over Z_q subject to the condition that

$$\prod_{i \in QUAL} A_{i0} = y \bmod p. \tag{6}$$

In other words, this distribution is characterized by the choice of polynomials $f_i(z)$, $f'_i(z)$ for $i \in (\mathcal{G} \backslash \{n\})$ and $f'_n(z)$ as random independent t-degree polynomials over Z_q, and of $f_n(z)$ as a uniformly chosen polynomial from the family of t-degree polynomials over Z_q that satisfy the constraint $f_n(0) = \log_g(y) - \sum_{i \in (QUAL \backslash \{n\})} f_i(0) \bmod q$. (This last constraint is necessary and sufficient to guarantee (6).) Note that, using the notation of values computed by $\mathcal{S}$ in Step 2 of the simulation, the last constraint can be denoted as $f_n(0) = \log_g(A^*_{n0})$.

We show that the simulator $\mathcal{S}$ outputs a probability distribution which is *identical* to the above distribution. First note that the above distribution depends on the set $QUAL$ defined at the end of Step 2 of the protocol. Since all the simulator's actions in Step

1 of the simulator are identical to the actions of honest parties interacting with $\mathcal{A}$ in a real run of the protocol, we are assured that the set *QUAL* is defined at the end of this simulation step identically to its value in the real protocol. We now describe the output distribution of $\mathcal{S}$ in terms of t-degree polynomials f_i^* and $f_i'^*$ corresponding to the choices of the simulator when simulating the actions of the honest parties and defined as follows: For $i \in \mathcal{G}\backslash\{n\}$, set f_i^* to f_i and $f_i'^*$ to f_i'. For $i = n$, define f_n^* via the values[5] $f_n^*(0) = \log_g(A_{n0}^*)$ and $f_n^*(j) = s_{nj}^* = f_n(j)$, $j = 1, \ldots, t$. Finally, the polynomial $f_n'^*$ is defined via the relation $f_n^*(z) + d \cdot f_n'^*(z) = f_n(z) + d \cdot f_n'(z) \bmod q$, where $d = \log_g(h)$. It can be seen by this definition that the values of these polynomials evaluated at the points $j \in \mathcal{B}$ coincide with the values $f_i(j)$, $f_i'(j)$ which are seen by the corrupted parties in Step 1 of the protocol. Also, the coefficients of these polynomials agree with the exponentials C_{ik} published by the simulated honest parties in Step 1 of the protocol (i.e., $C_{ik} = g^{a_{ik}^*} h^{b_{ik}^*}$ where a_{ik}^* and b_{ik}^* are the coefficients of polynomials $f_i^*(z)$, $f_i'^*(z)$, respectively, for $i \in \mathcal{G}$), as well as with the exponentials A_{ik}, $i \in \mathcal{G}\backslash\{n\}$, and A_{nk}^* published by the simulator in Step 2(a) on behalf of the honest parties (i.e., $A_{ik} = g^{a_{ik}^*}$, $i \in \mathcal{G}\backslash\{n\}$, and $A_{nk}^* = g^{a_{nk}^*}$, $k = 0, \ldots, t$) corresponding to the parties' values in Step 4(a) of the protocol. Thus, these values pass the verifications of (4) and (5) as in the real protocol.

It remains to be shown that polynomials f_i^* and $f_i'^*$ belong to the right distribution. Indeed, for $i \in \mathcal{G}\backslash\{n\}$ this is immediate since they are defined identically to f_i and f_i' which are chosen according to the uniform distribution. For f_n^* we see that this polynomial evaluates in points $j = 1, \ldots, t$ to random values (s_{nj}) while at 0 it evaluates $\log_g(A_{n0}^*)$ as required to satisfy (6). Finally, polynomial $f_n'^*$ is defined (see above) as $f_n'^*(z) = d^{-1} \cdot (f_n(z) - f_n^*(z)) + f_n'(z)$ and since $f_n'(z)$ is chosen in Step 1 as a random and independent polynomial then so is $f_n'^*(z)$.

4.4. *Remarks*

Efficiency. We point out that our secure New-DKG protocol does not lose much in efficiency with respect to the previously known insecure JF-DKG protocol. Instead of Feldman-VSS, each party performs Pedersen-VSS (Steps 1–3), which takes the same number of rounds and demands at most twice more local computation. The extraction of the public key in Step 4 adds only two rounds (one if no party is dishonest) to the whole protocol. We point out that all the long modular exponentiations needed during this extraction have already been computed during the Pedersen-VSS phase, thus Step 4 is basically "for free" from a computational point of view. (See Section 6 for a discussion of performance versus security trade-offs between JF-DKG and New-DKG relevant to applications for which JF-DKG is sufficiently secure.)

Generation of h. We notice that our New-DKG protocol requires the public value h in order to run Pedersen's VSS. The crucial property for h is that the adversary should not know $\log_g h$. One possibility is to assume that a random h is made public as part of the

[5] Note that in this description we use discrete-log values unknown to the simulator; this provides a mathematical description of the output distribution of $\mathcal{S}$ useful for our analysis but does not require or assume that $\mathcal{S}$ can compute these values.

public parameters of the scheme (i.e., together with p, q, g). However, this requires a trusted process to do this. Another alternative is to have h generated jointly by the parties in a preliminary phase of the protocol. For this task we can use the protocol JF-DKG. Indeed, as we prove in Lemma 2 below (Section 5), if h is the output of a run of JF-DKG the adversary will not be able to compute $\log_g h$. Notice that there is no requirement for h to be chosen with uniform probability from the group generated by g, but only that $\log_g h$ be infeasible to compute.

Extension to adaptive adversaries. Canetti et al. [CGJ+] showed a modification of our New-DKG protocol which is secure against an *adaptive* adversary. In this model the attacker can make its decision of what parties to corrupt at any point during the run of the protocol (while in our static model the corrupted parties are fixed in advance before the protocol starts).

