Improved On-Line/Off-Line Threshold Signatures

Emmanuel Bresson[1], Dario Catalano[2⋆], and Rosario Gennaro[3]

[1] DCSSI Crypto Lab, 51 bd de La Tour-Maubourg, 75700 PARIS 07 SP, France.
Email: emmanuel.bresson@polytechnique.org
[2] Dipartimento di Matematica e Informatica, Università di Catania, Viale Andrea Doria 6, 95125 Catania, Italy. Email: catalano@dmi.unict.it
[3] I.B.M. T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598.
Email: rosario@us.ibm.com

Abstract. At PKC 2006 Crutchfield, Molnar, Turner and Wagner proposed a generic threshold version of on-line/off-line signature schemes based on the "hash-sign-switch" paradigm introduced by Shamir and Tauman. Such a paradigm strongly relies on *chameleon hash functions* which are collision-resistant functions, with a secret trapdoor which actually allows to find arbitrary collisions efficiently. The "hash-sign-switch" paradigm works as follows. In the off-line phase, the signer hashes and signs a random message s. When, during the on-line phase, he is given a message m to sign the signer uses its knowledge of the hash trapdoor to find a second preimage and "switches" m with the random s. As shown by Crutchfield *et al.* adapting this paradigm to the threshold setting is not trivial. The solution they propose introduces additional computational assumptions which turn out to be implied by the so-called one-more discrete logarithm assumption.

In this paper we present an alternative solution to the problem. As in the previous result by Crutchfield *et al.*, our construction is *generic* and can be based on any threshold signature scheme, combined with a chameleon hash function based on discrete log. However we show that, by appropriately modifying the chameleon function, our scheme can be proven secure based *only* on the traditional discrete logarithm assumption. While this produces a slight increase in the cost of the off-line phase, the efficiency of the on-line stage (the most important when optimizing signature computation) is unchanged. In other words the *efficiency* is essentially preserved. Finally, we show how to achieve *robustness* for our scheme. Compared to the work by Crutchfield *et al.*, our main solution tolerates at most $\lceil n/4 \rceil$ (arbitrarily) malicious players instead of $\lceil n/3 \rceil$ however we stress that we do not rely on random oracles in our proofs. Moreover we briefly present a variant which can achieve robustness in the presence of $\lceil n/3 \rceil$ malicious players.

⋆ Work partially done while CNRS Researcher at the Laboratoire d'Informatique de l'Ecole Normale Superieure, Paris, France.

1 Introduction

In a threshold signature scheme [6], digital signatures can be produced by a group of n players (rather than by one party) who hold the secret key in a shared form among them. In order to produce a valid signature on a given message m, the individual players engage in a communication protocol that has the signature as its output: a simplified way to think about this is that each player produces a *partial signature* on the message, and then the players combine them into a full signature on m. A threshold signature scheme achieves threshold $t < n$, if no coalition of t (or less) players can produce a new valid signature, even after the system has produced many signatures on different messages. Threshold signatures are mainly motivated by the need to protect signature keys from the attack of internal and external adversaries: by keeping these keys shared, the adversary must compromise at least $t+1$ servers to learn the private signing key. Threshold signatures have found many practical applications, not only in the area of protecting high-security keys (such as the signature key of a certification authority), but also as a tool implementing secure distributed protocols, such as large-scale distributed data storage systems [17, 19].

The most serious obstacle in the practical deployment of threshold signatures is the time needed to compute signatures, since the "normal" costs of public-key operations required by a "centralized" digital signature are magnified by the communication and computations required by the distributed protocol that computes threshold signatures. As pointed out in [4], Pond (a prototype for OceanStore a large-scaled distributed data storage system [17]) spends 86% of its time computing threshold signatures. Thus it is important to look for ways of speeding up signature computation, without compromising security.

The idea proposed in [4] (the inspiration for our work) is to use *on-line/off-line signatures* (introduced in [8]). In these signatures the signing process is divided in two parts: a computationally intensive part which is done off-line, i.e. before the message being signed is known. This off-line part produces some temporary data which is stored and then used at the time the message to be signed is known. At that point, the computation of the actual signature requires very little effort. Such signatures can be constructed starting from any regular digital signature, via combination with one-time signatures (as in [8]) or chameleon hashing ([15, 21]). It is also worth pointing out that some digital signature schemes (e.g. the Digital Signature Standard [16]) are intrinsically on-line/off-line.

What we need then, is an *on-line/off-line threshold digital signature*. We should point out that the threshold DSS signatures presented in [12] is an example of such signature. What we are interested however is a *generic* solution: a way to convert any threshold signature into an on-line/off-line one. The work by Crutchfield *et al.* in [4] is the first example of that. They showed how to combine any threshold signature with a threshold version of a specific chameleon hash function based on the discrete logarithm problem (the well known Pedersen commitment [18]). The final result is a reasonably efficient scheme whose security holds under the security of the original signature scheme together with

the *one-more discrete-log assumption* (recalled below), which is stronger than the traditional assumption of computational infeasibility of the discrete log function.

OUR RESULTS. We present a new and improved generic on-line/off-line threshold signature scheme. As in [4] we combine any threshold signature scheme, with a chameleon hash function based on discrete log. However our scheme can be proven secure based only on the traditional discrete log assumption. The price to pay is a slight increase in the cost of the off-line signature computation component, but the efficiency of the on-line part remains unchanged with respect to [4]. Thus we present a scheme that without compromising the overall level of efficiency improves [4] on the security assumption.

1.1 The Approach in a Nutshell

First we describe the so-called "hash-sign-switch" paradigm as introduced by Shamir-Tauman [21], that uses chameleon hashing [15] to construct on-line/off-line signatures. Then we discuss the threshold version in [4] and finally our improvement on it.

