One Round Threshold Discrete-Log Key Generation without Private Channels

Pierre-Alain Fouque and Jacques Stern

École Normale Supérieure, Département d'Informatique
45, rue d'Ulm, F-75230 Paris Cedex 05, France
{Pierre-Alain.Fouque,Jacques.Stern}@ens.fr

Abstract. Pedersen designed the first scheme for generating Discrete-Log keys without any trusted dealer in 1991. As this protocol is simple and efficient, it appeared to be very attractive. For a long time, this robust algorithm has been trusted as being secure. However, in 1999, Gennaro *et al.* proved that one of the requirements is not guaranteed : more precisely, the property that the key is uniformly distributed in the key space.
Their main objective was to repair the security flaw without sacrificing on efficiency. As a result, the protocol became secure but somehow unpractical. In particular, the "complaint phase", in which cheaters are thrown out, makes the scheme overly complex and difficult to deal with in practical situations. In order to avoid this phase and other drawbacks such as the initialization phase where private channels have to be created, we present a *one round* scheme which generates a discrete-log key with public channels only. Finally, we show how to improve the efficiency of our algorithm when the number of servers increases.

Key words: Threshold DLK Generation, Publicly Verifiable Encryption, Adaptive and Concurrent Adversary

1 Introduction

In order to design threshold cryptosystems such as signature or public key encryption schemes, the first stage consists in sharing the key generation procedure. Indeed, if a trusted dealer is used in the key generation protocol, the security of the overall distributed scheme depends on a unique server. Key generation protocols are based on random distribution processes. The servers jointly generate a random key such that, at the end of the process, all honest servers have a share of the secret key.

Improvements to the random generation of private keys for public key cryptography usually fall into two areas : the distribution of a secret for discrete-log based cryptosystems and the distribution of RSA keys.

The latter case is partially solved by the nice paper of Boneh and Franklin [4]. However, the protocol does not allow to efficiently share RSA modulus with strong primes and is not robust against cheaters. Following this paper, two articles provide robustness using different techniques. The first one by Frankel *et al.* [11], is based on the same methods as [4] and uses the protocol of Ben-Or, Goldwasser and Widgerson [2] with private channels between each pair of participants. Frankel *et al.* also propose protocols that make the scheme proactive in [11, 10, 12]. In [22] Poupard and Stern present a protocol for two players which avoids private channels. They introduce a new technique simpler and more efficient that does not need to perform many rounds of communication. It is based on a trapdoor version of the discrete logarithm problem.

This kind of protocol is well-suited to small group of participants which, from a practical point of view, is the usual case. Gilboa has followed this method in [14].

Methods for distributing keys for discrete-log cryptosystems have been known for a long time, starting with Feldman and Pedersen papers [9, 20, 21]. However, a flaw in the requirements has been discovered and a first solution as well as a security model for DKG protocols have been defined by Gennaro *et al.* in [13]. The solution has been improved by Canetti *et al.* in [7] to withstand adaptive attacks. In [16], Lysyanskaya and Jarecki have proposed two new models of security for this kind of attacks. The first one dealt also with concurrent adversaries whereas the second presents erasure-free adaptive security with persistently inconsistent players. The schemes are based on Pedersen Verifiable Secret Sharing and consequently use private channels. Only, Jarecki's solution uses public channels but it needs non-committing encryption scheme which makes the protocol less efficient.

Whereas previous solutions to DKG prove security in the information-theoretic model, we use here a computational model as the goal of such protocol is to construct a public key. Therefore, we eliminate the committing values of [16, 7] which are needed to prove the security against adaptive adversaries. To cope with such adversaries, we design a one round protocol.

Following the new approach proposed by Poupard and Stern, the contribution of this paper is to introduce public channels in order to reduce the communication rounds to a unique phase. If we use non-interactive protocol, we can also ignore concurrent and adaptive adversaries as this kind of attackers make no sense in a one round protocol. To achieve a non-interactive protocol, we need primitives such that all servers can decide whether the other servers have correctly performed their tasks and synchronous network to prevent "rushing attacks". Consequently, we need NIZK proofs secure in the random oracle model and public channels. In appendix 8.1, we relax the assumption of synchronous network and present a model where we can easily prevent the "rushing attack" and adaptive adversaries at the price of a particular player.

1.1 Background and Related Work

Pedersen scheme [20] is a non-interactive scheme with broadcast and private channels. The scheme is organized in two phases : in the first stage, the participants select the key while in the second, the key is distributed between servers. In the distributed phase, each server acts as the dealer in a Feldman protocol [9] which uses verifiable secret sharing.

Participant P_i chooses $x_i \in \mathbb{Z}_q$ at random and picks t random numbers $a_{i,k}$ in $\mathbb{Z}_q$. Then, he sets $f_i(X) = \sum_{k=0}^{t} a_{i,k} X^k$ where $a_{i,0} = x_i$. He privately sends a secret share $s_{i,j} = f_i(j) \bmod q$ to participant P_j and broadcasts as public information $y_{i,j} = g^{s_{i,j}} \bmod p$, and $A_{i,k} = g^{a_{i,k}} \bmod p$ for $k = 0, \ldots, t$. These data can be used by all servers to check whether the random, x_i chosen by P_i, has been correctly distributed. Let

$$y_{i,j} = \prod_{k=0}^{t} A_{i,k}^{j^k} \bmod p$$

If $s_{i,j}$ is not the discrete logarithm of $y_{i,j}$, participant P_j broadcasts a complaint against P_i.

The complaints are managed through different strategies. A possible one is the following : if more than t participants complain against server P_i, that server is clearly faulty and is disqualified. Otherwise, P_i reveals the share $s_{i,j}$ for each complaining player P_j. If any of the revealed shares fails the equation $y_{i,j} = g^{s_{i,j}} \bmod p$, P_i is disqualified, otherwise P_i can still be qualified. One can then define QUAL as the set of non-disqualified players. The public key is defined as $y = \prod_{i \in \mathrm{QUAL}} y_i$ where $y_i = A_{i,0} = g^{x_i} \bmod p$.