We do not know if our protocol is secure or not against an adaptive adversary. We do not know of any attack against our protocol in this model, yet no proof of security seems to go through. The reason is that when the simulator adjusts the value A_{n0} to hit the value y, it only knows t points on the polynomial defined by the A_{nk}'s which it interpolated in the exponent. This means that there are $n - t$ "inconsistent parties," i.e., parties that if corrupted will not be able to present an internal state consistent with the public information (indeed, their internal state should include their share of the polynomial). In the presence of an adaptive adversary these parties could be corrupted at any time and make the simulation fail. Being so many (more than half) even if the simulator chooses them at random, the adversary will have a substantial probability of choosing one. Against a static adversary we did not have this problem, since we knew which parties were corrupted and we could make sure that the inconsistent parties were all honest ones.

The only modification to our protocol introduced in [CGJ+] is in the y-extracting step (Step 4), where they replace our method of publishing $y_i = A_{i0} = g^{z_i}$ values via Feldman-VSS with the following: Each party broadcasts a pair $(A_{i0}, B_{i0}) = (g^{a_{i0}}, h^{b_{i0}})$ such that $A_{i0} \cdot B_{i0} = C_{i0} \bmod p$, and proves in zero-knowledge that he knows the discrete logs $\log_g(A_{i0})$ and $\log_h(B_{i0})$. Proving this ensures that $y_i = g^{z_i}$. If a party fails the proof then his shared value z_i is reconstructed via the Pedersen-VSS reconstruction, as in our New-DKG protocol. Notice that we need not reveal the values A_{ik} for $k > 0$.

This modification turns out to suffice to make the protocol secure against an adaptive adversary because it allows the construction of a simulator that, at any point in the simulation, has at most a *single* "inconsistent party." Namely, there is at most one party that if corrupted will make the simulation fail, while all other corruptions can be handled successfully by the simulator. The way the simulator proceeds is by choosing this "inconsistent party" at random and hoping the attacker will not corrupt him. If it does, the simulation rewinds to a previous state, a new choice of inconsistent party is made, and the simulation continues. It is shown in [CGJ+] that this brings the successful end of the simulation in expected polynomial-time.

The protocol in [CGJ+] requires the honest parties to *erase* data used during the computation and no longer needed. It also does not guarantee security in a setting in which several copies of the protocol can be executed concurrently. These two concerns were later addressed in [JL].

5. Secure Applications of JF-DKG: Threshold Schnorr Signatures

In Section 3 we showed that JF-DKG does not constitute a secure DKG protocol by building an attacker that could successfully bias the distribution of the generated value y away from the uniform distribution. This weakness of JF-DKG invalidates the use of this protocol as a generic (secure) DKG protocol. A natural question that arises is whether weaker security properties of the protocol suffice for its use in specific applications (namely, applications that impose more relaxed requirements on DKG security). Here we provide a positive answer to this question by showing that (i) JF-DKG enjoys the property that *no* adversary $\mathcal{A}$ can force an output y from the protocol for which $\mathcal{A}$ can compute the discrete logarithm of y; and (ii) this property suffices for building secure threshold Schnorr signature schemes (as well as threshold variants of other cryptosystems which enjoy a proof of security solely based on the hardness of the discrete-logarithm problem).

5.1. JF-DKG *Produces Hard Instances of the Discrete-Log Problem*

For showing that an attacker against JF-DKG cannot succeed in biasing the output y from the protocol into values for which it knows the discrete logarithm x, we use the following observation. Recall that the public key y produced by an execution of JF-DKG is a product of two values: y_G chosen with uniform distribution by the good parties, and $y_B = g^{x_B}$ where x_B is the contribution of the bad parties into the computation. The important fact to note is that this contribution x_B can be efficiently reconstructed by the good parties by interpolating the ($t + 1$ or more) good shares that they possess. This allows us to transform efficiently an attacker that is able to learn the discrete logarithm of the generated y into an attacker that breaks the Discrete-Log Assumption. The rest of this subsection is devoted to the formalization and proof of this claim.

If $\mathcal{A}$ is an adversary during an execution of JF-DKG, we denote by $(y, \mathit{view}(\mathcal{A})) \leftarrow \text{JF-DKG}(p, q, g)$ the event that JF-DKG outputs y as the public key and $\mathcal{A}$ gets the view $\mathit{view}(\mathcal{A})$ (with public inputs p, q, g). The following lemma states that if, according to Assumption 1, Discrete-Log is hard, then it is infeasible to compute the discrete-log of the output of JF-DKG.

Lemma 2. *Let $\mathcal{A}$ be a polynomial-time adversary that breaks into at most t servers during an execution of* JF-DKG. *Let $\mathcal{A}'$ be a polynomial-time machine that runs on the view of $\mathcal{A}$ after the execution of* JF-DKG. *Under the Discrete-Log Assumption* (*Assumption* 1), *we have that for every $\mathcal{A}$, $\mathcal{A}'$, every polynomial $P(\cdot)$, and sufficiently large k, $\Pr(\mathcal{A}'(p, q, g, y, \mathit{view}(\mathcal{A})) = x) \leq 1/P(k)$, where $(p, q) \in_R \mathit{PRIMES}(k)$, g is an element of order q in Z_p^*, and $(y = g^x, \mathit{view}(\mathcal{A})) \leftarrow \text{JF-DKG}(p, q, g)$.*

Proof. We prove this lemma, by reduction to the Discrete-Log Assumption. That is, we assume that we have a pair $\mathcal{A}$, $\mathcal{A}'$ that contradicts the conclusion of the lemma, and we show how to use them to solve a target instance $p, q, g, y_T = g^{x_T}$ of the discrete-log problem.

We set up a *simulation* of the execution of JF-DKG for $\mathcal{A}$. The simulator runs on input $p, q, g, y_T = g^{x_T}$ and will set up a virtual network of n parties for $\mathcal{A}$.

Let $\mathcal{B}$ be the set of parties corrupted by the adversary $\mathcal{A}$. Let $\mathcal{G}$ denote the set of remaining good parties which will be run by the simulator. Without loss of generality, we assume that $P_n \in \mathcal{G}$.

A regular instance of JF-DKG is started, with the following difference. The simulator will run the correct instructions of JF-DKG for all the parties in $\mathcal{G}\backslash\{n\}$. For party P_n instead it will run a simulation of the Feldman-VSS protocol which results in y_T being P_n's contribution to JF-DKG.