A chameleon hash function is defined by a public key CH (we use the public key to denote the actual function) and a secret trapdoor T. It takes two arguments a message m and a random string r. The function is collision-resistant, unless one knows the trapdoor T. But knowledge of T allows to find arbitrary collisions, i.e. given $c = CH(m, r)$ and an arbitrary message m', the holder of the trapdoor can find r' such that $c = CH(m', r')$. For many chameleon hash functions, this collision-finding procedure is very efficient, requiring only a single modular multiplication. The Shamir-Tauman [21] idea is to construct on-line/off-line signatures as follows. The off-line part would consists of computing $c = CH(a, r')$ for some arbitrary a, r' and then computes s the signature of c under an ordinary signature scheme. On input the actual message m the signer (who knows the trapdoor T as part of the signing key) computes r such that $c = CH(m, r)$ and outputs r, s. The verifier computes c and verifies s on it.

The contribution of Crutchfield *et al.* is to build a way to compute the values c and r distributively, i.e. by servers who hold T in a shared form. They use Pedersen's commitment [18] as the chameleon hash function: $CH(m, r) = g^r h^m$ in a cyclic group of prime order. To "thresholdize" CH they use techniques developed in the context of discrete-log based threshold cryptography (e.g. [12]). The proof of their scheme has, however, a subtle issue. The proof of security of the threshold scheme in [4] is carried out via simulation: an adversary forging the threshold scheme is transformed via a simulation of the distributed environment into a forger for the centralized scheme, or a collision-finder for CH. In the [4] protocol the value c is revealed to the adversary (who may have corrupted up to t of the signing servers) *before* the final signature is computed (as opposed to the centralized Shamir-Tauman solution where the adversary sees c only after the signature is issued). This in turns means that the simulation of the on-line phase is constrained to use a specific c generated in a simulation performed before m was known. This is why the proof in [4] must use the stronger one-more discrete log assumption.

Our contribution is an alternative way to get around the above problem, so that we do not require this stronger assumption. The basic idea is to use a variation of Pedersen's commitment for the chameleon hashing. We define $CH(m,r,s) = g^m h_1^r h_2^s$. The crucial property of this chameleon hash function is that it has two "independent" trapdoors ($\log_g h_1$ and $\log_g h_2$), and we can give a random one to the simulator to help in performing the simulation[4]. On the other hand if the adversary finds a collision for CH, with probability $1/2$ this collision will reveal the other trapdoor, thus allowing us to solve the discrete log problem.

On the difference between the assumptions. Our proof relies on the standard discrete log assumption: given a cyclic group G of prime order q, a generator g, and a random value $y \in G$, find $z \in Z_q$ such that $y = g^x$. This assumption has been widely used and it's the basis of many of the cryptographic schemes used in practice. The assumption used in [4] is stronger since the adversary is given access to an oracle that computes discrete logs: on input y the oracle returns x such that $y = g^x$. The task is then: given k random values in G, $y_1, y_2, \ldots, y_k$, find *all* the discrete logs x_i s.t. $y_i = g^{x_i}$, while being allowed to query the oracle at most $k-1$ times. This assumption is newer and not as established as the traditional discrete log assumption.

On the robustness guarantees. It is desirable in a distributed environment to be able to guarantee robustness. Informally, this means that, even if up to t players behave dishonestly, the remaining honest ones are still able to perform the computation correctly. The scheme proposed in [4] enables such property, either in the random oracle model, or by using a technique pointed out by Damgård and Dupont [5], and provided that $n > 3t + 1$. As a comparison, our technique allows to deal with up to $n/3$ players that can be either *halting* at any time, or arbitrarily *malicious* except during the on-line signing phase. If we want to tolerate malicious players at any step of the protocol, we have to[5] restrict the threshold to $t < n/4$.

1.2 Related work

As we pointed out, Even *et al.* introduced the notion of on-line/off-line signatures in [8] and constructed them combining regular signatures with efficient one-time signatures. However the length of the signatures is an issue in this approach. Shorter signatures can be obtained by using chameleon hashing [15] combined with regular signatures as pointed out by Shamir and Tauman [21].

[4] The idea of using two independent trapdoors to construct a secure digital signature scheme is not new, as it goes back to the seminal paper of Goldwasser, Micali and Rivest [14].

[5] More precisely, it would be possible to tolerate one third of the players behaving maliciously at any time, by using general techniques such as non-interactive zero-knowledge proofs in order to enhance every protocol step with robustness. However, the obtained scheme would have become highly inefficient; we decided to maintain practicability rather than optimizing threshold.

Threshold signature schemes were introduced by Desmedt and Frankel [6]. We point out that threshold DSS signatures (constructed in [12]) are intrinsically on-line/off-line and do not require the extra steps described in this paper or [4]. On the other hand the techniques in this paper allow the underlying signature to be any desired scheme. RSA-based threshold signatures (which can be used as the underlying scheme in our construction) are presented in [11, 22].

2 Definitions and Notation

A function $f : \mathbb{N} \to \mathbb{R}$ is said to be negligible if for any $c > 0$, there exists an index $k_c \in \mathbb{N}$ such that $f(k) < k^{-c}$ for all $k > k_c$. PPT stands for Probabilistic Polynomial-Time. In several points in this paper we make use of a cyclic subgroup of prime order in a finite field $\mathbb{Z}_p$. To fix the notations, we denote by p and q two prime numbers such that $q|p-1$ and q is sufficiently large. Moreover, we will denote by G a subgroup of $\mathbb{Z}_p^\star$ with order q and by g a generator of G.

Definition 1 (Discrete logarithm assumption). *Let $k = |q|$ be a security parameter. The Discrete Logarithm (DLOG) Assumption in G states that for any PPT algorithm $\mathcal{A}$, the probability that $\mathcal{A}$ outputs x on input (p, q, g, g^x) is negligible in k (the probability space is on the random choice of p, q, g and $x \in \mathbb{Z}_q$ and the internal coins tosses of $\mathcal{A}$).*

For lack of space we omit the definition of digital signature schemes.