In the previous scheme, participant P_i chooses a random secret x_i and shares it between the other servers. Consequently, since disqualifications may occur after the distribution phase, the selection phase which aims at unambiguously fixing the public key, is not completed before the beginning of the distribution phase. Hence, an adversary can compute the public key at the end of the distribution phase using the values $A_{i,0}$. Depending on this intended value and on the goal of the adversary, this player can, for example, disqualify some members in order to modify the public key distribution. Gennaro *et al.* describe an attack in which two malicious members can create a bias in the distribution of the last bit of the public key with probability 3/4 rather than 1/2. Their attack relies on the fact that the following scheme uses secret channels between each member such that we cannot know whenever an error appears which party has cheated if both players are corrupted.

To avoid this attack, Gennaro *et al.* duplicate the scheme: in the first part, the honest group is selected and in a second phase, the public value associated to the shared secret is made public. In this case, the qualified group is determined at the end of the first phase. In the first selection phase, each server commits a random value with the unconditional scheme of Pedersen in [21] ; whereas in the distribution phase, players release information enabling everyone to compute the public value. If a player cheats in the second phase, but belongs to QUAL, the other players run an error-correcting algorithm with the values that the cheating player had distributed during the first phase.

The need for two phases comes from the fact that the private channels used hide faulty players. Consequently, after the first phase which uses a symmetric algorithm, we do not know if the cheating player is :

- the sender who has sent a false share within the private channel, or
- the receiver who claims that he has received a bad share.

Therefore, the second phase is needed to solve the “complaints”. However, we can expect that in general no server will be corrupted. Thus this second phase appears to be redundant and useless. Moreover, it is the most time consuming phase of the protocol.

1.2 Our solution

Our approach is focused on simplifying previous protocols. In a real implementation of private channels, an additional previous round must be executed to share the secret key between each pair of servers. This first round is usually put as a requirement for the channel but this phase involves penalties in practical implementations.

Moreover, in private channels, we cannot know whether the faulty player is the sender or the receiver in the first phase. The second phase of [13] is therefore needed to solve the complaints coming from this ambiguity. Hence, if we use a Publicly Verifiable Secret Sharing scheme (PVSS) and Publicly Verifiable Encryption scheme (PVE) [5], we are able to immediately detect whether the sender has sent faulty parts. Then, malicious players are caught and we do not allow them in the group of qualified members. Consequently, our scheme consists of only one phase where each participant shares his random number with a PVSS scheme and transfers the share to the intended party with a PVE scheme in such a way that all parties can verify that the receiver is able to recover his share. As we use only one phase with a PVSS, all users must be able to determine the public key at the end of this stage. Consequently, we use a synchronous network to avoid "rushing attacks", where an adversary waits until all other servers have played before defining its own value. In this scenario, an adversary can choose a public key or at least bias the distribution. Therefore, the release of values $A_{i,0}$ will be made at a fixed time for each server. In 8.1 we show how to avoid s ynchronous network with a new kind of player.

Complexity of the protocol. In our scheme, all users have to verify a lot of proofs. Particularly, if the numbers of participants is ℓ and each of them shares his secret random in ℓ pieces, we have $O(\ell^2)$ shares in the scheme. Moreover, in our scenario, all of these pieces are broadcast and must be verified by all participants. We do not use a "complaint phase" in which a misleading player informs the others that a verification does not work. Hence, the computation complexity is $O(k\ell^2)$, whereas other schemes have a complexity in $O(k\ell)$. However, we believe that it does not tamper practicality since the number of participants is usually limited. Finally, the hidden constant in the O-notation of other protocols makes the comparison a bit meaningless.

Improvement of the complexity. At first glance, our scheme seems to be costly in terms of computations since all servers have to check the shares of the others. However, as noted above in practical situations we have to deal only with few servers. Furthermore, if we want to execute our protocol with more servers, we provide in appendix 8.2 a solution to speed-up the verification phase. Thanks to a fast batch verification, we prove that the computation complexity of our scheme is comparable with previous ones.

1.3 Outline of the paper

In section 2, we define the security model for the distributed key generation of discrete-log keys. In section 3, we recall some cryptographic primitives and present a proof of fairness. In section 4, we describe the scheme and in section 5 the security proof. Finally, we discuss the complexity of our protocol in section 6.

2 The Security Model

2.1 The Network and the Players

Our game includes the following players connected through a synchronous broadcast channel : a set of ℓ servers $P_1, \ldots, P_\ell$ and an adversary who may control up to t servers. A player P_i is considered *good* as long as he has followed the protocol and *faulty* once he has deviated from the protocol.

2.2 Formal Definition

A t-out-of-ℓ *threshold key generation scheme* is a protocol that allows any subset of $t+1$ players out of ℓ to generate the secret key, but disallows the generation if fewer than t players participate to the protocol.

A t-out-of-ℓ threshold key generation is composed by a *key generation algorithm* that takes as input a security parameter k, the number ℓ of generation servers, and the threshold parameter t ; it outputs a public key PK, and a list $SK_1, \ldots, SK_\ell$ of shares of the private key associated to the list $PK_1, \ldots, PK_\ell$ of shares of the public key.

2.3 Security Requirements

The security requirements for a threshold key generation scheme are *correctness* and *secrecy*. We present here the requirements for discrete logarithm public key $PK = y = g^x \bmod p$ and $SK = x$. The SK is shared among the ℓ servers.

The *correctness* property consists of the three followings items.

- All subsets of $t+1$ shares provided by honest players define the same unique secret key x.
- All honest parties have the same value of public key $y = g^x \bmod p$, where x is the unique secret guaranteed by the previous item.
- The value x is uniformly distributed in $\mathbb{Z}_q$, and hence y is uniformly distributed in the subgroup generated by g.

The *secrecy* property means that no information on x can be learned by an adversary beyond what follows from equality $y = g^x \bmod p$. The secrecy condition can be more formally expressed in terms of simulatability. The simulation enables to prove that the attacker $\mathcal{A}$ learns nothing on the random numbers of the uncorrupted servers. More precisely, if $\mathcal{A}$ has knowledge of the t random numbers of the corrupted servers and knows the public value y, a program called simulator $\mathcal{S}$ can be executed in expected polynomial time, so that the view of the adversary during a real run is indistinguishable from the output of the simulator. Hence, the adversary cannot see if the distribution comes from a simulator or from a real run. Consequently, as the simulator does not know the secret information, the output of the simulator cannot be used by the adversary to learn knowledge on the secret numbers of the uncorrupted players.