More specifically for party P_n the simulator does the following. It chooses t values $s_{nj} \in_R Z_q$, for $j \in \mathcal{B}$. Then P_n sends to $P_j \in \mathcal{B}$ the value s_{nj}. Notice that there exists a unique polynomial $f_n(z)$ of degree t such that $f_n(0) = x_T$ and $f_n(j) = s_{nj}$, $j \in \mathcal{B}$. Let

$$f_n(z) = a_{n0} + a_{n1}z + \cdots + a_{nt}z^t.$$

Although we cannot compute the values a_{ni}'s, we know that they are a linear combination of x_T and the s_{nj}'s, i.e.,

$$a_{ni} = \lambda_{n0}x_T + \sum_{j=1}^{t} \lambda_{nj}s_{nj}$$

for the appropriate Lagrange coefficients λ_{nj}. Thus we can compute

$$A_{ni} = g^{a_{ni}} = (y_T)^{\lambda_{n0}} \prod_{j=1}^{t} g^{\lambda_{nj}s_{nj}}.$$

Party P_n broadcasts the values $A_{n0}, \ldots, A_{nt}$.

The protocol now proceeds regularly. Notice that P_n has enough information to handle correctly any complaint that is brought up against him (since only bad parties can complain against P_n and the simulator knows the points $f_n(j)$ held by those parties).

Let y be the output of the protocol. We can write $y = y_T \cdot y_G \cdot y_B$ where y_G is the contribution of the parties in $\mathcal{G}\backslash\{n\}$, and y_B is the contribution of the parties in the set $\mathcal{B} \cap QUAL$ (i.e., the parties controlled by the adversary that are not disqualified).

Notice that if we write $y_G = g^{x_G}$ and $y_B = g^{x_B}$, the simulator knows both x_G and x_B. Indeed, the simulator chose x_G on behalf of the good parties. On the other hand, the contribution of each party in $\mathcal{B}$ that has not been disqualified is the free term of a polynomial of degree t, and the simulator holds at least $t + 1$ points on this polynomial; thus the simulator can compute each of these contributions and, in particular, the value x_B.

We now run $\mathcal{A}'$ on $(y, view(A))$ and denote by x its output. The simulator outputs $x_T = x - x_G - x_B$, which is the correct discrete-log of y_T whenever x is the correct discrete-log of y. So if $\mathcal{A}'$ guesses right with non-negligible probability, so does the simulator.

We end the proof by showing that the simulated view of $\mathcal{A}$ is identically distributed as in a real execution (this is to guarantee that $\mathcal{A}'$ guesses with the same probability as in a real execution of $\mathcal{A}$). This indeed is easy to see since the actions of parties $P_{t+1}, \ldots, P_{n-1}$ are exactly the same as in a real execution. With regard to party P_n the adversary sees t shares s_{nj} which are uniformly distributed, and the values $A_{ni} = g^{a_{ni}}$. The t-degree polynomial determined by the values a_{ni} is random, conditioned on the fact that $f_n(j) = s_{nj}$. Therefore, the probability distributions of these values is the same in the real and simulated executions. □

Remark. A consequence of the above lemma is the following. Even if the adversary can somewhat bias the distribution of the result y of JF-DKG, he cannot bias it too much. In particular, the adversary cannot make y fall into a pre-specified small (polynomial-sized) subset $Y \subset G$. We prove this formally in the following corollary.

Corollary 1. *Let $(p, q) \in_R PRIMES(k)$, let g be an element of order q in Z_p^*, and let $Y \subset G = (g)$ such that $|Y| \leq Q(k)$ for some polynomial $Q(\cdot)$. Let $\mathcal{A}$ be a polynomial-time adversary that breaks into at most t servers during an execution of* JF-DKG *and is given Y as input. Under the Discrete-Log Assumption* (*Assumption* 1), *we have that for every $\mathcal{A}$, every polynomial $P(\cdot)$, and sufficiently large k,* $\Pr(y \leftarrow \text{JF-DKG}(p, q, g), y \in Y) \leq 1/P(k)$.

Proof. The public key y produced by JF-DKG is a product of y_G chosen with uniform distribution by the good parties, and of value $y_B = g^{x_B}$, where x_B is the contribution of the bad parties into the computation. Notice that this contribution x_B can be reconstructed by the good parties via interpolation.

Assume, for the sake of contradiction, that there exists an adversary $\mathcal{A}$ and a polynomial $P(\cdot)$ such that $\Pr(y \leftarrow \text{JF-DKG}(p, q, g), y \in Y) > 1/P(k)$. We now show how to use $\mathcal{A}$ to solve an instance of the discrete-log problem

We are given a target value $y_T \in_R G$ and we want to compute $x_T \in Z_q$ such that $y_T = g^{x_T}$. We choose $y' \in Y$ uniformly at random and run a simulation of JF-DKG where the contribution of the good parties is $y_G = y' y_T^{-1}$. Since y_T is random in G, so is y_G. Moreover, by the same argument that one uses to argue secrecy of the Feldman-VSS protocol, the simulator can perfectly simulate the adversary's view of the contribution of the good parties to any value y_G which is uniformly distributed in G.

By assumption, the adversary with probability $> 1/P(k)$, creates x_B such that $y = y_G g^{x_B} \in Y$. Since we chose y' uniformly at random in Y, with probability $1/|Y| \geq 1/Q(k)$ we have that $y = y'$. If this event happens then $x_B = \log_g(y_T)$, and hence the simulator would solve the discrete-log problem on a random instance y_T with non-negligible probability $> 1/P(k)Q(k)$. □

In the next section we see how the above corollary enables the simulation of certain threshold signature schemes that use the JF-DKG protocol as a subroutine.

5.2. *Threshold Schnorr Signature Scheme Using* JF-DKG

Here we show that the limited security guarantee proven for JF-DKG in Lemma 2 is sufficient to build a threshold Schnorr signature scheme that uses JF-DKG and for which security can be proven solely based on the Discrete-Log Assumption.

Schnorr's Signature Scheme

We first recall Schnorr's signature scheme [S1]. Let H be a hash function, H: $\{0, 1\}^* \rightarrow Z_q$, which we model as an ideal random function. The private key is x, chosen at random in Z_q, and 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 $r = g^k \bmod p$, $c = H(m, r)$, and $s = k + cx \bmod q$. 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 [FS], 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 [GMR] reduces to the discrete-log assumption, as proven by [PS].