Definition 2 (On-line/off-line signature scheme). *An on-line/off-line signature scheme $\Sigma^{\mathsf{on,off}} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ is a signature scheme in which the signing algorithm Sign can be divided into two phases:*

- Off-line phase: *an algorithm $\mathsf{Sign}^{\mathsf{off}}$ that takes as input the private key and generates a signature token σ^{off},*
- On-line phase: *an algorithm $\mathsf{Sign}^{\mathsf{on}}$ which on input a message m and a signature token σ^{off}, together with the private signing key, produces a signature σ on m.*

For this definition to be of practical interest, it is required that the cost of the on-line phase is as small as possible.

Definition 3 (Threshold signature scheme). *A threshold signature scheme $\mathsf{T}\text{-}\Sigma$ consists of the following PPT algorithms $(\mathsf{T}\text{-}\mathsf{KeyGen}, \mathsf{T}\text{-}\mathsf{Sign}, \mathsf{Ver})$:*

- *the key generation algorithm $\mathsf{T}\text{-}\mathsf{KeyGen}(1^\ell)$ is a distributed key generation algorithm that generates a public key pk and provides each party with a share sk_i of the secret key sk;*
- *the threshold signing protocol $\mathsf{T}\text{-}\mathsf{Sign}$ runs in two phases:*
  - *the signature share generation $\mathsf{T}\text{-}\mathsf{Sign}^{\mathsf{share}}(m, \{sk_i\})$ is run interactively so that each party obtains a share σ_i of a signature on the message m,*
  - *the signature reconstruction $\mathsf{T}\text{-}\mathsf{Sign}^{\mathsf{combine}}(\{\sigma_i\})$ builds the full signature σ given the generated signature shares σ_i;*

– *the verification algorithm* $\mathsf{Ver}(pk, m, \sigma)$ *is unchanged.*

It is required that such scheme is simulatable, in the sense of [10]. Also it is worth noticing that the notion is independent of the on-line/off-line feature. Here, we are going to consider "threshold" on-line/off-line signatures. In this case, the signature share generation coincides with the off-line phase of the scheme: the obtained "shares" are generated without knowing the message to be signed; and the signature reconstruction coincides with the on-line phase. For completeness, we provide below a formal definition for such scenario.

Definition 4 (On-line/off-line threshold signature). *An on-line/off-line threshold signature* $\mathsf{T}\text{-}\Sigma^{\mathsf{on,off}}$ *is made of the following components:*

– $\mathsf{T}\text{-}\mathsf{KeyGen}(1^\ell)$ *is the distributed key generation algorithm that generates* pk *and provides each party with a share* sk_i*;*
– $\mathsf{T}\text{-}\mathsf{Sign}^{\mathsf{share,off}}$ *is the off-line signature share generation that generates a signature token* σ^{off} *and provides each party with a signature share* σ_i*;*
– $\mathsf{T}\text{-}\mathsf{Sign}^{\mathsf{combine,on}}$ *is the on-line reconstruction phase that, given the message* m *to be signed, produces the final signature* σ *from the token* σ^{off} *and the private signature shares* σ_i*;*
– Ver *is the verification algorithm (unchanged).*

Security of a threshold signature scheme can be defined in several ways, but the strongest definition (see [12]) requires the protocols to be *simulatable*, which guarantees that the threshold signature scheme is as secure as its centralized version. If the protocol is secure even in the presence of t arbitrarily malicious players, then the protocol is called *robust*.

3 Building blocks

In this section we briefly discuss some basic protocols that are going to be useful in the sequel. In the following we will denote by n the number of players involved in the protocol (in particular we assume $n \ll q$). We assume that the players are connected through point-to-point private channels and by a broadcast channel. We model failures on the network by allowing the existence of an adversary who is allowed to corrupt up to $t < n/3$ players[6]. The adversary is assumed to be *static*, meaning that the set of corrupted players is chosen at the beginning of the protocol.

All the basic protocols presented in this section require $O(1)$ rounds of communication. We assume that all secrets are shared through a secret sharing scheme *à la Shamir* [20], using polynomials of degree t and, throughout this section, we assume that $t < n/3$. We remark that, for these choices of parameters, all the following protocols already provide robustness (or they can easily be modified to do so using very standard techniques).

[6] For the protocols described in the next section, however, we will require $t < n/4$ to guarantee robustness.

MULTIPLYING TWO SHARED SECRETS. For this task we adopt the well known protocol by Ben-Or et al. [2]. In what follows we denote $\texttt{MUL}[a_i, b_i] \rightarrow [c_i]$ an execution of this protocol, where a_i and b_i are the original shares held by player P_i and c_i is the share obtained after the additional communication round.

We stress that it is well known [2], how to modify the above protocols in order to achieve robustness against a static adversary controlling up to $t < n/3$ players.

PEDERSEN'S VSS. Pedersen's Verifiable Secret Sharing protocol [18], extends Shamir secret sharing scheme [20] in order to tolerate a malicious adversary corrupting up to $n/2$ players, including the dealer. Moreover the scheme preserves the security of the secret in a strong information theoretic sense. In a nutshell the scheme goes as follows. Let h be another generator of G, such that the discrete logarithm of h in base g is unknown and assumed to be hard to compute. In the sharing phase, the dealer $\mathcal{D}$ starts the protocol by choosing two (random) polynomials $f(\cdot)$ and $g(\cdot)$ of degree t, such that $f(0) = a$, where a is the secret being shared. Next, it gives the values $(a_i, r_i) = (f(i), g(i))$ to each participants P_i. Moreover it broadcasts the verification values $V_j = g^{\alpha_j} h^{\beta_j} \bmod p$ where α_j (resp. β_j) is the j-th coefficient of $f(\cdot)$ (resp. $g(\cdot)$). By these positions, each player is allowed to verify the validity of the received shares by simply checking that $g^{a_i} h^{r_i} = \prod_{j=0}^{t} V_j^{i^j} \bmod p$.

