On-line/Off-line Threshold Signatures Without Trusted Dealers

Chris Crutchfield, David Molnar, David Turner, and David Wagner

University of California, Berkeley
{cyc, dmolnar, dbturner, daw}@cs.berkeley.edu

Abstract. We propose *on-line/off-line threshold signature schemes*, in which the bulk of signature computation can take place "off-line" during lulls in service requests [7]. Such precomputation can help systems using threshold signatures quickly respond to requests. For example, tests of the Pond distributed file system showed that computation of a threshold RSA signature consumes roughly 86% of the time required to service writes to small files [15]. Because a large number of writes in file systems are for small files [17], threshold signatures form a performance bottleneck in Pond and similar systems. We apply the "hash-sign-switch" paradigm of Shamir and Tauman [19] and the distributed key generation protocol of Gennaro et al. [8] to convert any existing secure threshold digital signature scheme into a threshold on-line/off-line signature scheme. Our construction is fully distributed and requires no trusted dealers. We show that the straightforward attempt at proving security of the resulting construction runs into a subtlety that does not arise for Shamir and Tauman's construction. We resolve the subtlety and prove our signature scheme secure against a static adversary in the partially synchronous communication model under the one-more-discrete-logarithm assumption [2]. The on-line phase of our scheme is efficient: computing a signature takes one round of communication and a few modular multiplications in the common case.

Keywords: On-line/Off-line, Signature Schemes, Threshold Cryptography, Chameleon Hash Functions, Bursty Traffic.

1 Introduction

We propose *on-line/off-line threshold signatures* to improve the performance of threshold signature schemes, and we show how to construct such signatures from existing threshold signature schemes. In a threshold signature scheme, given a group of n players, and a threshold $t < n$, no subset of the players of size at most t can generate a signature. In other words, unlike standard signature schemes — in which a single player must protect his or her secret key — at most t of the n players in a threshold signature scheme may be compromised without endangering the security of the signature scheme.

Threshold signatures have been applied in several areas to avoid concentrating trust in any single entity. For example, OceanStore [13, 15] is a large-scale distributed data storage system that requires the computation of threshold signatures by an "inner ring" of servers for performing a Byzantine agreement when writing a file. Latency tests in Pond [15], the OceanStore prototype, show that for a 4 KB write, 77.8 ms out of 90.2 ms total time to service the write operation is spent on computing Shoup's RSA threshold signature scheme [20]. Therefore, computation is the dominant factor; although network communication and local file system access contributes to the time, the bulk of the contribution to service time comes from computing the threshold signatures [15].

Optimizing threshold signature computation is particularly important for distributed file systems because small file writes are common [17]. For example, Baker et al. found that for a file trace from the Sprite file system, 80% of all sequential transfers were less than 2300 bytes in length [1]. For larger files in Pond (2 MB), there is little change in the time spent computing the threshold signature; instead, the time spent on writing the file dominates the threshold signature time. Even so, because threshold signature computation takes up 86% of the time to service a small write in Pond, optimizing this computation improves the common case. Threshold signatures have also been applied as part of other applications, such as distributed certificate authorities, so increasing their performance can help these applications as well [22].

Our Approach. In an on-line/off-line scheme [7], servers can perform the bulk of the computation in an *off-line phase* before even seeing the message to be signed. The results of this precomputation are saved and then used in the *on-line phase* when a message must be signed. Because distributed systems often have "bursty" traffic, resources are available for such precomputation. For example, during the day and evening, traffic is high, but during the night and morning, traffic is low. Enabling threshold signatures to be computed off-line allows systems such as OceanStore to build up a stockpile of precomputed values while traffic is low. These values can be used to quickly sign messages later when traffic is high. Furthermore, other distributed file systems have been observed to have bursty traffic [18, 21], and so they can enjoy the benefits of our on-line/off-line threshold signature scheme.

The main idea of our scheme is to apply the "hash-sign-switch" paradigm of Shamir and Tauman [19] to a threshold signature scheme. In this paradigm, we make use of a *chameleon hash function,* which is a special type of two-argument hash function $CH_{HK}(m, r)$ endowed with a public and secret key [11]. Knowledge of the public key HK allows one to evaluate the hash function, while knowledge of the secret key allows one to find collisions. Shamir and Tauman show that any standard signature scheme can be converted to an on-line/off-line scheme as follows: for the off-line phase, compute a standard signature on $CH_{HK}(a, r)$, where a and r are chosen randomly. Then, at the on-line phase, given the message m, use the secret key to find an r' such that $CH_{HK}(m, r') = CH_{HK}(a, r)$. The signature on $CH_{HK}(a, r)$ together with r' then forms a signature on the message m; in a sense, we "switch" m for the random value a. We refer to the signed value of $CH_{HK}(a, r)$ as the *signature stamp.* If finding a collision in the chameleon hash is more efficient than signing the message directly (as is the case for several chameleon hash functions), this is a net performance win.

Overview of Our Construction. For our work, we focus on the specific chameleon hash function $CH_{HK}(m,r) = g^r h^m \bmod p$ with public key $HK = (p, g, h)$ and the secret key y is the discrete logarithm of h to the base g. We show how to use the discrete logarithm distributed key generation algorithm of Gennaro et al. [8] to perform chameleon hash key generation and computation of the signature stamp. We then show an efficient distributed algorithm for finding collisions with low overhead per player. We stress that *no trusted dealer* is required by our scheme; given an underlying threshold signature scheme with distributed key generation and distributed signing algorithms, we obtain a fully distributed signature scheme.