2.4 The Adversarial Game

To define *correctness and security against a static adversary*, we consider the following game played against such adversary.

A1 The adversary $\mathcal{A}$ has knowledge of the intended output of the distributed discrete-log public key algorithm : the public key y.
A2 The attacker chooses to corrupt t servers. $\mathcal{A}$ learns all their secrets and she actively controls their behavior.
A3 Each participant chooses a random number and shares it using a Publicly Verifiable Secret Sharing scheme among the others.

3 Cryptographic Primitives

3.1 The Paillier cryptosystem

Various cryptosystems based on randomized encryption schemes $E(M)$, which encrypt a message M by raising a basis g to the power M and suitably randomizing the result, have been proposed so far [15, 3, 17–19]. Their security is based on various "residuosity" assumptions and the trapdoor is a hidden subgroup where discrete log computations are feasible. We call those cryptosystems *trapdoor discrete logarithm schemes.* As an important consequence of this encryption technique, those schemes have homomorphic properties that can be informally stated as follows:

$$E(M_1 + M_2) = E(M_1) \times E(M_2) \quad \text{and} \quad E(k \times M) = E(M)^k$$

Paillier has presented three closely related such cryptosystems in [19]. We only recall the first one.

This cryptosystem is based on the properties of the Carmichael lambda function $\lambda(N)$ in $\mathbb{Z}_{N^2}{}^*$. We refer to $\lambda(N)$ as λ. We recall here the main two theorems: for any $w \in \mathbb{Z}_{N^2}{}^*$,

$$w^\lambda = 1 \bmod N, \quad \text{and} \quad w^{N\lambda} = 1 \bmod N^2$$

Key Generation. Let N be an RSA modulus $N = pq$, where p and q are prime integers. Let G be an integer whose order is a large multiple of N modulo N^2. The public key is $PK = (N, G)$ and the secret key is $SK = \lambda$.

Encryption. To encrypt a message $M \in \mathbb{Z}_N$, randomly choose u in $\mathbb{Z}_N{}^*$ and compute the ciphertext $c = G^M u^N \bmod N^2$.

Decryption. To decrypt c, compute $M = \dfrac{L(c^\lambda \bmod N^2)}{L(G^\lambda \bmod N^2)} \bmod N$ where the L-function takes in input elements from the set $\mathcal{S}_N = \{x < N^2 | x = 1 \bmod N\}$ and computes $L(x) = \frac{x-1}{N}$.

The integers $c^\lambda \bmod N^2$ and $G^\lambda \bmod N^2$ are equal to 1 when they are raised to the power N so they are N^{th} roots of unity. Furthermore, such roots are of the form $(1 + N)^\beta = 1 + \beta N \bmod N^2$. Consequently, the L-function allows to compute such values $\beta \bmod N$ and $L((G^M)^\lambda \bmod N^2) = M \times L(G^\lambda \bmod N^2) \bmod N$.

The Residuosity Class Problem. Assume the order of G is a multiple of N. A number v is said to be a N^{th} residue modulo N^2 if there exists a number $u \in \mathbb{Z}_{N^2}{}^*$ such that $v = u^N \bmod N^2$. For $w \in \mathbb{Z}_{N^2}{}^*$, we call N^{th} residuosity class of w with respect to G the unique integer $r \in \mathbb{Z}_N$ for which there exists $u \in \mathbb{Z}_N{}^*$ such that $G^r u^N = w \bmod N^2$.

The Composite Residuosity Class Problem is defined to be the computational problem of computing the class of a random element in $\mathbb{Z}_{N^2}{}^*$.

Security. This problem that exactly consists in inverting the cryptosystem, is believed to be intractable. The semantic security is based on the difficulty to distinguish N^{th} residues modulo N^2. We refer to [19] for details.

3.2 A proof of fairness

In this section, we present a proof in the style of [23] which enables to prove that decryption of $Y = G^x u^N \bmod N^2$ in base G allows to recover the discrete logarithm of $y = g^x \bmod p$ in base g, where g is of order a prime q in $\mathbb{Z}_p{}^*$.

We describe a non-interactive statistical zero-knowledge proof of the existence of two small numbers σ and τ so that $|\sigma| < A$ and $|\tau| < B$ which verify that $G^\sigma Y^{-\tau}$ is a N^{th} residue for $\sigma\tau^{-1} = \log_g y$. We prove the security of the proof in the random oracle model.

Description of the proof. Let $x \in [0, S[$ be the secret value, and A, B and S three integers such that $|A| \geq |B| \cdot |S| + k'$ where k' is a security parameter. The value B is the output length of a hash function H.

The prover chooses a random r in $[0, A[$ and a random $s \in \mathbb{Z}_N{}^*$. Then, he computes $t = (g^r \bmod p, G^r s^N \bmod N^2)$. Let e be the hash value $H(g, G, y, Y, g^r \bmod p, G^r s^N \bmod N^2)$. Next, the prover computes $z = r + ex$ and $w = su^e \bmod N$. If $z \notin [0, A[$, the prover restarts with another random values r and s until $z \in [0, A[$. The proof is the triple $(e, z, w) \in [0, B[\times[0, A[\times[0, N[$. It is checked by the equations $e = H(g, G, y, Y, g^z y^{-e} \bmod p, G^z w^N Y^{-e} \bmod N^2)$, $z \in [0, A[$, and $y^q = 1 \bmod p$.

Completeness. *The execution between a prover who knows the secret x and a verifier is successful with overwhelming probability if $SB/A < 1/2^{k'}$ is negligible.*

Proof: The verifier has access to (e, z, w) where $z = r + ex < A$, $w = su^e \bmod N$, and $e = H(g, G, y, Y, g^r \bmod p, G^r s^N \bmod N^2)$. He can check whether $z < A$, $g^r = g^{r+ex}(g^x)^{-e} = g^z y^{-e} \bmod p$, and $G^r s^N = G^{r+ex}(su^e)^N (G^x u^N)^{-e} = G^z w^N Y^{-e} \bmod N^2$.