If some player holds shares that do not satisfy the equation above, he broadcasts a complain against the dealer. If more than t players do so, the dealer is disqualified. Otherwise the dealer publishes the values $f(i), g(i)$ matching the equation above for each complaining party P_i.

In the reconstruction phase each player P_i is required to reveal both $f(i)$ and $g(i)$. This is to make sure that players provide the (correct) shares they originally received. Notice that a dishonest player can provide incorrect shares that are consistent with the equation above if and only if it can compute the discrete logarithm of h in base g. Thus, Pedersen's VSS guarantees soundness only with respect to polynomially bounded adversaries.

We denote by $\texttt{Ped-VSS}[a, r](g, h, p, q, t, n) \rightarrow [a_i, r_i](V)$ an execution of Pedersen's VSS protocol where the dealer distributes a secret a, using the additional random value r, with public parameters (g, h, p, q, t, n). Moreover, a_i, r_i denote the local (secret) shares received by player P_i at the end of the distribution phase. $V = \{V_j\}$ denotes the set of commitments broadcasted by the dealer during the execution of the protocol.

JOINT PEDERSEN'S VSS. The sharing phase of Pedersen's VSS can be easily generalized to the case where no special dealer is required and where the players jointly generate a random shared secret. We denote with $\texttt{Joint-RPed-VSS}(g, h, p, q, t, n) \rightarrow [a_i, r_i, T_S, a](V, Q)$ the execution of the protocol with public parameters g, h, p, q, t, n and where each player P_i gets as local output the shares a_i, r_i, with a_i referring to the final secret a. Q denotes the subset of $\{1, \ldots, n\}$ of the indexes of the players that have not been disqualified during the execution of the protocol. Finally T_S denotes the transcript produced by the n VSS's executed by the players.

COMPUTING SHARES OF THE INVERSE OF A SHARED SECRET. Let a be an *invertible* element in $\mathbb{Z}_q$. Assume that a is shared among the players and denote with a_i the share held by player P_i. The following protocol, due to Bar-Ilan and Beaver [1], allows to compute shares of b, such that $ab \equiv 1 \bmod q$ from shares of a. The basic idea is as follows. First the players jointly generate a shared random value r (using the protocol described above), then they multiply the two shared secrets a and r by means of the (full) multiplication protocol. To conclude this phase, the players reveal the shares obtained after the execution of the multiplication protocol and they jointly reconstruct the value $u \equiv ar \bmod q$. If $u \equiv 0 \bmod q$ the protocol is restarted. Otherwise u is invertible modulo q and every player can locally compute his share of $a^{-1} \bmod q$ by setting $b_i = r_i \cdot u^{-1} \bmod q$. We denote this protocol by $\texttt{INV}[a_i] \rightarrow [b_i]$.

SHARED EXPONENTIATION OF SECRETS [7]. This allows to compute $g^a h_1^b h_2^c$ when a, b, c are shared secrets. For lack of space the description of this protocol is omitted. The interested reader can find full details in [7]. In the following we will refer to this protocol as $\texttt{Share-Exp}(g, h_1, h_2) \rightarrow (g^a h_1^b h_2^c)$.

DISCRETE LOG-BASED DISTRIBUTED KEY GENERATION [10]. This protocol allows a set of user to securely generate private keys for discrete log based encryption schemes (see [10] for details). In the following we will refer to this protocol as $\texttt{DL-DKG}(g, h, p, q, t, n) \rightarrow [x_i](y, V, Q)$.

4 The new scheme

We now describe our generic on-line/off-line threshold signature scheme. This scheme can be based on any threshold signature $\mathsf{T}\text{-}\Sigma = (\mathsf{T}\text{-}\texttt{KGen}, \mathsf{T}\text{-}\texttt{Sig}, \texttt{Ver})$. We will focus on an *optimistic* version of it where, instead of verifying correctness each time a new signature is generated, verification occurs only if a signature happens to be invalid.

Recall that a generic threshold on-line/off-line signature scheme $\mathsf{T}\text{-}\Sigma^{\mathsf{off,on}}$ is composed of the following algorithms

$$\mathsf{T}\text{-}\Sigma^{\mathsf{off,on}} = (\mathsf{T}\text{-}\mathsf{KeyGen}, \mathsf{T}\text{-}\mathsf{Sign}^{\mathsf{share,off}}, \mathsf{T}\text{-}\mathsf{Sign}^{\mathsf{combine,on}}, \mathsf{Ver})$$

In what follows we assume that $t < n/4$.

KEY GENERATION. This protocol is performed only once. The full description is given in Figure 1. We assume that the primes p, q and two generators g, g_1 of a subgroup G of order q in $\mathbb{Z}_p^*$ are given as public parameters to the players. Note that such an assumption can be relaxed using standard techniques: for example it is possible to consider a more general key generation protocol where the parties jointly choose the primes p and q as well as the generators g and g_1. However we believe that such a formulation would only make the presentation more intricate, thus distracting the reader from the focus of this paper, which are the protocols for threshold on-line/off-line threshold signatures.

OFF-LINE SIGNING. The signing protocol for the off-line phase is described in Figure 2. We remark here that every time the `Joint-RPed-VSS` protocol is executed, the sharing polynomial is tacitly assumed to have degree t.

