Secure Distributed Key Generation for Discrete-Log Based Cryptosystems

Rosario Gennaro[1], Stanisław Jarecki[2], Hugo Krawczyk[3], and Tal Rabin[1]

[1] IBM T.J.Watson Research Center, PO Box 704, Yorktown Heights, NY 10598, USA
{rosario,talr}@watson.ibm.com
[2] MIT Laboratory for Computer Science, 545 Tech Square, Cambridge, MA 02139, USA, stasio@theory.lcs.mit.edu
[3] Department of Electrical Engineering, Technion, Haifa 32000, Israel, and IBM T.J. Watson Research Center, New York, USA
hugo@ee.technion.ac.il

Abstract. Distributed key generation is a main component of threshold cryptosystems and distributed cryptographic computing in general. Solutions to the distributed generation of private keys for discrete-log based cryptosystems have been known for several years and used in a variety of protocols and in many research papers. However, these solutions fail to provide the full security required and claimed by these works. We show how an active attacker controlling a small number of parties can bias the values of the generated keys, thus violating basic correctness and secrecy requirements of a key generation protocol. In particular, our attacks point out to the places where the proofs of security fail.
Based on these findings we designed a distributed key generation protocol which we present here together with a rigorous proof of security. Our solution, that achieves optimal resiliency, can be used as a drop-in replacement for key generation modules as well as other components of threshold or proactive discrete-log based cryptosystems.

Keywords: Threshold Cryptography. Distributed Key Generation. VSS. Discrete Logarithm.

1 Introduction

Distributed key generation is a main component of threshold cryptosystems. It allows a set of n servers to jointly generate a pair of public and private keys according to the distribution defined by the underlying cryptosystem without having to ever 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, $t+1$, 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 schemes, distributed key generation amounts to generating a

J. Stern (Ed.): EUROCRYPT'99, LNCS 1592, pp. 295–310, 1999.
© Springer-Verlag Berlin Heidelberg 1999

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 players and forces them to follow an arbitrary protocol of his 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 players 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, of course, what is learned from the public value y.

Solutions to the shared generation of private keys for discrete-log based threshold cryptosystems [DF89] have been known and used for a long time. Indeed, the first DKG scheme was proposed by Pedersen in [Ped91a]. It then appeared, with various modifications, in several papers on threshold cryptography, e.g., [CMI93, Har94, LHL94, GJKR96, HJJ+97, PK96, SG98], and distributed cryptographic applications that rely on it, e.g., [CGS97]. 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 discrete-log based signature schemes (for example the r value in a (r, s) DSS signature as in [GJKR96]). Another example is the generation of the refreshing polynomial in proactive secret sharing and signature schemes [HJKY95, HJJ+97, FGMY97].

The basic idea in Pedersen's DKG protocol [Ped91a] (as well as in the subsequent variants) is to have n parallel executions of Feldman's verifiable secret sharing (VSS) protocol [Fel87] in which each player P_i acts as a dealer of a random secret z_i that he 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.

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. 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; specifically, they show that simulation arguments (à la zero-knowledge) as used to prove the secrecy of these protocols must fail.

In contrast to the above, we present a 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 player 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 (verifiable secret sharing) protocol as presented in [Ped91b]. Very importantly, our solution preserves most of the efficiency and simplicity of the original DKG solution of [Ped91a], in particular it has comparable computational complexity and the same optimal threshold of $t < n/2$.

Organization: In Section 2 we present the basic communication and adversarial models for our protocols. In Section 3 we describe previously proposed solutions to the DKG problem and show where they fail. In Section 4 we present our solution and its full analysis; we also discuss some other applications of our protocol. Finally, in Section 5 we discuss an enhanced (and more realistic) security model under which our solution works as well.

2 Preliminaries

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

For simplicity of the discussion that follows, we assume a *fully synchronous* communication model, i.e. that messages of a given round in the protocol are sent by all players simultaneously, and that they are simultaneously delivered to their recipients. This model is not realistic enough for many applications, but it is often assumed in the literature; moreover, our attacks against known DKG protocols (Section 3) work even in this simplified setting.

In Section 5 we introduce a more realistic, *partially synchronous* communication model. Our solution to the DKG problem (Section 4) and its security proof work in this strictly stronger adversarial model.

The Adversary. We assume that an adversary, $\mathcal{A}$, can corrupt up to t of the n players 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 players 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 players at the beginning of the protocol (see section 4.2 for a reference to a recent extension of our results to the non-static – or *adaptive* – adversary setting).