If the prover follows the protocol, the proof fails only if $z \geq A$. The probability of failure of such an event taken over all possible choice of r is smaller than SB/A. Consequently, the execution of the protocol is successful with probability greater than $1 - \frac{SB}{A}$. Thus, if SB/A is negligible, the probability of success is overwhelming. □

Soundness. *If the verifier accepts the proof, with probability $\geq 1/B + \epsilon$ where ϵ is a non-negligible quantity, then using the prover as a "black-box" it is possible to compute σ and τ such that $|\sigma| < A$ and $|\tau| < B$ such that $\sigma\tau^{-1} = x \bmod q$, $g^x = y \bmod p$ and $G^\sigma Y^\tau$ is a N^{th} residue modulo N^2.*

Proof: For a given t, if a prover can find two triples (e, z, w) and (e', z', w') which pass the proof with non-negligible probability, he can obtain the following equalities : $G^{z-z'}(w/w')^{Ne} = Y^{e-e'} \bmod N^2$ and $g^{z-z'} = y^{e-e'} \bmod p$.

Hence, if we note $\sigma = z - z'$ and $\tau = e - e'$:

$$G^{\sigma}(w/w')^{Ne} = Y^{\tau} \bmod N^2 \quad \text{and} \quad g^{\sigma} = y^{\tau} \bmod p \tag{1}$$

and $|\sigma| < A$ and $0 < |\tau| < B$.

As $y^q = 1 \bmod p$ and as there is a unique subgroup of order q in $\mathbb{Z}_p{}^*$, the value y is in $\langle g \rangle$. Hence, from the second equality, we deduce that $\sigma\tau^{-1} \bmod q$ is the discrete log of y.

We note $d = \gcd(\sigma, \tau)$. As q is a prime number, we get $\sigma/d = \tau/d \times x \bmod q$. Let $\sigma_0 = \sigma/d$, $\tau_0 = \tau/d$. Knowledge of (σ_0, τ_0) enables to compute the secret $x = \sigma_0{\tau_0}^{-1} \bmod q$.

Let $\tilde{x}$ be the result of the decryption of Y. If $g^{\tilde{x}} = y \bmod p$, we are done. Otherwise, we search the values σ_0 and τ_0 where $\sigma_0 = \sigma/d$, $\tau_0 = \tau/d$ and $d = \gcd(\sigma, \tau)$ to find x. In [23], Poupard and Stern describe how to find σ_0 and τ_0 from $\tilde{x}$ and N provided that the proof is correct. They show that the smallest vector of the lattice of dimension two where a basis is $((N, 0), (\tilde{x}, 1))$ corresponds to the vector (σ_0, τ_0) whenever $N \geq 2\sqrt{2}AB$. Hence, as the dimension of the lattice is two, Gauss algorithm can be used to efficiently recover the smallest vector in $O(\log N)$.

Consequently, if the proof is well-formed, participant P_i can always recover the intended share x which matches $y = g^x \bmod p$. This fairness property is useful to guarantee that the receiver will receive the correct data. □

Zero-Knowledge. *This proof is a non-interactive statistical zero-knowledge proof.*

Proof: We can construct a simulator that simulates the adversary's view without knowing the value x in the random oracle model. When an uncorrupted player is supposed to generate a proof for a given y, Y, the simulator chooses $e \in [0, B[$, $z \in [0, A[$ and $w \in \mathbb{Z}_N{}^*$ at random, and defines the value of the random oracle at $(g, G, y, Y, g^z y^{-e} \bmod p, G^z w^N Y^{-e} \bmod N^2)$ to be e. With overwhelming probability, the simulator has not yet defined the random oracle at this point. The proof is just (z, w, e). It is straightforward to verify that the distribution produced by this simulator is statistically close to perfect provided that BS/A is negligible. □

4 The scenario

Each server has to verify all proofs broadcast by other players and to select the qualified group of servers. All players are considered as probabilistic polynomial time Turing machines. In the initialization stage, each participant performs the key generation algorithm of Paillier's cryptosystem. For $i = 1$ to ℓ, the public keys $PK_i = (G_i, N_i)$ are published and the server P_i secretly stores SK_i. The value N_i is a RSA modulus, G_i is an element in $\mathbb{Z}_{N_i^2}{}^*$ of order a multiple of N_i and $SK_i = \lambda(N_i)$.

We consider the following scenario :

- Participant P_i generates a random $s_{i,0}$, sets $a_{i,0} = s_{i,0}$ and chooses $a_{i,k}$ at random from $\mathbb{Z}_q$ for $1 \leq k \leq t$. The numbers $a_{i,0}, \ldots, a_{i,t}$ define the polynomial $f_i(X) = \sum_{k=0}^{t} a_{i,k} X^k \in \mathbb{Z}_q[X]$. Then, he computes $s_{i,j} = f_i(j) \bmod q$. He broadcasts : for $k = 0, \ldots, t$, $A_{i,k} = g^{a_{i,k}} \bmod p$ and $y_{i,j} = g^{s_{i,j}} \bmod p$, $Y_{i,j} = G_j^{s_{i,j}} u_{i,j}^{N_i} \bmod N_j^2$, and a proof $(e_{i,j}, w_{i,j}, z_{i,j})$.
- Then, for each $1 \leq i, j \leq \ell$, the servers verify that :

$$\prod_{k=0}^{t} A_{i,k}^{j^k} = \prod_{k=0}^{t} g^{a_{i,k} j^k} = g^{\sum_{k=0}^{t} a_{i,k} j^k} = g^{f_i(j)} \bmod p$$

  and check whether $g^{f_i(j)} \bmod p$ is equal to $y_{i,j}$ in order to verify that the distribution is correct. The servers also verify the proofs $(e_{i,j}, w_{i,j}, z_{i,j})$ and if $y_{i,j}^q = 1 \bmod p$ for $1 \leq i, j \leq \ell$.
- The set QUAL of qualified servers is defined from the players who have correctly played. The others are disqualified.
- Participant P_j decrypts $Y_{i,j}$ and obtains $s_{i,j}$ for $1 \leq i \leq \ell$. He stores the parts $s_{i,j}$ for $i \in \mathrm{QUAL}$ and computes the public key as $\prod_{i \in \mathrm{QUAL}} A_{i,0} = g^{f(0)} \bmod p$ if we note $f(X) = \sum_{i \in \mathrm{QUAL}} f_i(X)$. The share of the key obtained by participant P_j is equal to

$$\sum_{i \in \mathrm{QUAL}} s_{i,j} = f(j) \bmod q$$

  The secret key s is shared in polynomial form with $f(j) \bmod q$ and in additive form with $x_j \bmod q$ between all participants belonging to the set QUAL.