On-line/Off-line Threshold Key Generation Protocol

Public Parameters: a set of n players $P_1, \ldots, P_n$, a security parameter k, two primes p, q such that $q|p-1$ and $|q| = k$, two elements g, g_1 of order q in $\mathbb{Z}_p^*$, a threshold parameter $t < n/4$ and a threshold signature scheme T-Σ. We denote by G the subgroup of $\mathbb{Z}_p^*$ generated by g;
Common Output: the public key of the scheme;
Private Output (for player P_j): a share of the signing secret key.

1. The players jointly run the T-KGen algorithm, on input 1^k. This produces a public verification key vk. Moreover each player privately receives a share sk_i of the corresponding signing key.
2. The players jointly run the DL-DKG(g, g_1, p, q, t, n) algorithm twice (with parameters g, p, q) in order to obtain two public values h_1, h_2. We denote with y_i and z_i the shares of the two secret keys y, z (such that $g^y = h_1$ and $g^z = h_2$) held by player P_i.
3. The players run the INV(y_i) protocol to get shares Y_i of the inverse Y of y.
4. The public key is set as $PK = (g, p, q, g_1, vk, h_1, h_2)$, while each player P_i retains the quadruple $SK_i = (sk_i, y_i, z_i, Y_i)$ as its own local secret key.

Fig. 1. The Key Generation Protocol for our On-line/off-line Threshold Signature Scheme

On-line Signing. The signing protocol for the on-line phase is described in Figure 3. We remark here that no signature share verification is explicitly required by the protocol. This is because we decided to follow an optimistic approach (in general it is reasonable to assume that the signature shares are going to be correct almost all the time). Still, in order to guarantee robustness, we need to make sure that, even if some players sent incorrect shares, honest participants should be able to reconstruct a valid signature. Later we describe how to achieve that for the case $n > 4t$.

Verification. Given a purported signature $(\mathtt{Com}, \rho, r, s)$ on a message m, one accepts it as valid if the following relation is true

$$\mathtt{Ver}(vk, \mathtt{Com}, \rho) \stackrel{?}{=} 1 \quad \wedge \quad \mathtt{Com} \stackrel{?}{=} g^m h_1^r h_2^s$$

Achieving Robustness. If the verification procedure $\mathtt{Ver}(vk, \mathtt{Com}, \rho)$ fails, then some of the participants are providing incorrect shares. In principle, one can always reconstruct the correct signature as our assumption that $n > 4t$ assures us enough points to correctly interpolate s' and r'. The trivial approach of trying all the possible subsets of $2t+1$ shares, however, does not work, as the number of such subsets is in general exponential (in n). Here we suggest the following two-phases approach.

First Phase: In the first phase, the correctness of the s_i''s is verified. This is done though the commitment materials produced during round 6 of the off-

Off-line Threshold Signing Protocol

Public Parameters: a set of n players $P_1, \ldots, P_n$, a security parameter k, two primes p, q such that $q|p-1$, $|q| = k$, two generators g, g_1 of G, a threshold parameter $t < n/4$ and a threshold signature scheme $\mathsf{T}\text{-}\Sigma$;
Private Input (for player P_j): the local signing key $SK_j = (sk_j, y_j, z_j, Y_j)$;
Private Output (for player P_j): a signature token σ^{off} and a signature share σ_j.

1. The players jointly run the `Joint-RPed-VSS`(g, g_1, p, q, t, n) protocol three times to produce three shared random values m, r, s. Let m_i, r_i, s_i be the shares obtained by player P_i after participating to the three `Joint-RPed-VSS`.
2. The players execute the `Share-Exp` protocol, each holding local inputs m_i, r_i, s_i. Let $\mathtt{Com} = g^m h_1^r h_2^s$ be the public output.
3. The players run the (entire) `T-Sig` algorithm to compute a signature ρ on the message `Com`.
4. The players run (a simplified version of) the `Joint-RPed-VSS` algorithm (with parameters g, p, q) to generate shares ω_i of a $2t$-degree (random) polynomial p_0, such that $p_0(0) = 0$.
5. The players run the full multiplication protocol `MUL` twice to compute shares of the products $r \cdot y$ and $s \cdot z$. Finally they (non interactively) compute shares of the quantity $m + r \cdot y + s \cdot z$. Let τ_i be the share held by player P_i.
6. The players jointly run the `Joint-RPed-VSS`(g, g_1, p, q, t, n) protocol in order to produce a shared random value s'. Let s'_i be the share obtained by player P_i as a local output.
7. The output signature token is $\sigma^{\mathsf{off}} = (\mathtt{Com}, \rho)$ while the signature share for P_i is $\sigma_i = (\omega_i, \tau_i, s'_i)$

Fig. 2. The signing algorithm for the off-line stage

line threshold signing protocol. If t shares turn out to be incorrect, this allow us to identify and remove all the dishonest players immediately (and then there is no need to proceed to phase two).

Second Phase: If less than t incorrect shares have been identified in phase one, in round 3 of the on-line phase, the players interpolate the correct r' using the Berlekamp-Welch decoder [3]. The correctness of this approach follows from the error correcting capabilities of polynomial interpolation. Since we are interpolating a polynomial of degree $d = 2t$ and we have up to $f = t$ erroneous points, using the Berlekamp-Welch bound we get that the number of points needed to correctly interpolate is $d + 2f + 1$, which, in our case, means, $2t + 2t + 1 = 4t + 1$ (this is why we required $n > 4t$).

Remark 1. We stress that, the key generation and the off-line signing protocols, can achieve robustness even with respect to an adversary controlling up to $n/3-1$ players (rather than the more restrictive setting $t < n/4$). This is because, as observed in Section 3, all the protocols we are using as building blocks (i.e.