We also show methods for guaranteeing the robustness of our scheme using zero-knowledge proofs for verification. We provide two variants. The first is non-interactive and secure in the Random Oracle Model. The second uses an observation of Damgård and Dupont to obtain robustness at the cost of limited interaction but is secure without random oracles [5]. In both cases, instead of running verification each time a signature must be generated, we decide to forego this step and be *optimistic* because, as observed in [5], the signature shares will be correct almost always. If the signature created is not valid, then we can run the verification procedure in order to expose the corrupted players. The full details for our signature scheme appear in Section 3.

A Subtlety In The Proof. Surprisingly, the straightforward adaptation of the proof of Shamir and Tauman for non-threshold on-line/off-line signature schemes *fails* to establish security for our new on-line/off-line threshold scheme. The subtlety is that in our scheme, the "signature stamp" value $CH_{HK}(m,r)$ is disclosed to all parties at the close of our off-line threshold phase, including the adversary. While m and r are not disclosed, the output of the chameleon hash must be broadcast to allow for "black-box" use of the underlying threshold signature scheme in creating the stamp. As a result, any attempt at simulating the adversary's view of a signature query is "pinned down" by the value of the chameleon hash encoded in the stamp. In contrast, Shamir and Tauman do not reveal any chameleon hash values associated with a message to the adversary until *after* a signing query for that message is made. Therefore, their reduction is not "pinned down" in the same way and can easily answer adversary signing queries by simply evaluating the chameleon hash function on the queried message. While this is not an attack on the threshold on-line/off-line scheme, it shows that a new idea appears necessary to prove the scheme secure.

We resolve this subtlety by first introducing a new assumption for chameleon hash functions, which we call the *one-more-r assumption.* Informally, the new assumption says that given a sequence of random "challenge" outputs $v_1, \ldots, v_n$ of the chameleon hash function, the adversary may adaptively pick values v_i, provide messages m_i, and then learn r_i such that $CH_{HK}(m_i, r_i) = v_i$. Then, even given this extra information, the adversary has negligible advantage at inverting the chameleon hash on any challenge value not picked. We show that this new assumption is sufficient to prove security of our scheme. Then we justify the assumption in the case of the $g^r h^m \bmod p$ chameleon hash by showing it is implied by the *one-more discrete logarithm* assumption of Bellare et al [2]. This establishes the security of our scheme based on a standard assumption. The details for showing our scheme is existentially unforgeable and robust against a static adversary are in Section 5 and the Appendix.

Performance Results. We analyze the performance of our scheme in Section 6. We show the cost of our off-line phase is dominated by the cost of the distributed discrete logarithm key generation protocol. While our off-line phase in consequence requires several rounds of communication and computation, we argue that this overhead uses resources that would otherwise sit idle. If a new request arrives at a server during a busy time, the servers can simply fall back to directly computing a threshold signature.

Finally, we show that our optimistic on-line phase obtains a factor of $\mathcal{O}\left(\frac{k}{t}\right)$ improvement in computation compared to Shoup's RSA threshold signature scheme, where k is a security parameter, while also requiring only one round of communication [20]. For example, with the parameters suggested for Pond, this is a factor of 1024 improvement. Therefore, we believe our scheme will yield in many cases much improved performance for any distributed system that currently uses threshold signature schemes.

1.1 Previous Work

The first on-line/off-line signature scheme was developed by Even, Goldreich, and Micali [7]. This scheme allowed for the conversion of any standard signature scheme into a one-time on-line/off-line signature scheme. Their result, however, increased the size of the signature by a quadratic factor. In order to mitigate this, Shamir and Tauman [19] applied the results of Krawczyk and Rabin [11], using chameleon hash functions to construct a one-time on-line/off-line signature scheme that only increases the size of the signature by a factor of two. Although smart cards appear to be an important application of on-line/off-line signatures as noted in [7, 19], the application to bursty traffic has received little attention.

The origins of threshold signatures and threshold cryptography can be traced back to Desmedt and Frankel [6]. Some examples of threshold signatures include a robust threshold DSS signature scheme by Gennaro, *et al.* [9], and a robust, non-interactive threshold RSA signature scheme by Shoup [20]. The latter construction is the signature scheme implemented in Pond [15], a prototype version of the OceanStore [13] design, and partly our motivation for this paper.

1.2 Our Results

We compare our optimistic on-line/off-line threshold signature scheme with that of Shoup's signature scheme [20]. Shoup describes two variants of an RSA thresh-

Threshold Sig. Schemes:	Shoup's RSA Scheme	Our On-line/Off-line Scheme
Key Generation	$\mathcal{O}(k^2 nt \log t + k^3) + K_{RSA}$	$K_{On/Off} + K_{DKG}$
Off-line Phase	None	$3K_{DKG} + \mathcal{O}(k^2) + \tau$
On-line Player	$\mathcal{O}(k^3)$	$\mathcal{O}(k^2)$
On-line Reconstruction	$\mathcal{O}(tk^3)$	$\mathcal{O}(t^2k^2)$
On-line Rounds of Comm.	1	1

Table 1. Comparison between Shoup's Threshold RSA and our On-line/Off-line Threshold Scheme.

old signature scheme, and it is the first variant that we compare our scheme against. In both schemes, let n be the number of players, $t < \frac{n}{3}$ be the threshold[1], and $k \in \mathbb{N}$ be a security parameter. Our construction requires $2t+1$ players to construct a signature and tolerates the participation of at most t corrupted players. We analyze the bit complexity of both schemes using the following metrics and show the results in Table 1:

- Key Generation Complexity — Work done to perform key generation and distributing private key shares among the players. Let K_{RSA} denote the bit complexity for generating the RSA public and private keys, let $K_{On/Off}$ denote the bit complexity for generating public and private keys in our scheme, and let K_{DKG} denote the bit complexity for distributed key generation.
- Off-line Phase Complexity — Work done to perform precomputation, meaning the computation performed for a signature before a message arrives. Furthermore, let τ be the bit complexity for generating a standard threshold signature.
- On-line Player Complexity — Work done by a player in computing its signature share when a message arrives. Note that all players compute their signature share in parallel.
- On-line Reconstruction Complexity — Work done by the players in combining all of the signature shares and creating a signature.
- On-line Rounds of Communication — Number of rounds the players need to generate a signature.