5 Security Proof

In this section, we prove the security of the scheme following the security model defined in section 2.4. We have to ensure the correctness and the secrecy of the scheme.

Correctness means that all players obtain the same key at the end of the protocol, that $t+1$ correct shares allow to recover the secret key and that the secret value is uniformly distributed in the subgroup generated by g modulo p.

Secrecy means that no information on x can be learned by the adversary except what follows from equation $y = g^x \bmod p$.

Theorem 1 *The sharing scheme is correct against adversaries.*

Proof: Let us assume the existence of an adversary $\mathcal{A}$ able to break t servers.

Correctness. It is clear that, at the end of the protocol, each server obtains the same public key because each honest server has received the same information and deduced the same set QUAL.

At the end of the protocol, the secret value is shared in polynomial form such that any $t+1$ correct shares enable to interpolate the polynomial f of degree t whose constant coefficient represent the secret key s.

Finally, the public key y is uniformly distributed in the subgroup $\langle g \rangle$ because if one of the honest server is not disqualified and has selected his additive share x_i at random, the secret $\sum_{i \in \mathrm{QUAL}} x_i \bmod q$ is randomized uniformly in $\mathbb{Z}_q$. Therefore, the value y is uniformly randomized in $\langle g \rangle$. $\square$

Theorem 2 *Under the decisional composite residuosity assumption and in the random oracle model, the sharing scheme is secure against static adversaries.*

Proof: **Secrecy.** We describe a simulator $\mathcal{S}$ which takes as input an element $y \in \mathbb{Z}_p{}^*$ in the subgroup generated by g and produces an output distribution which is polynomially indistinguishable from $\mathcal{A}$'s view of a run of the protocol that ends with y as its public key output.

Let $\mathcal{A}$ the adversary who knows y in phase **A1** and corrupts t servers at the beginning of the protocol in phase **A2**. $\mathcal{A}$ learns all their secrets and she actively controls their behavior.

Here, we can take advantage of the synchronized network. We have to simulate the distribution of all servers. However, when we simulate the run of the protocol, the synchronization is not needed and we can wait until all malicious servers play. This allows us to determine the public value y_i^* of a specific good server P_i^* such that we do not know its internal state.

Hence, each server P_i, except P_i^*, chooses at random $x_i \bmod q$ and t other values $a_{i,k}$ for $k = 1, \ldots, t$. He sets $f_i(X) = \sum_{k=0}^{t} a_{i,k} X^k \in \mathbb{Z}_q[X]$. Then, he computes $A_{i,k} = g^{a_{i,k}} \bmod p$ for $k = 0, \ldots, t$ and calculates $y_{i,j} = g^{f_i(j)} \bmod p$ and $Y_{i,j} = G_j^{f_i(j)} u_{i,j}^{N_j} \bmod N_j^2$. The distribution of these values and the distribution of those of the real protocol are equal.

Now, we have to simulate the distribution of player P_i^*. If we want the ending value to be y, let $y_i^* = A_{i,0}^* = y \cdot \prod_{i \in \mathrm{QUAL} \setminus \{P_i^*\}} (y_i)^{-1} \bmod p$, as the set QUAL is defined at the end of the synchronization. We choose at random t values $f_i^*(i_j)$ for the t corrupted servers $\{i_1, \ldots, i_t\}$ and send these values to these servers. With the Lagrange interpolation formula, we can compute the public values $y_{i,j}^*$ of all other shares as :

$$y_{i,j}^* = g^{f_i^*(j)} = (y_i^*)^{\lambda_{j,0}^S} \cdot \prod_{j=1}^{t} g^{\lambda_{j,i_j}^S f_i^*(i_j)}$$

where $\lambda_{i,j}^S = \prod_{j' \in S \setminus \{j\}} \frac{i-j'}{j-j'}$ and $S = \{0, i_1, \ldots, i_t\}$.

Here we use an assumption which seems weaker than the assumption used in order to prove the semantic security of the Paillier cryptosystem. In fact, we have $g^x \bmod p$ and $G^x u^N \bmod N^2$ whereas the semantic security has to decide whether the value $G^x u^N \bmod N^2$ encrypts x or not. To simulate $Y_{i,j}^*$, we can therefore choose at random $x_{i,j} \in \mathbb{Z}_{N_j}$ and $u_j \in \mathbb{Z}_{N_j}{}^*$, and set $Y_{i,j}^* = G_j^{x_{i,j}} u_j^{N_j} \bmod N_j^2$.

Player P_i^* has an inconsistent internal state because he does not know the discrete-log of y_i in basis g modulo p. However, this player will not be attacked because in this model all corrupted servers are chosen at the beginning of the game A.

Finally, in the simulation, the distribution produced by the simulator is statistically close to perfect. In the random oracle model, where the simulator has a full control of the values returned by the hash function H, we define the value of H at $(g, G, y, Y, g^z y^{-e}, G^z w^N Y^{-e})$ to be e. With overwhelming probability, the simulator has not yet defined the random oracle at this point so the adversary $\mathcal{A}$ cannot detect the twist. $\square$

6 On the complexity of the protocol

All servers must perform $\ell \times (\ell - 1)$ verifications of the form $y_{i,j} = \prod_{k=0}^{t} A_{i,k}^{j^k} \bmod p$ for $1 \leq i \leq \ell$ and for $1 \leq j \leq \ell$ except for the ℓ shares generated by themselves. Finally, each server must decrypt its part of the secret x.

In this case, P_i has computed the $t+1$ values $A_{i,k} = g^{a_{i,k}} \bmod p$ and the 3ℓ values $s_{i,j} = f_i(j)$, $y_{i,j} = g^{s_{i,j}} \bmod p$, $Y_{i,j} = G_j^{s_{i,j}} u_{i,j}^{N_j} \bmod N_j^2$.

For the proofs, each server P_j has to check whether $e_{i,j} = H(g, G_j, y_{i,j}, Y_{i,j}, g^{z_{i,j}} y^{-e_{i,j}} \bmod p, G_j^{z_{i,j}} w_{i,j}^{N_j} Y_{i,j}^{-e_{i,j}} \bmod N_j^2)$ and $y_{i,j}^q = 1 \bmod p$ for each $1 \leq j \leq \ell$ and for each $1 \leq i \leq \ell$ except for the proofs generated by P_i.