We recall this proof here because it helps in understanding 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 $\log_g y$ as follows. It runs the adversary until a forgery (c, s) on some message m is generated. Now, since H is a random function, then, except for negligible probability, the adversary must invoke H on the point (m, r) where $r = g^s y^{-c}$, or 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' such that $r = g^{s'} y^{-c'}$, and hence the simulator can now compute $\log_g y = (s - s')/(c' - c)$. One can show that if the adversary's probability of forgery is ε, this simulation succeeds with probability $\varepsilon^2/4q_H$. (This probability is the result of multiplying the probability $O(\varepsilon)$ that the adversary forges in the first run, times the probability $O(\varepsilon/q_H)$ that it forges on the second run *and* that it chooses to forge on the *same* (m, r) query out of its q_H queries to H). We refer to [PS] for the full proof.

Threshold Version of Schnorr's Scheme Using Pedersen's DKG

The threshold version of Schnorr's scheme using JF-DKG is a very simple protocol. It works in the straightforward way as all standard threshold dlog-based protocols, e.g., [GJKR1], except that secret reconstruction is done on additive rather than polynomial shares.[6]

To initialize the threshold signature scheme, first the JF-DKG protocol is executed for distributed key generation, i.e., it outputs secret-sharing of a private key x and a public value, the public key $y = g^x$. Notice that at this point each player P_i in the set $QUAL$ (the set of players who performed good sharings) holds an additive share z_i of x—refer to Fig. 2).

The threshold Schnorr signature protocol TSch then proceeds as follows, on input a message m (see Fig. 4): We limit the generation of the signature to the parties in $QUAL$:

[6] 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 [FMY], [CGJ+] or to handle the no-erasure and concurrent adversary in [JL].

Threshold Signature Protocol TSch

Inputs: Message m to be signed, plus the secret-sharing of x generated by the initial JF-DKG protocol. In particular, each party $P_i \in QUAL$ holds an additive share z_i of x while values $y = g^x$ and $y_i = g^{z_i}$ for every P_i are public. Each value z_i is itself secret-shared with Feldman-VSS.

Outputs: Schnorr's signature (c, s) on m.

1. Parties perform an instance of the JF-DKG protocol (Fig. 1). Denote the outputs of this run of JF-DKG as follows. Each party $P_i \in QUAL'$ holds an additive share u_i of the secret-shared secret k. Each of these additive shares is itself secret-shared with Feldman-VSS. We denote the generated public values $r = g^k$ and $r_i = g^{u_i}$ for each P_i.
2. Each party locally computes the challenge $c = H(m, r)$.
3. Parties perform the reconstruction phase of Feldman's secret-sharing of value $s = k + cx$ as follows. Each party $P_i \in QUAL \cap QUAL'$ broadcasts its additive share $s_i = u_i + cz_i$. Each share is verified by checking if $g^{s_i} = r_i y_i^c$. Otherwise x_i and z_i are reconstructed and s_i is computed publicly.

   Values z_i for each party in $QUAL - QUAL'$ are publicly reconstructed and for those parties s_i is set to cz_i

   The protocol outputs signature (c, s) where $s = \sum_{i \in QUAL} s_i$.

Fig. 4. Threshold signature protocol for Schnorr's signature scheme.

they run an instance of the JF-DKG protocol to generate a secret-sharing of the one time secret k and the public value $r = g^k$; let $QUAL'$ be the set of parties who performed good sharing in this step. Again each party in $QUAL'$ holds an additive share, say u_i, of k.

Each party then locally computes the challenge $c = H(m, r)$, and each party $P_i \in QUAL \cap QUAL'$ broadcasts its *additive* share $s_i = u_i + cz_i$. Notice that these values can be publicly verified by checking if $g^{s_i} = K_{i0}(X_{i0})^c$, where $X_{i0} = y_i = g^{z_i}$ and $K_{i0} = r_i = g^{u_i}$ are verification values broadcast by party P_i, during the initial key-generation JF-DKG protocol, and during the JF-DKG protocol that generates r and the sharing of k, respectively. If verification fails for some P_i, the parties reconstruct the Feldman secret-sharing of both x_i and k_i and compute s_i publicly. The values z_i of the players in $QUAL$ but not in $QUAL'$ are publicly reconstructed (these are bad players) and for those parties s_i is set to cz_i. Finally, s is computed as $s = \sum_{i \in QUAL} s_i$. Therefore the secret s can be efficiently reconstructed as in the Feldman-VSS protocol.

Efficiency Considerations

Note that without faults, the above protocol requires only one round of broadcast during the JF-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 party P_i. When P_i receives and validates a message m to be signed, it triggers the JF-DKG protocol of Step 1 and broadcasts along it the message m. This allows the parties 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 party P_i sends to the verifier the value c and its share s_i. The verifier can check if signature $(c, s) = (c, s_1 + \cdots + s_n)$ is valid, and the protocol falls back to the robust reconstruction only if the signature

does not verify. Note also that Step 1 of the protocol can be performed off-line, and thus with preprocessing the threshold Schnorr signature is non-interactive as long as no faults occur.

Security of Threshold Signature Scheme (JF-DKG, TSch)

The robustness of the (JF-DKG, TSch) threshold signature scheme follows straightforwardly from the robustness of the JF-DKG protocol. The interesting part is the proof of unforgeability.

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

Before presenting a formal proof of Theorem 2 it is illustrative to discuss the main ideas behind this proof and show why the most "natural" approach to the proof does not work. If we consider the proof of security for centralized Schnorr's signatures as sketched above, we can see that the simulator first chooses s, c at random, and computes $r = g^s y^{-c}$; the pair (r, s) becomes a good signature on m, once the simulator "patches" the random oracle with the value $H(m, r) = c$. In order to translate this idea to the threshold scenario, we see that the simulator must simulate an execution of JF-DKG which must "hit" the computed r as its output. However, as demonstrated in Section 3, it is not possible to simulate JF-DKG in this way. That is because the adversary can bias the distribution of the output and make sure that it will never hit r.