3 Distributed Key Generation in DLog-Based Schemes

In this section we define the minimal requirements for a secure distributed key generation protocol. We show how previous solutions fail to satisfy these requirements. We also discuss the applicability of our attacks to other existing distributed protocols.

3.1 Requirements of a Secure DKG Protocol

As we mentioned in the introduction, distributed generation of keys in a discrete–log based scheme amounts to generating a secret sharing of a random, uniformly distributed value x and making public the value $y = g^x$. Specifically, in a discrete–log based scheme with a large prime p and an element g of order q in $\mathbb{Z}_p^*$ where q is a large prime dividing $p-1$, the distributed protocol DKG performed by n players $P_1, \ldots, P_n$ 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 players 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 $\mathbb{Z}_q$ (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 $\mathcal{A}$, there exists a (probabilistic polynomial-time) simulator SIM, such that on input an element y in the subgroup of $\mathbb{Z}_p^*$ 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, and where $\mathcal{A}$ corrupts up to t parties.

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 players and the public information produced by the DKG protocol, outputs the unique value x, even if up to t shares are submitted by faulty players.

3.2 The Insecurity of a Common DKG Protocol

The Joint-Feldman Protocol. Feldman [Fel87] presents a *verifiable secret sharing* (VSS) protocol, denoted by Feldman-VSS, that allows a *trusted* dealer to share a key x among n parties in a way that the above security properties are achieved (with the exception that the protocol assumes the dealer never to be corrupted by the attacker). Based on this protocol, Pedersen [Ped91a] proposes the first distributed solution to this problem, i.e. the first DKG protocol. It specifies the run of n parallel executions of Feldman-VSS as follows. Each player P_i selects a random secret $z_i \in \mathbb{Z}_q$ and shares it among the n players using Feldman-VSS. This defines the set *QUAL* of players that shared their secrets properly. The random secret x is set to be the sum of the properly shared secrets and each player 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 Figure 1 we present a simplified version of the protocol proposed in [Ped91a], which we call Joint-Feldman. By concentrating on the core of the protocol we are able to emphasize the central weakness in its design. We also show that several variants of this core protocol (including the full protocol from [Ped91a] and other modifications [HJKY95, HJJ+97]) are also insecure.

An Attack Against Joint-Feldman. We show how an adversary can influence the distribution of the result of Joint-Feldman to a non-uniform distribution.

It can be seen, from the above description of the protocol 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 player is in *QUAL* or not, even given the fully synchronous communication model, occurs after the adversary has seen the values y_i of all players. The values y_i are made public in Step 1 and the disqualification of players occurs in Steps 2-3. 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. It assumes two faulty players, say P_1 and P_2. In Step 1, P_1 gives players $P_3, ..., P_{t+2}$ shares which are inconsistent with his broadcast values, i.e. they do not pass the test of Step 2. The rest of the players receive consistent shares. Thus, in Step 2 there will be t complaints against P_1, yet t complaints are not sufficient for disqualification. Now, 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 P_1 will do nothing and continue the protocol as written. If α ends with 1 then the adversary forces the disqualification of P_1 in Step 3.

Protocol Joint-Feldman

1. Each player P_i chooses a random polynomial $f_i(z)$ over $\mathbb{Z}_q$ of degree t:

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

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

$$g^{s_{ij}} = \prod_{k=0}^{t} (A_{ik})^{j^k} \bmod p \quad (1)$$

   If the check fails for an index i, P_j broadcasts a *complaint* against P_i.
3. If more than t players complain against a player P_i, that player is clearly faulty and he is disqualified. Otherwise P_i reveals the share s_{ij} matching Eq. 1 for each complaining player 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 players.
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 player 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. An insecure solution for distributed generation of secret keys

This is achieved by asking P_2 to also broadcast a complaint against P_1, which brings the number of complaints to $t+1$. 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.

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 S which receives the value y and needs to "hit" this value. That is, 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 players, say $P_1, ..., P_t$. The simulator has enough information to compute the values $z_1, ..., z_t$ that the adversary has shared in Step 1. Now S needs to commit itself to the values shared by the good players. 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 players whose values will be considered in the final computation (i.e. $QUAL = Q \cup \{t+1, ..., n\}$). Consequently, in Step 1, the simulator S does not know

how to pick the good players' values $y_{t+1}, ..., y_n$ so that $(\prod_{i \in Q} y_i) \cdot (y_{t+1} \cdot ... \cdot y_n) = y \bmod p$, as 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 S has no effective strategy to simulate this computation in polynomial time.