Note that Shoup's RSA signature scheme, as well as other threshold signature schemes, is not considered to be an on-line/off-line scheme because no precomputation is performed. Furthermore, an optimistic version of Shoup's scheme does not reduce its asymptotic complexity in the on-line phase. Finally, referring to Table 1, we see that both schemes only require one round of communication because all of the members of the group do not have to wait for each other when a message m arrives; instead, they can immediately compute their signature shares for m. Because we can set the modulus in both schemes to be of the same size, we can compare fairly based on the bit complexity. A more complete analysis that includes robustness can be found in Section 6.

[1] Shoup's RSA threshold signature scheme can actually tolerate a threshold of $t < \frac{n}{2}$ and only needs $t+1$ players to generate a signature.

2 Preliminaries

We first define important terms used in this paper, followed by a review of the Shamir-Tauman on-line/off-line signature scheme.

2.1 Definitions

Definition 1 (Negligible Function). A function $\eta : \mathbb{N} \to \mathbb{R}$ is negligible if for all $c > 0$, $\eta(k) < \frac{1}{k^c}$ for all sufficiently large k.

Definition 2 (Negligible Probability). An event E occurs with negligible probability if its probability of occurring is given by some negligible function.

Definition 3 (Discrete Logarithm Assumption). Let $p = 2q+1$ be a prime where q is a random k-bit prime and let g be a generator for a subgroup of $\mathbb{Z}_p^*$ with order q. For all probabilistic polynomial time algorithms A, if x is chosen uniformly at random from $\mathbb{Z}_q$ and $h = g^x \pmod p$, then $\Pr[\mathsf{A}(p, q, g, h) = x] \leq \eta(k)$, where η is a negligible function.

Definition 4 (Lagrange Interpolating Polynomial). Let $\mathcal{F}$ be a field and let $p(x) \in \mathcal{F}[x]$ be a degree t polynomial. Given $t+1$ distinct points in $\mathcal{F}^2$, $(x_1, p(x_1)), (x_2, p(x_2)), \ldots, (x_{t+1}, p(x_{t+1}))$, the goal is to reconstruct $p(x)$. Then the Lagrange interpolating polynomial passing through these $t+1$ points is

$$f(x) = \sum_{i=1}^{t+1} p(x_i) f_i(x), \text{ where } f_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{t+1} \frac{x_j - x}{x_j - x_i}$$

and by uniqueness, $f(x) = p(x)$.

Definition 5 (Chameleon Hash Function). Given a public key HK and a private key or *trapdoor* TK, a message $m \in \mathcal{M}$, and a random $r \in \mathcal{R}$ where $\mathcal{M}$ is the message space, and $\mathcal{R}$ is some finite space, we denote a chameleon hash function [11] by $CH_{HK}(m, r)$, which is a hash function with the following properties:

– **Collision Resistance**. Given any probabilistic polynomial time malicious entity $\mathcal{A}$ that does not know the private key TK, but only the public key HK, define its *advantage* to be the probability of finding (m_1, r_1) and (m_2, r_2) such that $CH_{HK}(m_1, r_1) = CH_{HK}(m_2, r_2)$. We require the advantage of $\mathcal{A}$ to be negligible.
– **Trapdoor Collisions**. There exists a polynomial time algorithm A such that on inputs the pair (HK, TK), a pair $(m_1, r_1) \in \mathcal{M} \times \mathcal{R}$, and a message $m_2 \in \mathcal{M}$, then A outputs r_2 such that $CH_{HK}(m_1, r_1) = CH_{HK}(m_2, r_2)$.
– **Uniform Probability Distribution**. If $r_1 \in \mathcal{R}$ is distributed uniformly, $m_1 \in \mathcal{M}$, and $(m_2, r_2) \in \mathcal{M} \times \mathcal{R}$ such that $CH_{HK}(m_1, r_1) = CH_{HK}(m_2, r_2)$, then r_2 is computationally indistinguishable from uniform over $\mathcal{R}$.

Throughout the rest of this paper, we will work with a particular family of chameleon hash functions based on discrete logarithms. We do so because the discrete logarithm-based hash function is best suited for using Lagrange interpolation. There are also other chameleon hash functions, such as those based on factoring, for example, but the mathematics involved in the interpolation would not be as convenient.

We begin by picking a Sophie-Germain prime, $p' \in \mathbb{N}$, which has the property that $p = 2p' + 1$ and p' are both primes. Although it is not known if there are infinitely many Germain primes, we will assume that we can find one of the appropriate size. Let g' be a generator for $\mathbb{Z}_p^*$. Now let $Q_p \subset \mathbb{Z}_p^*$ denote the subgroup of quadratic residues generated by $g \equiv (g')^2 \pmod{p}$, so that $|Q_p| = \frac{p-1}{2} = p'$. Finally, pick the private key $y \in \mathbb{Z}_{p'}^*$. Then we define our chameleon hash function $CH_{HK} : \mathbb{Z}_{p'} \times \mathbb{Z}_{p'} \to Q_p$ to be

$$CH_{HK}(m, r) = g^{r+ym} \equiv g^r h^m \pmod{p}$$

where $h \equiv g^y \pmod{p}$ and the public key is $HK = (p, g, h)$. The above three properties are easily verified as follows. If the private key y is not known, then this family of chameleon hash functions is collision resistant because if there exists a probabilistic polynomial time algorithm that succeeds with non-negligible probability in finding pairs (m, r) and (m', r') in $\mathbb{Z}_{p'} \times \mathbb{Z}_{p'}$ such that $m \neq m'$, then the discrete logarithm of h to the base g is given by $y \equiv (r - r')(m' - m)^{-1} \pmod{p'}$. This contradicts the discrete logarithm assumption. With knowledge of the secret key y, it is easy to find r' such that $CH_{HK}(m', r') \equiv CH_{HK}(m, r) \pmod{p}$ because given any message m' and the known pair (m, r), set $r' \equiv r + ym - ym'$ $\pmod{p'}$, and the result follows. This also verifies the uniform probability distribution property because if $r \in \mathbb{Z}_{p'}$ is distributed uniformly at random, then r' is also distributed uniformly at random.