So we need to use a different strategy. The simulator runs on input a target value y_T and it wants to compute $x_T = \log_g y_T$. The simulator embeds this target value in the public key generated by a simulation of JF-DKG as the contribution of one of the good parties (say P_n's contribution $y_n = y_T$, exactly as in the proof of Lemma 2). Then to simulate a signature protocol on message m, the simulator chooses s_n, c at random and sets $r_n = g^{s_n} y_n^{-c}$. Now the simulator must make sure that whatever value r is computed, the random oracle can be set as $H(m, r) = c$. Indeed, if that happens, the simulator can just run the protocol TSch on behalf of the rest of the good parties and the signature will work out.

At this point of the simulation there is a (polynomially small) set of pairs $(m, \tilde{r})$ on which H has been queried. Because of the "weak" secrecy property of JF-DKG (see Corollary 1 after the proof of Lemma 2), we know that the adversary has negligible probability of hitting those. However, the adversary could hit a pair $(m, \tilde{r})$ asked *during* the computation of Step 1, but before Step 1 is completed. To cover these possibilities we let the simulator flip a bit b. If $b = 0$, it runs TSch normally for the remaining good parties; the simulation succeeds if, for the resulting value r, the pair (m, r) has not been asked yet. Otherwise, if $b = 1$, the simulator sets to c one of those queries $(m, \tilde{r})$ asked during the computation of Step 1 (a randomly chosen one). In this case the simulation succeeds if the resulting r equals $\tilde{r}$. Either way the simulation succeeds with a non-negligible probability.

We proceed with the full proof of Theorem 2.

Proof. Assume that there exists an adversary which breaks the unforgeability property of this signature scheme. We construct a simulator $\mathcal{S}$ which, using this adversary, will compute a discrete logarithm on input a random challenge value y_T in G. In the course of this simulation, $\mathcal{S}$ answers the adversary's queries to oracle H at random, except in a crucial case specified below. Let $\mathcal{B}$ be the set of corrupted parties and let $\mathcal{G}$ be the set of good parties. Assume without loss of generality that $P_n \in \mathcal{G}$. Let q_H, q_S be upper bounds on the number of H queries and the number of the signature queries, respectively, that the adversary makes.

To compute the discrete logarithm of a target value y_T, the simulator $\mathcal{S}$ embeds it in the value y_n contributed by party P_n to the public key generated in the initial JF-DKG protocol. In other words, the simulator follows the JF-DKG protocol on behalf of parties $P_i \in \mathcal{G}\backslash\{n\}$ as prescribed, but for party P_n the simulator *simulates* the adversarial view of the Feldman-VSS protocol performed as a part of JF-DKG by party P_n so that public value y_n broadcast by P_n is equal to the target value y_T.

After simulating the initial key generation protocol in this way, the simulator $\mathcal{S}$, for every message m submitted by the adversary for a signature, simulates an execution of TSch on this message as follows. $\mathcal{S}$ chooses values c and s_n at random in Z_q, computes $r_n = g^{s_n} y_n^{-c}$, and in Step 1 of TSch it follows the protocol on behalf of parties $P_i \in \mathcal{G}\backslash\{n\}$ as prescribed, but for P_n it again simulates the Feldman-VSS protocol in this step so that the value r_n broadcast by that party 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 parties $P_i \in \mathcal{G}\backslash\{n\}$ as prescribed, while for P_n it can publish the previously chosen value s_n. Notice that since $g^{s_n} = r_n y_n^c$, this value is accepted. Moreover, this yields a valid signature because $g^s = g^{\Sigma s_i} = \prod(g^{s_i}) = \prod(r_i y_i^c) = (\prod r_i)(\prod y_i)^c = r y^c$, and $c = H(m, r)$.

To set $H(m, r) = c$, where r is computed in Step 1 of TSch, the simulator $\mathcal{S}$ flips a coin b, before sending any Step 1 simulation messages to the adversary. If $b = 0$, then during the simulation of Step 1, $\mathcal{S}$ answers with random values all the (new) queries to H made by the adversary. If the value r computed in Step 1 is such that H was already asked on (m, r), the simulator fails. Otherwise, $\mathcal{S}$ sets $H(m, r)$ as c and succeeds. If $b = 1$, then the simulator chooses at random an index $\ell \in \{1, \ldots, q_H\}$ and if the adversary makes any $(m, r^{(j)})$ queries to H *after* the simulator publishes all values r_i for $P_i \in \mathcal{G}$ but *before* the adversary has finished sharing his values u_i for $P_i \in B$ (these are substeps of the simulation of Step 1), the simulator answers the ℓth query to H with c. If the value r output in this Step 1 is equal to $r^{(i)}$, then this counts as the simulator's success because $H(m, r) = c$. Otherwise the simulator fails.

We now estimate the simulator's probability of success. We use the intuition outlined in Section 3 in the discussion of security of the JF-DKG protocol. Let ε_{bef} (resp. ε_{dur}) be the probability with which the adversary, after seeing $r_G = \prod_{i \in \mathcal{G}} r_i$, 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 beginning of (resp. *during*) the simulation of Step 1. Notice that by Corollary 1, ε_{bef} must be negligible.

If $b = 0$ the probability of success of the simulation is clearly $(1 - \varepsilon_{\text{bef}})(1 - \varepsilon_{\text{dur}})$, i.e., the probability that (m, r) was not queried before or during the simulation of Step 1. On

the other hand, if $b = 1$ the success probability is at least $\varepsilon_{\text{dur}}/q_H$, i.e., the probability that (m, r) was asked during the simulation of Step 1, times the probability that we guess in which query it was asked.

Hence, the overall success probability is at least

$$\tfrac{1}{2}(1 - \varepsilon_{\text{bef}} - \varepsilon_{\text{dur}} + \varepsilon_{\text{bef}}\varepsilon_{\text{dur}}) + \frac{\varepsilon_{\text{dur}}}{2q_H} \geq \frac{1}{2} - \frac{\varepsilon_{\text{bef}}}{2} - \frac{\varepsilon_{\text{dur}}}{2}\left(1 - \frac{1}{q_H}\right).$$

The above is minimized for $\varepsilon_{\text{dur}} = 1$ and thus the probability of success of the simulation is larger than $\varepsilon_{ss} = 1/2q_H - \varepsilon_{\text{bef}}/2$, which is negligibly close to $1/(2q_H)$ by Corollary 1.