On-line Threshold Signing Protocol

Public Parameters: A set of n players $P_1, \ldots, P_n$, a security parameter k, two primes p, q such that $q|p-1$ and $|q| = k$, two generators g, g_1 of G, a threshold parameter $t < n/4$ and a threshold signature scheme $\mathsf{T}\text{-}\Sigma$;
Public Input: a message m' to be signed;
Private Input (for player P_j): the signing key $SK_j = (sk_j, y_j, z_j, Y_j)$, together with the signature token $\sigma^{\mathsf{off}} = (\mathtt{Com}, \rho)$ and the signature share $\sigma_i = (\omega_i, \tau_i, s'_i)$ produced during the off-line stage;
Public Output: a signature σ for m'

1. Each player broadcasts the share s'_i. This allows the player to locally interpolate the value s'.
2. Each player P_i locally computes the following share

$$r'_i = (\tau_i - m' - s' \cdot z_i) \cdot Y_i + \omega_i \bmod q$$

3. Finally the players broadcast their shares r'_i, in order to reconstruct r'.
4. The signature for m' is given by

$$\sigma = (\mathtt{Com}, \rho, r', s')$$

Fig. 3. The signing algorithm for the on-line stage

those described in Section 3) are already robust against such kind of adversaries, or they can easily be modified to achieve robustness. By contrast, the on-line signing protocol makes use of a reconstruction phase for values that are shared over $2t$-degree polynomials and thus, requires the threshold to be bounded by $n/4$.

5 Security Proof

Theorem 1. *Assuming that* $\mathsf{T}\text{-}\Sigma = (\mathsf{T}\text{-}\mathtt{KGen}, \mathsf{T}\text{-}\mathtt{Sig}, \mathtt{Ver})$ *is a threshold signature scheme secure against adaptive chosen message attack and the discrete logarithm assumption holds, the On-Line/Off-line Threshold Signature scheme presented above is existentially unforgeable against an adaptive chosen message attack, mounted by a static adversary controlling up to one fourth of the n participants.*

Proof (Sketch). The proof goes by contradiction, we assume that there exists an adversary $\mathcal{A}$ that breaks the existential unforgeability of the proposed scheme and we show how to exploit it to break either the unforgeability of the underlying signature scheme $\mathsf{T}\text{-}\Sigma$ or the discrete logarithm assumption. In other words, we build an efficient algorithm $\mathcal{B}$ that, using $\mathcal{A}$ as a black box, succeeds in the above mentioned tasks.

Notice that, any valid forgery must be of one of the following types

- **Type I:** $(\mathtt{Com}, \rho, s', r')$ on a message m' such that $\mathtt{Com} \neq \mathtt{Com}_i$ for all previously issued signatures $(\mathtt{Com}_i, \rho_i, s'_i, r'_i)$ on messages m'_i,
- **Type II:** $(\mathtt{Com}, \rho, s', r')$ on m' such that $\mathtt{Com} = \mathtt{Com}_i$ for some previously issued signature $(\mathtt{Com}_i, \rho_i, s'_i, r'_i)$ on a message m'_i, but at least two of the following conditions must hold
  1. $m' \neq m'_i$
  2. $s' \neq s'_i$
  3. $r' \neq r'_i$

It is easy to get convinced that the above (mutually exclusive) conditions cover the entire spectrum of possibilities.

TYPE I FORGERIES. We show how to build an algorithm $\mathcal{B}$ against the existential unforgeability of T-Σ using an adversary $\mathcal{A}$ that produces this type of forgeries with non-negligible probability. To do so we start with $\mathcal{B}$ receiving as input, in a preliminary phase, the public key material of a secure threshold signature scheme T-Σ = (T-KGen, T-Sig, Ver). His goal is to use the forgery produced by $\mathcal{A}$ to contradict the existential unforgeability of T-Σ. This means that, after having received a number of signatures $\mathtt{Sig}_i$ for messages M_i of its own choice, $\mathcal{B}$ should be able to produce a couple $(M, \mathtt{Sig})$ such that Sig is a valid signature for the message M with respect to the given public key (and, of course, $(M, \mathtt{Sig}) \neq (M_i, \mathtt{Sig}_i)$ for all i's).

First note that, being T-Σ a secure threshold signature scheme scheme we require that it is simulatable, in the sense of [10]. In particular this means (see [10]) that:

1. The algorithm T-KGen is simulatable, meaning with this that there exists a simulator S_1 that, on input the verification key and the public output generated by an execution of T-KGen, can simulate the view of the adversary on that execution.
2. The protocol T-Sig is simulatable, meaning with this that there exists a simulator S_2 that, on input the public input of T-Sig, t shares, and the produced signature σ, can simulate the view of the adversary on an execution of T-Sig that outputs σ.

With this in mind we show how to simulate the three protocols presented in the previous section.

On-line/Off-line Threshold Key Generation: $\mathcal{B}$ performs rounds 2, 3 and 4 exactly as in the real game, meaning with this that it plays the role of each honest player exactly as prescribed by the protocol.
Round 1 is done by running the simulator S_1 on input the relevant values $\mathcal{B}$ has received in the preliminary phase.
Thus, by the simulatability property of T-KGen, the entire simulation of T-KeyGen is indistinguishable from a real execution of the protocol.

Off-line Threshold Signing Protocol: $\mathcal{B}$ executes the following variant of the T-Sign$^{\mathsf{share,off}}$ protocol.
Steps 1, 2, 4, 5 and 6 are done exactly as in the original protocol, thus we

focus on step 3. At that point $\mathcal{B}$ queries his signing oracle (which is relative to T-Sig) in order to get a signature ρ_i on the computed $\mathtt{Com}_i$. Then $\mathcal{B}$ executes the simulator S_2 on input the public parameters, the shares of the controlled players and the value of ρ_i in order to produce the corresponding view. By the simulatability of T-Sig this is indistinguishable from a real execution.

On-line Threshold Signing Protocol: Whenever $\mathcal{A}$ asks the i-th signature query on a message m'_i, $\mathcal{B}$ executes the protocol exactly as prescribed in the previous section.