Definition 6 (Signature Scheme). A signature scheme $\mathcal{S}$ is a triple of randomized algorithms (Key-Gen, Sig, Ver) where:

- Key-Gen: $1^* \to \mathcal{PK} \times \mathcal{SK}$ is a key generation algorithm such that on input 1^k, where $k \in \mathbb{N}$ is a security parameter, it outputs (PK, SK), such that $PK \in \mathcal{PK}$, the set of all public verification keys, and $SK \in \mathcal{SK}$, the set of all secret keys.
- Sig : $\mathcal{SK} \times \mathcal{M} \to \mathcal{SIGS}$ is a signing algorithm such that $\mathcal{M}$ is the message space and $\mathcal{SIGS}$ is the signature space. For shorthand, let $S_{SK}(m) = \mathsf{Sig}(SK, m)$ for all $m \in \mathcal{M}$.
- Ver : $\mathcal{PK} \times \mathcal{M} \times \mathcal{SIGS} \to \{\mathsf{Reject}, \mathsf{Accept}\}$ is a verification algorithm such that $\mathsf{Ver}(PK, m, \sigma) = \mathsf{Accept}$ if and only if σ is a possible output of $\mathsf{Sig}(SK, m)$. Again, for shorthand, let $V_{PK}(m, \sigma) = \mathsf{Ver}(PK, m, \sigma)$ for all $m \in \mathcal{M}$ and $\sigma \in \mathcal{SIGS}$.

Definition 7 (Threshold Signature Scheme). Given a signature scheme $\mathcal{S} =$ (Key-Gen, Sig, Ver), a threshold signature scheme $\mathcal{TS}$ for $\mathcal{S}$ is a triple of randomized algorithms (Thresh-Key-Gen, Thresh-Sig, Ver) for a set of n players $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ with threshold value t where:

- Thres-Key-Gen is a distributed key generation algorithm used by the players to create $(PK, SK) \in \mathcal{PK} \times \mathcal{SK}$ such that each $P_i \in \mathcal{P}$ receives a share SK_i of the secret key SK.
- Thresh-Sig is a distributed signing algorithm used by the players to create a signature for a message $m \in \mathcal{M}$ such that the output of the algorithm is $S_{SK}(m)$. This algorithm can be decomposed into two algorithms: signature share generation and signature reconstruction.

In this paper, we assume that $\mathcal{TS}$ is *simulatable*, as defined in Gennaro, Jarecki et al. [9]. This means that there exists a simulator $\mathsf{SIM}_1^{\mathcal{TS}}$ which, on input PK, simulates the view of the adversary for a run of Thresh-Key-Gen that fixes the public key to be PK. In addition, there exists a simulator $\mathsf{SIM}_2^{\mathcal{TS}}$ for Thresh-Sig, such that on input: the public key PK, the message v, the signature σ of v, and the key shares $x_{i_1}, x_{i_2}, \ldots, x_{i_t}$ of the servers controlled by the adversary, simulates the view of the adversary for a run of Thresh-Sig on v that produces the signature σ.

Definition 8 (On-line/Off-line Signature Scheme). An on-line/off-line signature scheme is a method for signing messages in two phases: an on-line phase and an off-line phase. Most of the computation is done in the off-line phase, which happens before seeing the message to be signed. When the message m finally arrives, the on-line phase takes the precomputed value from the off-line phase and uses it to sign m. This particular signature scheme is useful because it splits the heavy cost of signing (modular exponentiations) from the actual action of signing, without adding much overhead.

Definition 9 (Signature Stamp). In an on-line/off-line signature scheme, we call the precomputed signature from the off-line phase a signature stamp.

Definition 10 (Distributed Key Generation). A Distributed Key Generation (DKG) protocol is often used in threshold signature schemes in order to construct the public key and private key. In a DKG protocol with n players, the public key is made known to all players, whereas the private key is known by none. Instead, each player receives a *key share*, from which they can — acting in concert — recover the private key. A DKG protocol is, of course, fully distributed, and requires no trusted dealer.

In this paper, we use a discrete logarithm-based DKG protocol (where the private key is y and the public key is $h = g^y$ for some g), namely the New-DKG protocol as defined by Gennaro et al. [8]. This protocol has the property that there exists a simulator SIM^{DKG} that on input h can simulate the interactions of the DKG protocol with a set $\mathcal{B}$ (where $|\mathcal{B}| \leq t$) of players controlled by the adversary, such that the resulting public key produced is fixed to be h. In addition, as a result of this simulation, SIM^{DKG} is able to recover the key shares held by the adversary's players, $\mathcal{B}$.