Other Insecure Variants of the Joint-Feldman Protocol. The many variants and extensions of the Joint-Feldman protocol which have appeared in the literature are also insecure. They all fail to achieve the correctness property (C3) and the secrecy requirement as presented in Section 3.1. The variants include: signatures on shares, commitments to y_i, committing encryption on broadcast channel, committing encryption with reconstruction, and "stop, kill and rewind". Due to space limitations, we invite the reader to the on-line appendix to this paper [GJKR99] for the description of these variants and their flaws.

4 The New Protocol

Our solution enjoys the same flavor and simplicity as the Joint-Feldman protocol presented in Figure 1, i.e. each player shares a random value and the random secret is generated by summing up these values.

But we use a different sharing and then introduce methods to extract the public key. We start by running a commitment stage where each player P_i commits to a t-degree polynomial (t is the scheme's threshold) $f_i(z)$ 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) player P_j to depend on the commitment(s) of any set of honest players. Second, for any player 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 players (this may be needed if player P_i misbehaves at a later stage of the protocol). Finally, for each honest player P_i and non-disqualified player 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 verifiable secret sharing (VSS) protocol due to Pedersen [Ped91b], and which we denote by Pedersen-VSS. 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 player misbehaves later in the protocol his value z_i is publicly reconstructed by the honest players). 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 efficiently and securely compute $g^x \bmod p$.

In the next subsection we present the detailed solution and its analysis. But first we expand on Pedersen's VSS protocol.

Pedersen's VSS. As said, we use the protocol Pedersen-VSS introduced in [Ped91b] as a central tool in our solution. For lack of space we do not explicitly describe Pedersen-VSS here, however its description is implicit in step 1 of

Figure 2. We note that this protocol uses, in addition to the parameters p, q, g which are inherent to the DKG problem, an element h in the subgroup of $\mathbb{Z}_p^*$ generated by g. It is assumed that the adversary cannot find the discrete logarithm of h relative to the base g. In section 4.2 we discuss how this value of h can be generated in the context of our DKG solution. 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. [Ped91b] Pedersen-VSS *satisfies the following properties in the presence of an adversary that corrupts at most t parties and which cannot compute $dlog_g h$:*

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

4.1 Secure DKG Protocol

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

Theorem 2. *Protocol* DKG *from Figure 2 is a secure protocol for distributed key generation in discrete-log based cryptosystems, namely, it satisfies the correctness and secrecy requirements of Section 3.1 with threshold t, for any $t < n/2$.*

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

(C1) At the end of Step 2 of the protocol it holds that if $i \in QUAL$ then player P_i has successfully performed the dealing of z_i under Pedersen-VSS. From part 1 of Lemma 1 we know that all honest players 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 $\mathcal{R}$:

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

Protocol DKG

Generating x:

1. Each player 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 $\mathbb{Z}_q$ of degree t:

   $$f_i(z) = a_{i0} + a_{i1}z + \ldots + a_{it}z^t \qquad f'_i(z) = b_{i0} + b_{i1}z + \ldots + b_{it}z^t$$

   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 player P_j.
   (b) Each player P_j verifies the shares he received from the other players. For each $i = 1, \ldots, n$, P_j checks if

   $$g^{s_{ij}}h^{s'_{ij}} = \prod_{k=0}^{t}(C_{ik})^{j^k} \bmod p \tag{2}$$

   If the check fails for an index i, P_j broadcasts a *complaint* against P_i.
   (c) Each player P_i who, as a dealer, received a complaint from player P_j broadcasts the values s_{ij}, s'_{ij} that satisfy Eq. 2.
   (d) Each player marks as *disqualified* any player that either
   - received more than t complaints in Step 1b, or
   - answered to a complaint in Step 1c with values that falsify Eq. 2.
2. Each player then builds the set of non-disqualified players *QUAL*. (We show in the analysis that all honest players 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 player 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 player $i \in QUAL$ exposes $y_i = g^{z_i} \bmod p$ via Feldman VSS:
   (a) Each player P_i, $i \in QUAL$, broadcasts $A_{ik} = g^{a_{ik}} \bmod p$ for $k = 0, ..., t$.
   (b) Each player P_j verifies the values broadcast by the other players in *QUAL*, namely, for each $i \in QUAL$, P_j checks if

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

   If the check fails for an index i, P_j *complains* against P_i by broadcasting the values s_{ij}, s'_{ij} that satisfy Eq. 2 but do not satisfy Eq. 3.
   (c) For players P_i who receive at least one valid complaint, i.e. values which satisfy Eq. 2 and not Eq. 3, the other players 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 players 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 discrete–log based systems

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 4a:

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

where the last equality follows from Eq. 3. Thus the publicly available value g^{x_j} makes it possible to verify the correctness of share x_j at reconstruction time.