Therefore, if $\mathcal{S}$ repeats this simulation γ/ε_{ss} times, the probability that it fails is at most $e^{-\gamma}$. In this way, with probability $(1 - e^{-\gamma})^{q_S}$, the simulator successfully goes through the simulation of each of the q_S instances of the TSch protocol. Setting $\gamma = \ln 2q_S$, 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 non-negligible probability ε, the simulator will get some forged signature (c, s) on some message m with probability at least $\varepsilon/2$ (the factor of $1/2$ comes from the above analysis).

By applying the same "forking lemma" argument as Pointcheval and Stern [PS] used to prove the security of the standard Schnorr signature (see the beginning of Section 5.2), we can argue the following. With probability at least $\varepsilon/2$, the simulator $\mathcal{S}$ gets one forgery *and* if he re-winds the adversary to the point when the adversary asks query $H(m, r)$ for $r = g^s y^{-c}$, and then continues to simulate from that point on with fresh randomness (in particular answering this query to H with fresh randomness c'), then $\mathcal{S}$ has about a $(\varepsilon/2)/q_H$ chance of getting a second forgery on the same message m but relative to a different random function H. In this case $\mathcal{S}$ gets two pairs (c, s) and (c', s') such that $r = g^s y^{-c} = g^{s'} y^{-c'}$ and thus can compute $\log_g y$ and from it also $\log_g y_n = \log_g y_T$ because $\mathcal{S}$ knows all $x_i = \log_g y_i$ for $P_i \neq P_n$.

Therefore, if the assumed threshold signature forger runs in time T and succeeds in forgery with probability ε, then if probability ε_{bef} is small, $\mathcal{S}$ computes the discrete logarithm in time $2c/\varepsilon_{ss} * T = 2\ln(2q_S)/\varepsilon_{ss} * T$ with probability at least about $\varepsilon^2/(4q_H)$. □

To exemplify the kind of security degradation created by the above reduction, consider the case in which T is big enough so that $\varepsilon \approx 1$. Then the simulator's computation is longer than the adversary's computation by the factor $\alpha = 4q_H * 2\ln 2q_S/\varepsilon_{ss}$. If $\varepsilon_{\text{bef}} < 1/(4q_H)$, then $\varepsilon_{ss} = 1/(2q_H) - \varepsilon_{hit} > 1/(4q_H)$, and hence $\alpha < 2^5 q_H^2 \ln(2q_S)$. Taking $q_S < 2^{32}$, we get $\alpha < 2^{11} q_H^2$ as the security degradation factor.

6. Security versus Efficiency Implications

The results presented above show that for certain threshold cryptosystems, like the threshold Schnorr signature protocol, we can use either JF-DKG or New-DKG and still obtain a secure protocol.

However, if we use New-DKG this will require one extra round of broadcast communication. Keeping communication to a minimum is especially important for a scheme

like Schnorr's, whose main cost lies in the DKG subprotocol that it uses. Thus it would seem that JF-DKG is the better implementation choice for threshold Schnorr's signatures (resulting in the protocol TSch).

However, the security reduction we are able to show for the TSch threshold Schnorr signature scheme (which uses JF-DKG) has a q_H^2 factor of security degradation compared with the security of the discrete-log problem (DLP). This represents an additional q_H degradation factor over the provable security of the centralized version of Schnorr signatures, for which Pointcheval and Stern [PS] show a reduction from DLP with a single q_H factor in security degradation. On the other hand, if we implement threshold Schnorr signatures using New-DKG (we denote this threshold Schnorr scheme by new-TSch) there is no loss of security between the threshold and centralized scheme and therefore the total security degradation relative to DLP is only q_H.

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 security bound b. In fact, the increase in computation resulting from the fact that the TSch scheme needs to work over a larger field to guarantee the same security as the new-TSch scheme outweighs the benefits resulting from the fact that the TSch scheme requires only one round of broadcast while new-TSch needs two.

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 [PS] 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 new-TSch scheme to the Schnorr signatures is tight, this implies that the new-TSch scheme is secure in the same 2^{160}-DLP group. On the other hand, the security reduction of Theorem 2 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, the new-TSch threshold Schnorr scheme requires a factor of $\alpha = 160/80 = 2$ growth in the DLP security parameter, while the TSch threshold Schnorr scheme requires a factor of $\alpha = 240/80 = 3$ growth in the DLP security parameter.

Recall that the cost of exponentiation modulo p with a $|q|$-bit exponent grows like $O(|q| \cdot |p|^{1.6})$. If we consider the DLP security in the classic Z_p^* setting, the factor α growth in the security parameter implies factor α^3 growth in the size of modulus p and factor α 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 α, and hence the overall cost of exponentiation grows only by factor $\alpha^{2.6}$.

As a reference point for the speed of cryptographic operations we take the performance table of [W], where on a Celeron 850 MHz an exponentiation in Z_p^* 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 Verheul [LV], that in both settings DLP has security

parameter 80 (i.e., the DLP in such groups is secure against an adversary with a $b = 2^{80}$ computational upper bound). If n is the number of parties and $t \approx 0.5 \cdot n$, then in the TSch threshold Schnorr protocol, each party makes $1.5 \cdot n$ long exponentiations, while in new-TSch each party performs $2.5 \cdot n$ long exponentiations.

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

The TSch scheme may be slightly faster than new-TSch in the elliptic curve setting with parties distributed over the Internet, but probably not when the parties are on a LAN network. Each party's computation is $18 \cdot 2^{2.6} \cdot 5$ ms $= 0.5$ s for new-TSch, and $11 \cdot 3^{2.6} \cdot 5$ ms $= 0.95$ s for TSch. Therefore, if the parties are distributed over the Internet where one round of broadcasts causes a delay of about 1 s, the 0.45 s difference in computation is outweighed by the broadcast cost, and therefore the TSch scheme is preferable. On the other hand, the new-TSch scheme still wins with TSch if the parties are connected via a LAN, where the extra round of broadcast costs only about 0.1 s.