Now assume that, once $\mathcal{A}$ is done with its signing queries, it produces a forgery of type I $(\rho, \mathtt{Com}, s', r')$ on a message m'. Type I forgery means that Com differs from all $\mathtt{Com}_i$ and thus was never queried by $\mathcal{B}$ to its signing oracle. Then $\mathcal{B}$ produces its own forgery against T-Σ by setting $M = \mathtt{Com}$ and $\mathtt{Sig} = \rho$.

TYPE II FORGERIES. We show how to build an algorithm $\mathcal{B}$ that breaks the discrete logarithm assumption using an adversary $\mathcal{A}$ that produces this type of forgeries with non-negligible probability. To do so we start with $\mathcal{B}$ receiving as input, in a preliminary phase, a couple $(g, h) \in G^2$. His goal is to use the forgery produced by $\mathcal{A}$ to determine the discrete log of h in base g.

We assume that $\mathcal{B}$ is allowed to program the common parameters g, g_1, in the sense that it is allowed to set g as the g received in the preliminary phase and to choose g_1 according to a distribution that is perfectly indistinguishable from the distribution according to which g_1 has to be chosen. In particular, notice that this allows $\mathcal{B}$ to choose g_1 in a way such that it knows the discrete log of g_1 in base [7] g. In what follows we assume, for simplicity, that $m' \neq m'_i$ always holds. It is straightforward to extend the proof to the more general case where m' (the forged message) may be equal to m'_i (the message queried for signing).

First, $\mathcal{B}$ flips a coin β. If $\beta = 0$ it bets on the fact that $\mathcal{A}$ will provide a forgery of type II where conditions 1 and 3 above hold true, that is $m' \neq m'_i$ and $r' \neq r'_i$. If $\beta = 1$ $\mathcal{B}$ bets on the fact that the forgery will satisfy $m' \neq m'_i$ and $s' \neq s'_i$. Informally the proof goes in two stages. In the first one $\mathcal{B}$ will simulate a real execution of the protocol, playing the role of non-corrupted parties. In this phase we have to make sure that the simulated protocol is perfectly indistinguishable from the real one. In the second part of the proof, we show how $\mathcal{B}$ can exploit the provided forgery to solve the received discrete logarithm challenge.

As for the first part of the proof, we describe in detail the simulation of the three protocols, described in the previous section.

On-line/Off-line Threshold Key Generation: $\mathcal{B}$ performs steps 1 and 4 exactly as in the real game, meaning with this that it plays the role of each honest player exactly as prescribed by the protocol.

Step 2 is done as follows. The first execution of the DL-DKG protocol (i.e. the one leading to the generation of h_1) is replaced by a execution of the simulator S for DL-DKG, as given in [10], on input (g, h). As a result, this produces the public value $h_1 = h$ and properly distributed values for the

[7] Formally this is equivalent to assume that all the public parameters are part of a shared random string, that the simulator is allowed to "program" in the proof

parties controlled by $\mathcal{A}$. In particular the simulation looks to $\mathcal{A}$ perfectly indistinguishable from the real execution of the protocol, however the players will share some secret value $\hat{y}$ that does not correspond to the actual discrete log of h_1 in base g. The second execution of the DL-DKG protocol is done as in the real game.

Step 3. $\mathcal{B}$ runs an execution of the INV protocol, but with each of the honest players holding a share $\hat{y}_i$ of $\hat{y}$. Notice that such an execution looks perfectly indistinguishable (with respect to the real one) to $\mathcal{A}$, as the latter is static and controls only up to $t < n/4$ players.

Hence the simulation provides the adversary with a view (public outputs + controlled players' private outputs) which is perfectly indistinguishable from a real execution.

Off-line Threshold Signing Protocol: Steps 1, 2, 3, 4 and 6 are done exactly as in the real game, thus we focus on step 5. Here the only difference with respect to the real protocol is that the (full) multiplication protocol MUL used to compute $r \cdot y$ is run by $\mathcal{B}$ using the shares $\hat{y}_i$ for the honest players. Once again, this results in a protocol which looks perfectly indistinguishable to the real one, from $\mathcal{A}$'s perspective.

On-line Threshold Signing Protocol: $\mathcal{B}$ first recovers the value s' shared during the off-line phase. Notice that it can do this as it controls $n - t > 3n/4 > t$ parties. Next, once m' is known, it sets $r' = r$ and and computes a value $\hat{s}'$ such that $r' = (m + r\hat{y} + sz - m' - \hat{s}'z)\hat{Y} \bmod q$, where $\hat{Y}$ is the (known) inverse computed in the key generation protocol. Notice that this means that $\hat{s}' = (m - m')z^{-1} + s \bmod q$. We stress that, since $\mathcal{B}$ controls more than $2t$ players it can easily compute all the values above. Next, $\mathcal{B}$ uses its knowledge of the discrete log of g_1 in base g to cheat and interpolate s' as $\hat{s}'$ (in a way that remains consistent with the previously broadcasted commitments). The rest of the protocol is done as in the real execution.

Note that the simulation is perfectly indistinguishable from the real one. This means that the adversary cannot know if the simulator knows both the values y and z or only z, as in our case. Thus if the adversary produces a forgery of type II such that $m'_i \neq m'$ and $r'_i \neq r'$ one can easily break the received discrete logarithm challenge. Indeed, since $\texttt{Com}' = \texttt{Com}_i$ we have that

$$g^{m'_i + zs'_i} h_1^{r'_i} = g^{m' + zs'} h_1^{r'}$$

and thus the required value is $((m'_i - m') + z(s'_i - s'))(r' - r'_i)^{-1} \bmod q$.