(C2) The value y is computed (by the honest players) 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 players. 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 $y = \prod_{i \in QUAL} y_i = \prod_{i \in QUAL} g^{z_i} = g^{\sum_{i \in QUAL} z_i} = g^x$. For parties $i \in QUAL$ against whom a valid complaint has been issued in Step 4b 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 player P_i in Step 4a define a t-degree polynomial $\hat{f}_i(z)$ in $\mathbb{Z}_q$. Since we assume that no valid complaint was issued against P_i then Eq. 3 is satisfied for all honest players, 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 players 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 DKG (since neither the values z_i nor the set $QUAL$ change later). Let P_i be a non-corrupted player; 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 SIM for the DKG protocol in Figure 3. Here we show that the view of the adversary $\mathcal{A}$ that interacts with SIM on input y is the same as the view of $\mathcal{A}$ that interacts with the honest players in a regular run of the protocol that outputs the given y as the public key.

In the description and analysis of the simulator we assume, without loss of generality, that the adversary compromises players $P_1, \ldots, P_{t'}$, where $t' \leq t$. We denote the indices of the players controlled by the adversary by $\mathcal{B} = \{1, \ldots, t'\}$, and the indices of the players controlled by the simulator by $\mathcal{G} = \{t'+1, \ldots, n\}$.

In a regular run of protocol 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 $\mathbb{Z}_q$ (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 the Eqs. (2) and (3) are satisfied for all $j \in \mathcal{B}$.

Algorithm of simulator SIM

We denote by $\mathcal{B}$ the set of players controlled by the adversary, and by $\mathcal{G}$ the set of honest parties (run by the simulator). Wlog, $\mathcal{B} = \{1, \ldots, t'\}$ and $\mathcal{G} = \{t'+1, \ldots, n\}$, $t' \leq t$.

Input: public key y

1. Perform Steps 1a-1d,2 on behalf of the uncorrupted players $P_{t'+1}, \ldots, P_n$ exactly as in protocol DKG. This includes receiving and processing the information sent privately and publicly from corrupted players to honest ones. At the end of Step 2 the following holds:
   - 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 $i \in QUAL$, $k = 0, \ldots, t$.
   - SIM 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 SIM, receive enough consistent shares from the adversary that allow SIM to compute all these parties' polynomials). In particular, SIM 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 $i \in QUAL \setminus \{n\}$, $k = 0, \ldots, t$
   - Set $A^*_{n0} = y \cdot \prod_{i \in (QUAL \setminus \{n\})} (A_{i0})^{-1} \bmod p$
   - 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 $i \in \mathcal{G} \setminus \{n\}$, and A^*_{nk} for $k = 0, \ldots, t$
   (b) Perform for each uncorrupted player the verifications of Eq. 3 on the values A_{ik}, $i \in \mathcal{B}$, broadcast by the players 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 players can publish a valid complaint only against one another.)
   (c) Perform Step 4c 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 DKG

Since here we are interested in runs of 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 players) of polynomials $f_i(z), f'_i(z)$, $i \in \mathcal{G}$, uniformly distributed in the family of t-degree polynomials over $\mathbb{Z}_q$ subject to the condition that

$$\prod_{i \in QUAL} A_{i0} = y \bmod p \ . \tag{4}$$

In other words, this distribution is characterized by the choice of polynomials $f_i(z), f'_i(z)$ for $i \in (\mathcal{G} \setminus \{n\})$ and $f'_n(z)$ as random independent t-degree polynomials over $\mathbb{Z}_q$, and of $f_n(z)$ as a uniformly chosen polynomial from the family of t-degree polynomials over $\mathbb{Z}_q$ that satisfy the constraint $f_n(0) = dlog_g(y) - \sum_{i \in (QUAL \setminus \{n\})} f_i(0) \bmod q$. (This last constraint is necessary and sufficient to guarantee Eq. (4).) Note that, using the notation of values computed by SIM in Step 2 of the simulation, the last constraint can be denoted as $f_n(0) = dlog_g(A^*_{n0})$.