In this case, P_i has to compute the proofs $(e_{i,j}, z_{i,j}, w_{i,j})$ where the heavy calculation is to compute $w_{i,j} = s_{i,j} u_{i,j}^{e_{i,j}} \bmod N_j$.

Finally, each server decrypts its own part x_j of the common secret x. To make this operation in an efficient way, the server P_j computes the product $Y_j = \prod_i Y_{i,j} = G_j^{\sum_i x_{i,j}} \bmod N_j^2$ using $\ell - 1$ multiplications and performs a single decryption on Y_j to recover $x_j \bmod N_j$. This operation consists in a single exponentiation as seen in 3.1. As $x_j = \sum_i x_{i,j}$ is upper bounded by $\ell^{\ell+2} \times q$, and $\ell \times \log_2(q\ell) < \log_2(N_j)$ for all j, the share $x_j \bmod N_j$ is equal to x_j.

The complexity of our scheme is in $O(k\ell^2)$ modular exponentiations where ℓ is the number of servers and k a security parameter whereas the complexity of [13, 7] is in $O(k\ell)$. However, for a small number of participants, our protocol is more efficient. In an appendix 8.2, we provide an improvement of the computation cost when ℓ becomes large and show that the complexity is of the same order as previous schemes.

7 Conclusion

In this paper, we have proposed a threshold discrete-log key generation scheme. We have described a distributed key generation with public channels using one round of communication. Since the communication has been reduced, the computational cost of the scheme increases, but if the number of servers is limited, the overhead is not significant.

Our approach tends to simplify previous work in an area where recent works have resulted in making schemes complex. Moreover, a second phase is not necessary if we use public channel instead of private channel as was done before. Our protocol is well-suited for small groups of servers. It runs in one round and does not require interaction between servers.

References

1. M. Bellare, J. A. Garay, and T. Rabin. Fast Batch Verification for Modular Exponentiation and Digital Signatures. In *Eurocrypt '98*, LNCS 1403, pages 236–250. Springer-Verlag, 1998. Available at `http://www-cse.ucsd.edu/users/mihir/`.
2. M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computing. In *Proceedings of the 20th STOC*, ACM, pages 1–10, 1988.
3. J. Benaloh. *Verifiable Secret-Ballot Elections.* PhD thesis, Yale University, 1987.
4. D. Boneh and M. Franklin. Efficient Generation of Shared RSA Keys. In *Crypto '97*, LNCS 1294, pages 425–439. Springer-Verlag, 1997.

5. J. Camenisch and I. Damgård. Verifiable Encryption and Applications to Group Signatures and Signature Sharing. Available at `http://philby.ucsd.edu/cryptolib/1999/99-08.html`, march 1999.
6. R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. In *Journal of Cryptology*, Volume 13, pages 143–202. Springer-Verlag, 2000.
7. C. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive Security for Threshold Cryptosystems. In *Crypto '99*, LNCS 1666, pages 98–115. Springer-Verlag, 1999.
8. Y. Dodis and S. Micali. Parallel Reducibility for Information-Theretically Secure Computation. In *Crypto '00*, LNCS 1880, pages 74–92. Springer-Verlag, 2000.
9. P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th annual Symposium on the Foundations of Computer Science*. IEEE, 1987.
10. Y. Frankel, P. Gemmel, Ph. MacKenzie, and M. Yung. Optimal-Resilience Proactive Public-Key Cryptosystems. In *Proc. 38th FOCS*, pages 384–393. IEEE, 1997.
11. Y. Frankel, P. Gemmel, Ph. MacKenzie, and M. Yung. Proactive RSA. In *Crypto '97*, LNCS 1294, pages 440–454. Springer-Verlag, 1997.
12. Y. Frankel, P. MacKenzie, and M. Yung. Adaptively-Secure Optimal-Resilience Proactive RSA. In *Asiacrypt '99*, LNCS. Springer-Verlag, 1999.
13. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Cryptosystems. In *Eurocrypt '99*, LNCS 1592, pages 295–310. Springer-Verlag, 1999.
14. N. Gilboa. Two Party RSA Key Generation. In *Crypto '99*, LNCS 1666, pages 116–129. Springer-Verlag, 1999.
15. S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
16. S. Jarecki and A. Lysyanskaya. Adaptively Secure Threshold Cryptography : Introducing Concurrency, Removing Erasures. In *Eurocrypt' 00*, LNCS 1807, pages 221–242. Springer-Verlag, 2000.
17. D. Naccache and J. Stern. A New Cryptosystem based on Higher Residues. In *Proc. of the 5th CCS*, pages 59–66. ACM press, 1998.
18. T. Okamoto and S. Uchiyama. A New Public Key Cryptosystem as Secure as Factoring. In *Eurocrypt '98*, LNCS 1403, pages 308–318. Springer-Verlag, 1998.
19. P. Paillier. Public-Key Cryptosystems Based on Discrete Logarithms Residues. In *Eurocrypt '99*, LNCS 1592. Springer-Verlag, 1999.
20. T.P. Pedersen. A Threshold Cryptosystem without a Trusted Party. In *Eurocrypt'91*, LNCS 547, pages 522–526. Springer-Verlag, 1991.
21. T.P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Crypto'91*, LNCS 576, pages 129–140. Springer-Verlag, 1991.
22. G. Poupard and J. Stern. Generation of Shared RSA Keys by Two Parties. In *Asiacrypt '98*, LNCS 1514, pages 11–24. Springer-Verlag, 1998.
23. G. Poupard and J. Stern. Fair Encryption of RSA Keys. In *Proceedings of Eurocrypt 2000*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