If $\beta = 1$ $\mathcal{B}$ bets on the fact that $\mathcal{A}$ will provide a forgery of type II where conditions 1 and 2 hold, that is, $m' \neq m'_i$ and $s' \neq s'_i$. Again, we describe the simulation of the three protocols, focusing on the differences with the case $\beta = 0$.

On-line/Off-line Threshold Key Generation: This time, the simulation of DL-DKG is used to generate h_2 and thus $\mathcal{B}$ knows y and a value $\hat{z}$ that differs from $\log_g h_2$.

Note, this change influences step 3, which, this time, is done exactly as in a real execution of the protocol (with $\mathcal{B}$ controlling the honest players).

Off-line Threshold Signing Protocol: Everything is done as before, by just switching the roles of z and y.

On-line Threshold Signing Protocol: In this simulation $\mathcal{B}$ uses its knowledge of the discrete logarithm of g_1 in base g to interpolate s' as the value s shared in round 1 of Off-line Threshold Signing Protocol. The rest of the protocol is done exactly as in the real game.

Again, note that the simulation is perfectly indistinguishable from the real one. Thus if the adversary produces a forgery of type II on a message m' such that $m'_i \neq m'$ and $s'_i \neq s'$ one can easily break the received discrete logarithm challenge in a way that is basically identical to what described for the case $\beta = 0$.

Remark 2 (Achieving robustness for up to $t < n/3$ faults). Notice that the protocol presented in previous section can be modified in order to tolerate up to $t < n/3$ malicious players. The modification is as follows. The key generation algorithm remains more or less unchanged: we add one additional round on which the players compute shares $\lambda_i = Y_i \cdot z_i$. In the off-line signing algorithm we add one additional execution of the (full) multiplication to create shares μ_i of the product $\tau \cdot Y$. Finally, in the on-line signing algorithm, step 2 is modified by setting $r'_i = \mu_i - m'Y_i - s'\lambda_i + \omega_i \bmod q$.

It is easy to check that the proof goes through in basically the same way. Notice that this modified protocol is less efficient than the proposed one, but the efficiency loss involves the off/line components only (i.e. key generation and off-line signing).

References

1. J. Bar-Ilan and D. Beaver. Non cryptographic fault tolerant computing in a constant number of rounds of interaction. In *Proceedings of the ACM Symposium on Principles of Distributed Computation*, pp.201–209, 1989.
2. M. Ben-or, S. Goldwasser and A. Widgerson Completeness Theorems for non-cryptographic fault tolerant distributed computation. In *Proc. of 20th Annual Symposium on Theory of Computing*, 1988.
3. E. Berlekamp and L. Welch Error correction of algebraic block codes. US Patent 4,633,470.
4. C. Crutchfield, D. Molnar, D. Turner and D. Wagner Generic On-Line/Off-Line Threshold Signatures In *Proc. of PKC '06*, pp.58–74, Lecture Notes in Computer Science vol.3958, Springer-Verlag, 2006.
5. I. Damgård and K. Dupont. Efficient Threshold RSA Signatures with General Moduli and No Extra Assumptions. In *Proc. of PKC '05*, pp 346–361, Springer-Verlag, 2005.
6. Y. Desmedt and Y. Frankel. *Threshold Cryptosystems.* CRYPTO'89, LNCS vol.435 pp.307–315, Springer 1990.
7. M. Di Raimondo and R. Gennaro Provably Secure Threshold Password-Authenticated Key Exchange. In *Proc. of Eurocrypt'03*, 2003.
8. S. Even, O. Goldreich and S. Micali. *On-Line/Off-Line Digital Signatures.* J. Cryptology 9(1): 35-67, Springer 1996.

9. P. Feldman A Practical Scheme for Non-Interactive Verifiable Secret Sharing. In *Proc. 28th FOCS*, pp. 427–437, 1987.
10. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Public-Key Cryptosystems. *Eurocrypt'99*, pp.295–310, Lecture Notes in Computer Science vol.1592, Springer-Verlag, 1999.
11. R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin. *Robust and Efficient Sharing of RSA Functions.* J. Cryptology 13(2): 273-300, Springer 2000.
12. R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin. *: Robust Threshold DSS Signatures.* Inf. Comput. 164(1): 54-84 (2001).
13. R. Gennaro, M. Rabin and T. Rabin. Simplified VSS and fast-track multi-party computations with applications to threshold cryptography. In *Proc. 17th ACM Symposium on Principle of Distributed Computing*, 1998.
14. S. Goldwasser, S. Micali and R. Rivest. A digital signature scheme secure against adaptive chosen message attacks. *SIAM J. on Computing* 17(2):281-308 1988.
15. H. Krawczyk and T. Rabin. *Chameleon Signatures.* 2000 NDSS Symposium, pp.143-154.
16. National Institute for Standards and Technology. Digital Signature Standard (DSS). Technical Report 169, August 30 1991.
17. J.Kubiatowicz, D.Bindel, Y.Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells and B. Zhao. *OceanStore: An architecture for GlobalScale Persistent Storage.* 2000 ACM Architectural Support for Programming Languages and Operating Systems Conference.
18. T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. *Crypto'91*, pp.129-140, Lecture Notes in Computer Science vol.576, Springer-Verlag, 1992.
19. S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao and J. Kubiatowicz. *Pond: The OceanStore prototype.* 2003 USENIX Conference on File and Storage Technologies.
20. A. Shamir, "How to share a secret," *Comm. of the ACM*, vol. 22, no. 11, pp. 612–613, November 1979.
21. A. Shamir and Y. Tauman. Improved On-line/Off-line Signature Schemes. *Crypto'01*, pp.355-367, Lecture Notes in Computer Science vol.2139, Springer-Verlag, 2001.
22. V. Shoup. *Practical Threshold Signatures.* EUROCRYPT 2000, LNCS vol.1807, pp.207–220, Springer 2000.