We show that the simulator SIM 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 players interacting with $\mathcal{A}$ in a real run of the protocol, thus 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 SIM 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 players and defined as follows: For $i \in \mathcal{G} \setminus \{n\}$, set f^*_i to f_i and f'^*_i to f'_i. For $i = n$, define f^*_n via the values[1] $f^*_n(0) = dlog_g(A^*_{n0})$ 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 $d = dlog_g(h)$. It can be seen that 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 $A_{ik}, i \in \mathcal{G} \setminus \{n\}$ and A^*_{nk} published by the simulator in Step 2a on behalf of the honest parties (i.e. $A_{ik} = g^{a^*_{ik}}$, $i \in \mathcal{G} \setminus \{n\}$ and $A^*_{nk} = g^{a^*_{nk}}, k = 0, \ldots, t$) corresponding to the players' values in Step 4a of the protocol. Thus, these values pass the verifications of Eq. (2) and (3) as in the real protocol.

It remains to be shown that polynomials f^*_i and f'^*_i belong to the right distribution. Indeed, for $i \in \mathcal{G} \setminus \{n\}$ this is immediate since they are defined identically to f_i and f'_i which are chosen according to the uniform distribution.

[1] Note that in this description we use discrete log values unknown to the simulator; this provides a mathematical description of the output distribution of SIM useful for our analysis but does not require or assume that SIM can compute these values.

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 $dlog_g(A_{n0}^*)$ as required to satisfy Eq. 4. 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 $f_n'^*(z)$.

4.2 Remarks

Efficiency. We point out that our secure protocol does not lose much in efficiency with respect to the previously known insecure Joint-Feldman protocol. Instead of Feldman-VSS, each player 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 player 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.

Generation of h. The public value h needed to run Pedersen's VSS can be easily generated jointly by the players. Indeed it is important that nobody knows the discrete log of h with respect to g. The procedure for generating h consists of a generic distributed coin flipping protocol which generates a random value $r \in \mathbb{Z}_p^*$. To generate a random element in the subgroup generated by g it will be enough to set $h = r^k \bmod p$ where $k = (p-1)/q$. If q^2 does not divide $p-1$ (which is easily checkable) then h is an element in the group generated by g.

4.3 Other Applications of a DKG Protocol

DKG protocols have more applications than just key generation. We sketch here two of these applications where previous flawed DKG protocols were used and for which our solution can serve as a secure plug-in replacement.

Randomizers in ElGamal/DSS Threshold Signatures. Signature schemes based on variants of the ElGamal scheme [ElG85], such as DSS, usually consist of a pair (r, s) where $r = g^k$ for a random value $k \in \mathbb{Z}_q$. Several robust threshold versions of such signature schemes have been proposed in the literature [CMI93, GJKR96, PK96]. In these schemes the public value r and the sharing of the secret value k is jointly generated by the players running a DKG protocol. Clearly, in order for the resulting threshold scheme to be identical to the centralized case, r must be uniformly distributed in the group generated by g. However, each of these papers uses a version of the Joint-Feldman protocol which allows an adversary to bias the distribution of r. Our DKG protocol fixes this problem.

Refresh Phase in Proactive Secret Sharing and Signature Schemes. Proactive secret sharing [HJKY95] and signature schemes [HJJ+97] were introduced to cope with mobile adversaries who may corrupt more than t servers during the lifetime of the secret. In these protocols time is divided into stages,

with an assumption that the adversary may corrupt at most t servers in each stage. However in different stages the adversary can control different players. In order to cope with such adversaries the basic idea of proactive secret sharing is to "refresh" the shares at the beginning of each stage so that they will be independent from shares in previous stages, except for the fact that they interpolate to the same secret. This is achieved by the players jointly creating a random polynomial $f(z)$ of degree t with free term 0 such that each player P_i holds $f(i)$. If the share of player P_i at the previous stage was s_i, the new share will be $s_i + f(i)$. In order to generate $f(z)$ the players run a variation of Joint-Feldman where each player shares value $z_i = 0$. The polynomial $f(z)$ is the sum of the polynomials $f_i(z)$ picked by each player (see Figure 1). It should be clear that the same attack described in Section 3.2 to bias the free term of $f(z)$ can be carried out to bias its any other coefficient. The result is that the polynomial $f(z)$ generated by this refresh phase is not truly random, which implies that shares from different stages are not independent. Our DKG protocol fixes this problem as well.