8 Appendix

8.1 Asynchronous Network

In some cases the synchronous network seems to be a strong requirement. We use such network to cope with "rushing attacks". A simple solution is to force a Incorruptible Third Party (ITP) to play at the end. We call this third party "incorruptible" since we do not require a "trusted" party, but only honest. At the beginning, this server picks a random value a in $\mathbb{Z}_q$ and commits $H(a)$ using the hash function H. The servers play and compute $y' = g^{x'}$. At the end, the ITP releases a to all players. The secret value is $x = x' + a \bmod q$. Each server can compute its share of the secret as $x_i = {x'}_i + a \bmod q$. This is due to the Lagrange interpolation since the sum of the Lagrange coefficients is equal to 1. Indeed, if S is a subset of cardinality $t+1$,

$$f(0) = \sum_{i \in S} \lambda_{i,0}^{S} f(i) \bmod q$$

Therefore, if we share the constant polynomial $c(x) = 1$ for all x, the value in 0 is always 1 and we obtain $1 = \sum_{i \in S} \lambda_{i,0}^{S} 1$. Consequently, if we write $f(i) = x'_i$,

$$f(0) + a = (\sum_{i \in S} \lambda_{i,0}^{S} f(i)) + a = \sum_{i \in S} \lambda_{i,0}^{S} (f(i) + a) \bmod q$$

Note 1. In this model, simulations against adaptive adversaries are easy since we can fix the value of the ITP in the random oracle model. We can see it as the "persistently inconsistent player" of [16].
Note 2. Dodis and Micali in [8] proved that a class of Secure Function Evaluation (SFE) remains secure if we compose simple and secure protocols in sequence or in parallel in the information-theoretic model. They use two models of parallel reducibility (or parallel composition of protocols) that they called concurrent reducibility and synchronous reducibility. The *Concurrent reducibility* applies when the order of sub-protocol calls is not important whereas the *Synchronous reducibility* applies when the sub-protocols must be executed simultaneously. Their results [8] hold only in the "information-theoretic model" where we allow private channels and not in the "computational model" that we need in our protocol. In [6], Canetti proved the same results in the computational model but only for concurrent reducibility and not for synchronous reducibility. Therefore, the asynchronous network scheme that we propose in this appendix can be also proved using Canetti's theorem.

8.2 Improvement of the complexity

When the number of servers is relatively small, our protocol is practical. However, it can be unpractical when the number ℓ of servers becomes larger. Here, we provide methods to reduce the computation cost in this situation. We present the computation cost in term of multiplications and we show that asymptotically our protocol has the same order of magnitude than others, i.e. $O(\ell^3 \log(\ell))$ multiplications. This is achieved by reducing all verifications of our protocol to three computations which are used in all protocols. This last calculation represents the heavy part of the computation cost and cannot be avoided in the verifiable interpolation phase.

Since bad players only appear in rare situations, we aim to design efficient protocols when all players are honest. However, we must be able to detect whenever bad players try to cheat and therefore, we require fast detection of active malicious servers. When a malicious server is detected, we need to carry out the protocol of section 6 or to reboot the system as our protocol is state-free.

The first remark is that we cannot avoid the complexity factor $O(\ell^2 k)$, where k is the bit-length of $|p|$ or $|N_j|/2$, from the communication point of view, since in one round, all servers must be able to test whether other servers have correctly played. But, as today's networks have large bandwidth and high speed performance, the bottleneck is not the network communication but the computation load. Consequently, our main objective is to decrease the computation complexity for detecting bad servers.

It is straightforward to see that the larger complexity factor comes from the verifications of $g^{f_i(j)} = \prod_{k=0}^{t} A_{i,k}^{j^k} \bmod p$. We have to check ℓ^2 such equations while the previous schemes only require ℓ. Here we reduce the computation of each player to 3ℓ such equations.

Batch verification. Bellare *et al.* in [1] describe algorithms to perform fast batch verification for modular exponentiation and digital signatures. In this paper, they present techniques to test whether many instantiations $(x_i, y_i)_{i=1}^{n}$ satisfy the equations $g^{x_i} = y_i \bmod p$. The naive method requires n exponentiations. However, if we use probabilistic batch tests, the sequence of modular exponentiations can be computed significantly faster than the naive re-computation method.

They also describe an efficient algorithm to compute $\prod_{i=1}^{n} a_i^{b_i}$ where the cost is $k + nk/2$ modular multiplications if we note k the greatest bit-length of the elements b_i ($b_i = b_i[k] \ldots b_i[1]$). This number is strictly less than n exponentiations followed by $n-1$ multiplications where the cost of a single exponentiation a^b can be estimated as $3k/2$ multiplications if k is the bit-length of b. This algorithm is hereafter called FastMult.

```
Algorithm FastMult((a_1, b_1), ... , (a_n, b_n))
        a := 1;
        for j = k downto 1 do
                for i = 1 to n do if b_i[j] = 1 then a := a.a_i;
                a := a^2;
return a
```

This algorithm does k multiplications in the outer loop and $nk/2$ multiplications on the average in the inner loop. Hence, for computing y we get a total of $k + nk/2$ multiplications.

Finally, they also provide a batch verification of the following form: given a set of points, determine whether there exists a polynomial of a certain degree, which passes through all these points. More formally, let $S = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ denote a set of points. We define the relation $\mathrm{DEG}_{\mathcal{F},t,(\beta_1,\beta_2,\ldots,\beta_m)}(S) = 1$ iff there exists a polynomial $f(x)$ such that the degree of $f(x)$ is at most t, and for all $i \in \{1, \ldots, m\}$, $f(\beta_i) = \alpha_i$, assuming that all the computations are carried out in the finite field $\mathcal{F}$. Let the batch instance of this problem be $S_1, \ldots, S_n$, where $S_i = (\alpha_{i,1}, \ldots, \alpha_{i,m})$. The batch instance is correct if $\mathrm{DEG}_{\mathcal{F},t,(\beta_1,\ldots,\beta_m)}(S_i) = 1$ for all $i = 1, \ldots, n$; incorrect otherwise. This test is called Random Linear Combination Test.

Random Linear Combination Test. This algorithm takes as input n sets $S_1, \ldots, S_n$ where $S_i = (\alpha_{i,1}, \ldots, \alpha_{i,m})$; $\beta_1, \ldots, \beta_m$, security parameter k, and a value t and checks whether for all $i \in \{1, \ldots, n\}$ there exists a polynomial $f_i(x)$ such that $deg(f_i) \leq t$ and $f_i(\beta_1) = \alpha_{i,1}, \ldots, f_i(\beta_m) = \alpha_{i,m}$.