2.2 The Shamir-Tauman On-line/Off-line Signature Scheme

Before discussing how to achieve an on-line/off-line threshold signature scheme, we review the single-player scheme, as defined by Shamir and Tauman [19], which introduces the *hash-sign-switch* paradigm. Let $\mathcal{S}$ be an arbitrary signature scheme. We show how to make $\mathcal{S}$ on-line/off-line, which we denote by $\mathcal{S}^{On/Off}$:

1. **Key Generation (done once):** On input 1^k, where $k \in \mathbb{N}$ is a security parameter, run Key-Gen for $\mathcal{S}$ to obtain a pair $(PK, SK) \in \mathcal{PK} \times \mathcal{SK}$. As for our chameleon hash function, we can run its key generation procedure and obtain a public key HK and trapdoor TK. The secret key is now (SK, TK), while the public key is (PK, HK).
2. **Off-line Phase (done per message):** First choose uniformly at random $(m, r) \in \mathcal{M} \times \mathcal{R}$. Now we sign $CH_{HK}(m, r)$, and we store the pair (m, r), the hash value $CH_{HK}(m, r)$, and the signature stamp $S_{SK}(CH_{HK}(m, r))$. This step corresponds to the traditional *hash-sign* paradigm.
3. **On-line Phase (done per message):** We are now given some message $m' \in \mathcal{M}$ that we want to sign. To do this, we first recover from memory the pair (m, r), the hash value $CH_{HK}(m, r)$, and the stamp $S_{SK}(CH_{HK}(m, r))$. Since trapdoor collisions can be found efficiently for a chameleon hash function given the secret key, we can now quickly find an $r' \in \mathcal{R}$ such that $CH_{HK}(m, r) = CH_{HK}(m', r')$. The signature for the message m' is the tuple $(S_{SK}(CH_{HK}(m, r)), m', r')$. This introduces the *switch* step in the new paradigm.
4. **Verification (done per message):** In order to verify that (σ, m', r') is indeed a valid signature for m', where $\sigma \in \mathcal{SIGS}$, simply check the value of $V_{PK}(CH_{HK}(m', r'), \sigma)$ and confirm that it equals to Accept.

Provided that CH_{HK} and the underlying signature scheme $\mathcal{S}$ are secure, Shamir and Tauman [19] proved $\mathcal{S}^{On/Off}$ is secure against adaptive chosen message attack.

3 An On-line/Off-line Threshold Signature Scheme

We shall construct an optimistic on-line/off-line threshold signature scheme $\mathcal{TS}^{On/Off}$ = (On/Off-Thresh-Key-Gen, Thresh-Sig-Off-line, Thresh-Sig-On-line, Ver) that does not require the use of a trusted dealer, and we show how existing threshold signature schemes, such as Shoup's threshold RSA [20] or Robust threshold DSS signatures [9], can be used in performing a threshold computation of the signature stamp off-line. Furthermore, we use the New-DKG from Gennaro, *et al* [8].

3.1 Key Generation (Done once)

On/Off-Thresh-Key-Gen

Inputs: A threshold signature scheme $\mathcal{TS}$ = (Thresh-Key-Gen, Thresh-Sig, Ver), a set of n players $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$, a threshold $t < \frac{n}{3}$, and a security parameter $k \in \mathbb{N}$.
Public Output: A set of public keys.
Private Output: All players $P_i \in \mathcal{P}$ receive a set of private keys.

1. Run Thresh-Key-Gen on input 1^k to obtain $(PK, SK) \in \mathcal{PK} \times \mathcal{SK}$ and each $P_i \in \mathcal{P}$ receives the secret key share SK_i.
2. Create a random k bit Germain prime $p' \in \mathbb{N}$, where $p = 2p' + 1$ is also a prime, and let g be a generator for Q_p.
3. Use the DKG protocol to create $h = g^y$, where $y \in \mathbb{Z}_{p'}$ is the secret key and $P_i \in \mathcal{P}$ receives the share y_i for a degree t polynomial $p_y(x) \in \mathbb{Z}_{p'}[x]$ such that $p_y(0) = y$.
4. Check that $n < p'$ so that each player $P_i \in \mathcal{P}$ has index $i \in \mathbb{Z}_{p'}^*$. Otherwise abort.
5. Publish the public keys $(PK, HK = (p, g, h))$. All players $P_i \in \mathcal{P}$ retain (SK_i, y_i).

3.2 Off-line Phase (Done per message)

In the off-line phase, we will show how to construct the chameleon hash function and create the signature stamp in a distributed manner.

Thresh-Sig-Off-line

Inputs: The same set of n players $\mathcal{P}$ and a threshold $t < \frac{n}{3}$.
Private Output: A signature stamp.

1. Use the DKG protocol to create g^r, where $r \in \mathbb{Z}_{p'}$ so that P_i receives the share r_i for another degree t polynomial $p_r(x) \in \mathbb{Z}_{p'}[x]$ such that $p_r(0) = r$.
2. Use the DKG protocol to create h^m where $m \in \mathbb{Z}_{p'}$. Each player P_i receives a share m_i for a degree t polynomial $p_m(x) \in \mathbb{Z}_{p'}[x]$ such that $p_m(0) = m$.
3. Finally, the DKG protocol is used to generate shares z_i for each $P_i \in \mathcal{P}$ of a degree $2t$ polynomial $p_0(x) \in \mathbb{Z}_{p'}[x]$ such that $p_0(0) = 0$.
4. Now g^r and h^m are both known to the players, so $CH_{HK}(r, m) = g^r h^m \pmod{p}$.
5. Use Thresh-Sig to compute the signature stamp $S_{SK}(CH_{HK}(r, m))$.

3.3 On-line Phase (Done per message)

Thresh-Sig-On-line

Inputs: A subset $\mathcal{P}' \subset \mathcal{P}$ of size $2t+1$ and a message $m' \in \mathbb{Z}_{p'}$.
Public Output: A signature for m'.