Finally, 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 over new-TSch: The computation time per party in TSch would be only about $11 \cdot 5$ ms $= 55$ ms compared with $18 \cdot 5$ ms $= 90$ ms for new-TSch. Moreover the TSch scheme, having only one round of broadcast would incur a communication delay of about 100 ms in the LAN setting or 1 s in the Internet setting, while the new-TSch scheme would take, respectively, 200 ms and 2 s.

Acknowledgments

We thank Don Beaver for motivational discussions on this problem.

Appendix. Other Insecure Variants of the JF-DKG Protocol

We describe several variants and extensions of the JF-DKG protocol which have appeared in the literature: signatures on shares, commitments to y_i, committing encryption on broadcast channel, committing encryption with reconstruction, and "stop, kill, and rewind." We show that all of them fail to achieve the correctness property (C3) and the secrecy (or simulatability) requirement as presented in Section 4.1.

Signatures in Share Distribution

In the original protocol proposed in [P1] each party (acting as dealer) signs the shares he distributes in Step 1. This was supposed to aid the honest parties in disqualifying

dishonest dealers, because a party which receives an incorrect share (i.e., not satisfying (3)) could prove that the dealer is dishonest by broadcasting this share with the dealer's signature. Indeed, the original protocol from [P1] uses this simplified procedure in Step 3 for disqualifying dishonest dealers: a party is disqualified if one valid complaint (i.e., incorrect share with valid signature) is broadcast against him.

We note first that with this modification even a single execution of Feldman's protocol fails to be a VSS. Indeed, if a dealer gives neither a correct share nor a signature to some party, the party cannot prove the dealer wrong by the above method.

Thus the joint execution of n such protocols may fail to produce a correct sharing at all. Also notice that the basic idea of the attack in Section 3 still works (P_1 can give to P_2 two signed shares, a correct one, and an incorrect one and all other parties consistent shares with valid signatures, and then (using only P_2) decide if he wants to be disqualified or not).

Initial Commitment Stage

Another difference between the original protocol presented in [P1] and JF-DKG of Fig. 1 is the use of an initial commitment stage in [P1], so that the modified protocol looks as follows:

1. Each P_i chooses $z_i \in Z_q$ uniformly at random, computes $y_i = g^{z_i} \bmod p$, chooses a random string r_i, and broadcasts a commitment $C_i = C(y_i, r_i)$ to all members.
2. Each P_i opens the commitment C_i by broadcasting proper y_i and r_i. A party who fails to do so is disqualified.

3–6. These steps follow Steps 1–4 of the JF-DKG protocol, with the exception that the parties disqualified in Step 2 above are barred from participation and the $f_i(z)$ polynomials picked in Step 3 (Step 1 in JF-DKG) have only $a_{i1}, \ldots, a_{ik}$ satisfy $f_i(0) = z_i$.

This initial commitment stage is supposed to force the parties to choose their random secrets z_i independently from one another in order to ensure the uniform distribution of the final sum x, but it fails to do that. Indeed, the attack in Section 3 is not based on how dishonest parties choose their contribution, but rather on their ability to pull such contribution out of the lot after seeing the honest parties' contributions. Thus the dishonest parties can follow the protocol, including the extra commitment stages, and still nothing stops them from carrying out the attack described in Section 3. If P_i decides to do so, he can get his value disqualified and the remaining parties will not add y_i to the public key y even if it matches the initial commitment C_i.

Committing Encryption on a Broadcast Channel

The attack described in Section 3 seems to rely on the assumption that each pair of parties is connected by a private channel which allows P_1 and P_2 to "lie" to the other parties about the share P_1 sent to P_2. This is possible in several implementations of private channels, for example physically untappable ones (e.g., lead pipes) or private channels built out of one-time pad encryption.

It would seem that using some form of encryption to implement private channels may help in thwarting the attack, since a complaining party may be required to "open" the encrypted message he received to prove that the incorrect share really came from the dealer. This strategy was followed in a modification of JF-DKG used for proactive secret sharing [HJKY], [HJJ+]). However, we prove now that this is not the case. The attack can still be carried out even if parties exchange messages using encryption.

Let us assume that the parties communicate simply via the the broadcast channels. Private communication is achieved via a public key "committing" encryption scheme, i.e., an encryption scheme that not only ensures the confidentiality of a message but also commits the sender to the message being sent.[7] In other words, to send a message m secretly to recipient R, the sender picks a random vector r and broadcasts a ciphertext $E = ENC^r_R(m)$. We assume that the recipient using the secret key of ENC_R can recover both m and r, and so he is able to prove that $E = ENC^r_R(m)$ to all other parties by revealing m, r.[8]

The JF-DKG variant using such committing encryption introduces the following modifications to the protocol of Fig. 1: Parties send the shares s_{ij} in Step 1 by broadcasting their committing encryptions $E_{ij} = ENC^{r_{ij}}_{P_j}(s_{ij})$. Then in Step 2 a valid complaint has to consist of a pair (s_{ij}, r_{ij}) such that s_{ij} is an incorrect share and $E_{ij} = ENC^{r_{ij}}_{P_j}(s_{ij})$. In Step 3 we disqualify a party if there is a single valid complaint against him.

Unfortunately, the protocol modified in such a way is still insecure. P_1 sends to P_2 a bad share s^*_{12} via the committing encryption in Step 2. At the same time however it chooses the sharing polynomial so that the correct share s_{12} is some fixed value on which P_1 and P_2 agreed earlier (or in other words think of P_1 and P_2 as the same entity, the adversary). If the adversary wants P_1's contribution to be "in", P_2 will not complain in Step 3, but will use s_{12} in all further computation. If the adversary wants P_1 disqualified, P_2 broadcasts (s^*_{12}, r_{ij}).

Committing Encryption with Reconstruction

In the JF-DKG with the committing encryption variant described above, we are following the policy of disqualifying a party P_i as soon as a single valid complaint is filed. Yet, it seems that as the values A_{ik}, $0 \le k \le t$, are broadcast by party P_i in Step 1 of JF-DKG and there is a fixed polynomial and a fixed secret, if we have enough shares to reconstruct it we would not need to disqualify the dealer based on a single complaint. We could follow this alternative policy instead: if a valid complaint is filed against P_i, all other parties open the encrypted shares P_i sent them: if more than $t + 1$ of them match (3) then publicly reconstruct z_i, otherwise disqualify P_i.