The algorithm works as follows:

1. Pick $r \in_R \mathcal{F}$
2. Compute $\gamma_i = r^n \alpha_{n,i} + \ldots + r\alpha_{1,i}$. This can be efficiently computed with the Horner algorithm.

3. If $\mathrm{DEG}_{\mathcal{F},t,(\beta_1,\dots,\beta_m)}(\gamma_1,\dots,\gamma_m)=1$, then output “correct”, else output “incorrect”.

NOTATION If $f_i(x)=a_m x^m+\ldots+a_0$, where $a_m\neq 0$, we denote by $f_i(x)|^{t+1}$ the polynomial $a_m x^m+\ldots+a_{t+1}x^{t+1}$. Consequently, for $m\leq t$, $f_i(x)|^{t+1}$ must be equal to 0.

The polynomial $F(x)=\sum_{i=1}^{n} r^i f_i(x)$ is of degree at most t, and, therefore, it holds that $\sum_{i=1}^{n} r^i f_i(x)|^{t+1}$ must be equal to 0. This is an equation of degree n in the unknown r and hence has at most n roots. Therefore, to output "correct" when in fact the instance is incorrect, r must be one of the roots of the equation. This algorithm fails with probability at most $\frac{n}{|\mathcal{F}|}$, which is a negligible quantity. The running time of this algorithm is $O(nm)$ while the naive method which consists in computing the polynomial interpolation with $t+1$ points and checking for each of them requires $O(m^2 n)$ multiplications.

Application to our situation. First of all, we note that the computation cost of checking whether the ℓ^2 values $y_{i,j}$ encrypt $g^{f_i(j)}$, require ℓ^2 exponentiations. In fact, we do not need to exactly test whether these equations hold but rather whether a server send false shares. Obviously, each server must verify its own shares but can only check whether the others are correct with high probability.

To this end, we can use the RANDOM LINEAR COMBINATION TEST. We need to run this algorithm “in the exponents” and the proof follows from the fact that g is a primitive element in the subgroup of $\mathbb{Z}_p{}^*$ of order q. In our situation, we have values $\alpha_{i,j}$ correspond to $y_{i,j}$ and values β_i to i.

The algorithm works as follows:

1. Pick $r\in_R\mathbb{Z}_q$
2. Compute $\gamma_i=\alpha_{\ell,i}^{r^\ell}\times\ldots\times\alpha_{1,i}^{r}$. This can be efficiently computed from the values $y_{i,1},\dots,y_{i,\ell}$ using ℓ times the FastMult algorithm.
3. If $\mathrm{DEG}_{\mathbb{Z}_q,t,(1,\dots,\ell)}(\gamma_1,\dots,\gamma_\ell)=1$, then output “correct”, else output “incorrect”.

The DEG consists in checking whether for all $j=1,\dots,\ell$, $g^{F(j)}=\prod_{k=0}^{t} A_{F,k}^{j^k}$ is equal to γ_j. The values $A_{F,k}$ correspond to the coefficients of the polynomial F and are equal to $g^{\sum_{i=1}^{\ell} a_{i,k}r^i}=\prod_{i=1}^{\ell} A_{i,k}^{r^i}$. In fact, γ_j is equal to $y_{\ell,j}^{r^\ell}\times\ldots\times y_{1,j}^{r}=g^{f_\ell(j)r^\ell+\ldots+f_1(j)r}=g^{\sum_{i=1}^{\ell} r^i f_i(j)}=g^{F(j)} \bmod p$.

Consequently, we have to compute the ℓ values γ_j, the $t+1$ values $A_{F,k}$ and the ℓ relations DEG.

This algorithm fails with probability at most $\frac{\ell}{q}$ which is a negligible quantity.

As usual, we estimate t as $\ell/2$ and accordingly, the complexity of the scheme in the number of multiplications is:

1. To compute the values γ_j, we need ℓ times calls to the FastMult algorithm for ℓ products of powers where the size of the exponents are in $\ell|q|$; hence, $\ell\times[\ell|q|+\frac{\ell}{2}\ell|q|]=O(\ell^3|q|)$.
2. To compute the coefficients of $g^{F(x)}$, we need $(t+1)$ calls to the FastMult algorithm for ℓ products of powers where the size of the exponents are in $|q|\ell$; hence, $(t+1)\times[|q|\ell+\frac{\ell^2}{2}|q|]=O(\ell^3|q|)$.

3. To check the relation DEG, we need ℓ calls to the FastMult algorithm for $(t+1)$ products of powers where the size of the exponents are in $t\log(\ell)$; hence $\ell \times [t\log(\ell) + \frac{t(t+1)}{2}\log(\ell)] = O(\ell^3\log(\ell))$.

This algorithm requires $O(\ell^3|q| + \ell^3\log(\ell))$ multiplications.

The previously proposed algorithms cannot used this trick as they have only one value in all set S_i. Therefore, each server j has to perform ℓ calls to the FastMult algorithm to verify whether $y_j = \prod_{k=0}^{t} A_{i,k}^{j^k}$ for $i = 1$ to ℓ. This leads to ℓ times the $t+1$ products of powers where the size of the exponents are in $t\log(\ell)$; hence, $\ell[t\log(\ell) + \frac{t(t+1)}{2}\log(\ell)] = O(\ell^3\log(\ell))$. If we have used this method whereas the Random Linear Combination Test "in the exponents", the complexity will have to be $O(\ell^4\log(\ell))$.

In general, $|q| = 160$ and if we take $\ell = 32 = 2^5$, $\ell\log(\ell) = 160$. Therefore, when the number of server is greater than 32, our batch verification method becomes to be more efficient than the standard method.

In order to be self-contained, we provide here an estimation of the complexity of the proofs. The verifications of the ℓ^2 proofs has a complexity negligible in relation to the previous operation. We can essentially summarize the proofs as checking whether the precomputed value g^r is equal to $g^z \times y^{-e} \bmod p$ and G^r is equal to $G^z w^N Y^{-e} \bmod N^2$. We have to carry out ℓ^2 products of two or three numbers where the size of the exponents are in $|A|$ or $|N|$. Therefore, if we call the FastMult algorithm, we obtain $|A|\ell^2$ for the verifications in $\mathbb{Z}_p{}^*$ and $|N|\ell^2$ for the proofs in N^2. Consequently, the computation complexity is upper bounded by the Random Linear Combination Test in the exponent for all schemes.