1. For each $P_i \in \mathcal{P}'$, define $\mathsf{col\text{-}1}_i = r_i - y_i m'$ and $\mathsf{col\text{-}2}_i = y_i m_i + z_i$, which are P_i's share of the trapdoor collision. Then, P_i broadcasts the pair $(\mathsf{col\text{-}1}_i, \mathsf{col\text{-}2}_i)$ to all of the other players in $\mathcal{P}'$.
2. Define $f_i(x)$ to be $f_i(x) = \prod_{P_j \in \mathcal{P}' \setminus \{P_i\}} \frac{j-x}{j-i}$, as in the definition of Lagrange interpolation. Now use Lagrange interpolation on the shares to compute the trapdoor collision
$$\begin{aligned} r' &= \sum_{P_i \in \mathcal{P}'} (\mathsf{col\text{-}1}_i + \mathsf{col\text{-}2}_i) f_i(0) \\ &= \sum_{P_i \in \mathcal{P}'} (r_i + y_i m_i + z_i - y_i m') f_i(0) \\ &\equiv r + ym - ym' \pmod{p'}. \end{aligned}$$
3. In this way, the signature for message m' is
$$(S_{SK}(CH_{HK}(m, r)), m', r').$$

Notice that the definition of $\mathsf{col\text{-}2}_i$ requires adding the share z_i. This is necessary because we have to multiply the secrets y and m, so each player computes $y_i m_i$ which becomes a share of a degree $2t$ polynomial that is not chosen uniformly at random; thus, adding the share z_i will make the polynomial random. Furthermore, this degree $2t$ polynomial is the reason for requiring $t < \frac{n}{3}$.

3.4 Verification (Done per message)

Given the signature (σ, m', r'), where $\sigma \in \mathcal{SIGS}$, simply check that

$$V_{PK}(CH_{HK}(m', r'), \sigma) = \mathsf{Accept}$$

holds true, as in the standard signature scheme. Notice that verification requires two modular exponentiations for computing $CH_{HK}(m', r')$, as well as any possible exponentiations for V_{PK}. This burden, however, is placed onto anybody running V_{PK} and not the players involved in creating a signature. For systems in which clients make relatively few requests compared to the number served by threshold signature servers, this is an acceptable tradeoff.

3.5 Signature Share Verification (Performed if necessary)

If $V_{PK}(CH_{HK}(m', r'), \sigma) = \mathsf{Reject}$, then some players are sending incorrect shares. In order to ensure robustness, we must be able to construct a valid signature. The naïve solution of trying all possible subsets of size $2t+1$ to construct a valid signature is unacceptable because there are an exponential number of such subsets. Instead, we will identify and remove the corrupted players. To do so, we have each player in $\mathcal{P}$ check the validity of the pair $(\mathsf{col\text{-}1}_i, \mathsf{col\text{-}2}_i)$ for each player $P_i \in \mathcal{P}'$:

1. **Verifying** $\mathsf{col\text{-}1}_i$. Because g^{r_i} and g^{y_i} are known values from the DKG protocol, we can compute for each $P_i \in \mathcal{P}'$, $g^{r_i} \cdot (g^{y_i})^{-m'} = g^{r_i - y_i m'} \pmod p$ and confirm that $g^{\mathsf{col\text{-}1}_i} = g^{r_i - y_i m'}$ as desired.
2. **Verifying** $\mathsf{col\text{-}2}_i$. Although we have access to g^{z_i} from the DKG protocol, we do not have $g^{y_i m_i}$. Instead, what we will do is confirm that the discrete logarithm of $g^{\mathsf{col\text{-}2}_i} g^{-z_i} = g^{\mathsf{col\text{-}2}_i - z_i}$ to the base g^{m_i} is equal to the discrete logarithm of g^{y_i} to the base g. Now we can apply Chaum and Pedersen's ZKP for equality of discrete logarithms [4] with the Fiat-Shamir heuristic: Let $d = g^{y_i}, e = g^{m_i}$, and $f = g^{\mathsf{col\text{-}2}_i - z_i}$. Player $P_i \in \mathcal{P}'$ chooses $r \in \mathbb{Z}_{p'}$ uniformly at random and computes $H(g, d, e, f, g^r, e^r) = c$, where H is a random oracle and c is the challenge. P_i computes $v = y_i c + r$ and broadcasts the pair (c, v). Finally, all players compute and confirm that $H(g, d, e, f, g^v d^{-c}, e^v f^{-c}) = c$.

If any of the shares are deemed incorrect, then broadcast a *complaint* against P_i. If there are at least $t+1$ complaints, then clearly P_i must be corrupt since with at most t malicious players, there can be at most t false complaints. Also, if P_i is corrupt, there will always be enough honest players to generate at least $t+1$ complaints and P_i will surely be disqualified in this case. Once eliminated, P_i is removed from $\mathcal{P}'$ and is replaced with a new player, thus resulting in a new signature. As long as at most t players are corrupted, there will always be enough honest players to create a valid signature.

3.6 Remarks

First we justify our use of the chameleon hash function $CH_{HK}(m, r) \equiv g^r h^m \pmod p$. Let us do this by looking at how one would compute the value of h from g and the shares y_i given to each player, if we had a subset $\mathcal{P}' \subset \mathcal{P}$ of size $t+1$. If $P_i \in \mathcal{P}'$, then let $h_i = g^{y_i f_i(0)}$ and then

$$h = \prod_{P_i \in \mathcal{P}'} h_i = \prod_{P_i \in \mathcal{P}'} g^{y_i f_i(0)} \equiv g^{\sum_{P_i \in \mathcal{P}'} y_i f_i(0)} \equiv g^{p_y(0)} = g^y \pmod p.$$

The interesting step is the Lagrange polynomial interpolation. Recall that g has order p', so in fact we are interpolating over the finite field $\mathbb{Z}_{p'}$. Thus, the chameleon hash function $CH_{HK}(m, r)$ has the property that the interpolation happens in the exponents and succeeds because g has prime order.