However, even for this policy we show a strategy for the adversary to decide if (say) P_1 is disqualified or not, at a stage where he can influence the distributions. Assume $n = 2t + 1$, P_1 sends correct encrypted shares to all the parties (honest and dishonest)

[7] Contrast this to *deniable encryption* [CDNO] where the sender or the receiver may lie about the content of encrypted messages.

[8] Not all encryption schemes allow the receiver to recover both the message m and the random vector r. However what we are arguing here is that even if the committing encryption provided the random vector recovery, this JF-DKG variant still cannot be made secure.

except for one honest party. This party will complain and everybody is required to open their encrypted shares. The honest parties have only t matching shares, thus an additional one is required in order to incorporate P_1's value into the computation. Thus, the decision of whether the secret will be considered is again left in the hands of the adversary.

We can also show that variations on this policy do not work.

"Stop, Kill, and Rewind" Procedure

Notice that in some of the above attacks against JF-DKG and its variants, the adversarial strategy involves a behavior on the part of some of the parties which can be publicly identified as faulty. We could modify the JF-DKG protocol so that whenever some party is clearly faulty, the protocol stops, that party is excluded from the set of parties, and the protocol is started from scratch in the smaller group of parties. Note that this can happen at most t times.

However, the adversary can still skew the distribution of the outputs. He just follows the same strategy as in Section 3. The event that the protocol is repeated is conditioned on the fact that the adversary made a dishonest party visibly faulty. Thus the distribution of the final output is not necessarily uniform, even if the second repetition has a uniformly distributed output.

For the example in Section 3, the final output y ends with 0 if either

- α ended with 0 and P_1 was not disqualified (probability $1/2$).
- P_1 was disqualified (i.e., α ended with 1, which happens with probability $1/2$) and the output of the second run ends with 0 (probability $1/2$).

Thus the probability that in the presence of such an adversary the protocol outputs a value y which ends with 0 remains $1/2 + 1/2 * 1/2 = 3/4$.

The "stop, kill, and rewind" procedure can be employed in other variants of JF-DKG. For example, it was used in [HJKY], [HJJ$^+$] in the committing encryption variant. In each case a similar argument shows that the adversary can force the distribution of the resulting y not to be uniform.

References

[CDNO] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In *Crypto* '97, pages 90–104. LNCS No. 1294. Springer-Verlag, Berlin, 1997.

[CGH] R. Canetti, O. Goldreich and S. Halevi. The random oracle methodology, revisited. *Proc. STOC*, pages 209–218, 1998.

[CGJ$^+$] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. In *Crypto* '99, pages 98–115. LNCS No. 1666. Springer-Verlag, Berlin, 1999.

[CGMA] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proc. 26th FOCS*, pages 383–395. IEEE, Piscataway, NJ, 1985.

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

[CMI] 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.

[CP] C. Cachin and J. A. Poritz. Secure intrusion-tolerant replication on the Internet. In *Proc. Internat. Conf. on Dependable Systems and Networks* (*DSN*-2002), pages 167–176, Washington, DC. IEEE, Piscataway, NJ, 2002. (See also http://eprint.iacr.org/.)

[DF] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Crypto* '89, pages 307–315. LNCS No. 435. Springer-Verlag, Berlin, 1989.

[ElG] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, IT31:469–472, 1985.

[Fel] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. 28th FOCS*, pages 427–437. IEEE, Piscataway, NJ, 1987.

[FGMY] Y. Frankel, P. Gemmell, P. Mackenzie, and M. Yung. Optimal resilience proactive public-key cryptosystems. In *Proc. 38th FOCS*, pages 384–393. IEEE, Piscataway, NJ, 1997.

[FMY] 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. LNCS No. 1643. Springer-Verlag, Berlin, 1999.

[FS] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Crypto'86*, pages 186–194. LNCS No. 263. Springer-Verlag, Berlin, 1986.

[GJKR1] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. *Inform. Comput.*, 164:54–84, 2001.

[GJKR2] 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. LNCS No. 1592. Springer-Verlag, Berlin, 1999.

[GJKR3] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Revisiting the distributed key generation for discrete-log based cryptosystems. In *RSA-CT* '03, pages 373–390. LNCS No. 2612. Springer-Verlag, Berlin, 2003.

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

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

[HJJ+] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *Proc.* 1997 *ACM Conf. on Computers and Communication Security*, pages 100–110, 1997.

[HJKY] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing, or: How to cope with perpetual leakage. In *Crypto* '95, pages 339–352. LNCS No. 963. Springer-Verlag, Berlin, 1995.

[JL] S. Jarecki and A. Lysyanskaya. Adaptively secure threshold cryptosystems without erasures. In *Eurocrypt* '00, pages 221–242. LNCS No. 1807. Springer-Verlag, Berlin, 2000.

[Lan] S. Langford. Weaknesses in some threshold cryptosystems. In *Crypto* '96, pages 74–82. LNCS No. 1109. Springer-Verlag, Berlin, 1996.

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

[LV] A. K. Lenstra and E. R. Verheul Selecting cryptographic key sizes. In *J. Cryptology*, 14(4):255–293, 2001.

[OY] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *PODC* '91, pages 51–59. ACM Press, New York, 1992.

[P1] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Crypto* '91, pages 129–140. LNCS No. 576. Springer-Verlag, Berlin, 1991.

[P2] T. Pedersen. A threshold cryptosystem without a trusted party. In *Eurocrypt* '91, pages 522–526. LNCS No. 547. Springer-Verlag, Berlin, 1991.

[PK] C. Park and K. Kurosawa. New ElGamal type threshold digital signature scheme. *IEICE Trans. Fundamentals*, E79-A(1):86–93, January 1996.

[PS] D. Pointcheval, and J. Stern, Security proofs for signature schemes. In *Eurocrypt* '96, pages 387–398. LNCS No. 1070. Springer-Verlag, Berlin, 1996.

[S1] P. Schnorr. Efficient identification and signatures for smart cards. In *Crypto* '89, pages 235–251. LNCS No. 435. Springer-Verlag, Berlin, 1989.

[S2] A. Shamir. How to share a secret. *Comm. ACM*, 22:612–613, 1979.

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

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