5 Enhanced Security: Partially Synchronous Model

In the design of distributed cryptographic protocols it is often assumed that the message delivery is fully synchronous (see Section 2). This assumption is unrealistic in many cases where only *partially synchronous* message delivery is provided (e.g. the Internet). By *partially synchronous* communication model we mean that the messages sent on either a point-to-point or the broadcast channel are received by their recipients within some fixed time bound. A failure of a communication channel to deliver a message within this time bound can be treated as a failure of the sending player. While messages arrive in this partially synchronous manner, the protocol as a whole proceeds in synchronized rounds of communication, i.e. the honest players start a given round of a protocol at the same time. To guarantee this round synchronization, and for simplicity of discussion, we assume that the players are equipped with synchronized clocks.

Notice that in a partially synchronous communication model all messages can still be delivered relatively fast, in which case, in every round of communication, the malicious adversary can wait for the messages of the uncorrupted players to arrive, then decide on his computation and communication for that round, and still get his messages delivered to the honest parties on time. Therefore we should always assume the worst case that the adversary speaks last in every communication round. In the cryptographic protocols literature this is also known as a *rushing* adversary.

Clearly the fully synchronous communication model is strictly stronger than the partially synchronous one, thus the previously existing DKG protocols which we recalled in Section 3 remain *insecure* also in this model. In fact, the relaxation of the model allows stronger attacks against many of the Joint-Feldman variants. For example, the adversary could choose the z_i's of the dishonest players dependent on the ones chosen by the honest ones (while in the fully synchronous model

he is restricted to deciding whether the previously decided z_i's of the dishonest players will be "in" or "out" of the computation).

In contrast, the DKG protocol we propose in this paper is *secure* even in this more realistic partially synchronous communication setting. Intuitively, this is because the first stage involves an information-theoretic VSS of the z_i values. Thus the adversary has *no* information about these values and he has to choose the z_i's of the dishonest players in an independent fashion even if he speaks last at each round. When the values $y_i = g^{z_i}$ are revealed, it is too late for the adversary to try to do something as at that point he is committed to the z_i's which are recoverable by the honest players. A formal proof of security of our protocol in this stronger model is identical to the proof presented in Section 4.1. Indeed, it can be easily verified that the proof of security carries over to the partially synchronous communication model basically unchanged.

Extension to Adaptive Adversary. Recently, [CGJ+99] showed a modification of our 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 model the corrupted parties are fixed in advance before the protocol starts). The only modification to our protocol introduced in [CGJ+99] 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 player broadcasts a pair $(A_{i0}, B_{i0}) = (g^{a_{i0}}, h^{b_{i0}})$ s.t. $A_{i0} \cdot B_{i0} = C_{i0} \bmod p$, and proves in zero-knowledge that he knows the discrete logs $DLOG_g(A_{i0})$ and $DLOG_h(B_{i0})$. Proving this ensures that $y_i = g^{z_i}$. If a player fails the proof then his shared value z_i is reconstructed via the Pedersen-VSS reconstruction, as in our DKG protocol.

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 player". Namely, there is at most one player 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 player" 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 player is made, and the simulation continues. It is shown in [CGJ+99] that this brings to the successful end of the simulation in expected polynomial-time.

Acknowledgments. We thank Don Beaver for motivational discussions on this problem.

References

[CGJ+99] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive Security for Threshold Cryptosystems. Mansuscript, 1999.

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

[CMI93] M. Cerecedo, T. Matsumoto, and H. Imai. Efficient and secure multiparty generation of digital signatures based on discrete logarithms. *IEICE Trans. Fundamentals*, E76-A(4):532–545, 1993.

[DF89] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Advances in Cryptology — Crypto '89*, pages 307–315. LNCS No. 435.

[ElG85] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Trans. Info. Theory*, IT 31:469–472, 1985.

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

[FGMY97] Y. Frankel, P. Gemmell, P. Mackenzie, and M. Yung. Optimal resilience proactive public-key cryptosystems. In *Proc. 38th FOCS*, pages 384–393. IEEE, 1997.

[GJKR96] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *Advances in Cryptology — Eurocrypt '96*, pages 354–371. LNCS No. 1070.

[GJKR99] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems `http://www.research.ibm.com/security/dkg.ps`

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

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

[HJKY95] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing, or: How to cope with perpetual leakage. In *Advances in Cryptology — Crypto '95*, pages 339–352. LNCS No. 963.

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

[Ped91a] T. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology — Eurocrypt '91*, pages 522–526. LNCS No. 547.

[Ped91b] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology — Crypto '91*, pages 129–140. LNCS No. 576.

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

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

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

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