Second, observe that the secret key y is never explicitly revealed to the adversary because each $P_i \in \mathcal{P}'$ only makes public $\mathsf{col\text{-}1}_i$ and $\mathsf{col\text{-}2}_i$. Since r_i, m_i and z_i are shares that are kept private by each player, it is not possible for the adversary to uncover the value of y_i unless it has corrupted player P_i in order to have enough shares (at least $t+1$) to reconstruct y.

Third, notice that new signature stamps must be computed for every message to be signed, and hence this is a one-time threshold signature scheme. If a stamp is reused, then for any two messages m_1 and m_2 along with their respective r_1 and r_2, then $CH_{HK}(m_1, r_1) = CH_{HK}(m_2, r_2)$, and one can immediately compute the secret $y \equiv (r_1 - r_2)(m_2 - m_1)^{-1} \pmod{p'}$. Since we are assuming a bursty traffic model throughout this paper, we can run the off-line phase during periods of low traffic to compute more stamps.

Finally, we note two extensions to our scheme. One is that we can use batching techniques in order to improve performance. For instance, Krohn *et al.* explore this in the context of verifying erasure codes authenticated with a similar hash function [12]. A second extension is that the random oracle can be eliminated from the verification procedure. We further discuss these extensions in Appendix D.

4 Security Model

4.1 Security Definitions

Definition 11 (Random Oracle Model). In the **random oracle model** [3], all players, including the adversary, have access to a random oracle H, which implements a truly random function. In practice, an algorithm such as SHA-1 is used in constructions designed to approximate a random oracle.

We define two assumptions that we will use in our proof. The first is the one-more-discrete-logarithm assumption introduced by Bellare et al. [2]

Definition 12 (One-More-Discrete-Logarithm Assumption). Let $p = 2q+1$ be a prime where q is a random k-bit prime and let g be a generator for a subgroup of $\mathbb{Z}_p^*$ with order q. We let $n : \mathbb{N} \to \mathbb{N}$ be a function of k. Now let $\{x_1, x_2, \ldots, x_{n(k)}, x_{n(k)+1}\}$ be elements of $\mathbb{Z}_q$ chosen uniformly at random, and for each $i \in \{1, 2, \ldots, n(k)+1\}$, define $z_i = g^{x_i} \pmod{p}$. Now let the adversary $\mathcal{A}$ have access to a discrete log oracle DLog such that if $x \in \mathbb{Z}_q$, $z = g^x \pmod{p}$, then $\mathsf{DLog}(g, z) = x$. In the one-more discrete-logarithm problem [2], $\mathcal{A}^{\mathsf{DLog}}$ is given $\{z_1, z_2, \ldots, z_{n(k)+1}\}$ and must output $\{x_1, x_2, \ldots, x_{n(k)+1}\}$ by querying DLog at most $n(k)$ times. The assumption is that $\Pr[\mathcal{A}^{\mathsf{DLog}}(g, z_1, z_2, \ldots, z_{n(k)+1}) = (x_1, x_2, \ldots, x_{n(k)+1})] \leq \eta(k)$, where η is a negligible function.

We define a similar assumption that is related to finding collisions in a chameleon hash function. We will use this assumption to show our new scheme is secure. In Appendix C, we show that this assumption is implied by one-more-discrete-logarithm for the chameleon hash function we use.

Definition 13 (One-More-R Assumption). As above, we let g be a generator for a subgroup of $\mathbb{Z}_p^*$ with order q, a k-bit prime. In addition, we let k' be randomly chosen from $\mathbb{Z}_q$ and let $h = g^{k'}$. We let $n : \mathbb{N} \rightarrow \mathbb{N}$ be a function of k. Now let $\{v_1, v_2, \ldots, v_{n(k)}, v_{n(k)+1}\}$ be randomly chosen elements in the range of $CH_{HK}(\cdot)$. Now we give the adversary $\mathcal{A}$ access to a $\mathsf{Get\text{-}An\text{-}R}(v, m)$ oracle, such that if $r = \mathsf{Get\text{-}An\text{-}R}(v, m)$, then $CH_{HK}(m, r) = v$. In the One-More-R problem, $\mathcal{A}^{\mathsf{Get\text{-}An\text{-}R}}$ is given $\{v_1, v_2, \ldots, v_{n(k)+1}\}$ and with at most $n(k)$ queries to $\mathsf{Get\text{-}An\text{-}R}$, must output $\{(m_1, r_1), (m_2, r_2), \ldots, (m_{n(k)+1}, r_{n(k)+1})\}$ such that $v_i = CH_{g,k}(m_i, r_i)$. The assumption is that $\Pr[\mathcal{A}^{\mathsf{Get\text{-}An\text{-}R}}(g, h, v_1, v_2, \ldots, v_{n(k)+1}) = \{(m_1, r_1), (m_2, r_2), \ldots, (m_{n(k)+1}, r_{n(k)+1})\}] \leq \eta(k)$, where η is a negligible function.

4.2 Adversarial Model

We assume that there is a static adversary $\mathcal{A}$ that corrupts some subset of the players in $\mathcal{P}$ before beginning the threshold signature scheme. Furthermore, we can analyze two different types of static adversaries: one that compromises before the off-line phase and the other compromises after the off-line phase terminates. We assume the former case in our proof of existential unforgeability. As for the communication model, we assume that all players are connected by secure point-to-point channels. Furthermore, we will assume a partially synchronous communication model during the key generation and off-line phases for the purpose of using the DKG protocol of Gennaro et al. [8]

5 Proof of Security

5.1 Robustness

Theorem 1. *Suppose that an adversary corrupts at most $t < \frac{n}{3}$ players. Then, our on-line/off-line threshold signature scheme is **robust**.*

Proof. The full details of the proof appear in Appendix B. □

5.2 Existential Unforgeability

Theorem 2. *Suppose that a static adversary corrupts at most t players before beginning the off-line phase. Then our threshold signature scheme is **existentially unforgeable** assuming that the underlying signature scheme $\mathcal{S}$ is also existentially unforgeable.*

Proof. The full details of the proof appear in Appendix C. □

6 Evaluation

We analyze the number of bit operations required by our scheme, as previously shown in Table 1. First, in our scheme, is the threshold key generation. The bit complexity of $\mathsf{Thresh\text{-}Key\text{-}Gen}$ for $\mathcal{TS}$, as well as generating a Germain prime is

Our On-line Phase Complexity	Additions	Multiplications	Exponentiations
Player Signature Share	2	2	0
Signature Reconstruction	$4t^2 + 6t + 1$	$4t^2 + 4t + 1$	0
Signature Share Verification	0	3	6

Table 2. Our On-line Phase Computational Complexity.

included in $K_{On/Off}$. Afterwards, we invoke the DKG protocol once, so this costs K_{DKG}, so the entire threshold key generation phase takes $K_{On/Off} + K_{DKG}$ bit operations.

Next, we analyze our off-line phase. First, we invoke the DKG protocol three times, so this gives $3K_{DKG}$. Next, we have g^r and h^m, so we multiply both terms to get $CH_{HK}(r, m)$. Moreover, a single multiplication requires $\mathcal{O}(k^2)$ bit operations over $\mathbb{Z}_p$. Finally, the signature stamp $S_{SK}(CH_{HK}(m, r))$ requires τ bit operations. Thus the off-line phases requires a total of $3K_{DKG} + \mathcal{O}(k^2) + \tau$ bit operations.

Finally, for our on-line complexity, we can separate a player's computational complexity for generating a signature share from the signature reconstruction complexity. Each player $P_i \in \mathcal{P}'$ performs only two addition and two multiplications when computing $\textsf{col-1}_i$ and $\textsf{col-2}_i$. The on-line signature reconstruction requires computing $f_i(0)$, which is $2t$ multiplications, and this is done for all $P_i \in \mathcal{P}'$, so we have a total of $(2t + 1)^2$ multiplications when we compute r'. Only addition of the $2(2t+1)$ shares as well as performing $2t$ subtractions when computing $f_i(0)$ is required giving a total of $(2t)(2t+1)+2(2t+1)-1 = 4t^2+6t+1$ additions. Furthermore, each addition over $\mathbb{Z}_p$ requires $\mathcal{O}(k)$ bit operations. Already we see that the number of multiplications in the on-line phase is substantially fewer than k since the threshold t is quite small when compared to a k bit prime. If verification of the signature shares is required, then each signature share requires six modular exponentiations. A summary of the number of operations performed in the on-line phase, as well as signature share verification complexity, appears in Table 2.

We also review the complexity of Shoup's RSA threshold signature scheme [20], which was also shown in Table 1. The key generation phase of Shoup's signature scheme requires a trusted party, but asymptotically the computation cost is the same as our distributed key generation. In Shoup's on-line phase, the reconstruction complexity, once again, can be separated from the share verification complexity. The reconstruction of the signature requires t modular exponentiations, $t-1$ modular multiplications, and one invocation of the extended Euclidean algorithm. Finally, verifying an individual signature share also requires six modular exponentiations and three modular multiplications. Although both Shoup's and our threshold signature schemes have approximately the same signature share verification complexity, we have managed to avoid any modular exponentiations in the reconstruction complexity of our signature scheme.

References

[1] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout. Measurements of a distributed file system. In *SOSP*, 1991.

[2] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme. *Journal of Cryptology*, 16(3):185–215, 2003.

[3] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM CCS*, pages 62–73, 1993.

[4] David Chaum and Torben P. Pedersen. Wallet Databases with Observers. In *CRYPTO*, pages 89–105, 1992.

[5] Ivan Damgård and Kasper Dupont. Efficient Threshold RSA Signatures with General Moduli and No Extra Assumptions. In *PKC*, pages 346–361, 2005.

[6] Yvo Desmedt and Yair Frankel. Threshold Cryptosystems. In *CRYPTO*, 1989.

[7] Shimon Even, Oded Goldreich, and Silvio Micali. On-Line/Off-Line Digital Schemes. In *CRYPTO*, pages 263–275, 1989.

[8] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure Distributed Key Generation for Discrete Logarithm Cryptosystems. *Journal of Cryptology*. To appear.

[9] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust Threshold DSS Signatures. In *EUROCRYPT*, pages 354–371, 1996.

[10] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive Secret Sharing Or: How to Cope With Perpetual Leakage. In *CRYPTO*, pages 339–352, 1995.

[11] Hugo Krawczyk and Tal Rabin. Chameleon Signatures. In *Proceedings of NDSS*, pages 143–154, 2000.

[12] Maxwell N. Krohn, Michael J. Freedman, and David Mazières. On-the-Fly Verification of Rateless Erasure Codes for Efficient Content Distribution. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.

[13] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of ASPLOS*, Novemeber 2000.

[14] Ralph Merkle. Protocols for Public Key Cryptosystems. In *IEEE Symposium on Security and Privacy*, pages 122–134, April 1980.

[15] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Pond: The OceanStore Prototype. In *Proceedings of USENIX FAST*, 2003.

[16] Sean Rhea and John Kubiatowicz. The OceanStore Write Path. `http://roc.cs.berkeley.edu/retreats/summer_02/slides/srhea.pdf`, June 2002.

[17] Mendel Rosenblum and John K. Ousterhout. The Design and Implementation of a Log-Structured File System. In *ACM Transactions on Computer Systems*, volume 10, pages 26–52, February 1992.

[18] Chris Ruemmler and John Wilkes. UNIX Disk Access Patterns. In *USENIX Winter 1993 Conference Proceedings*, January 1993.

[19] Adi Shamir and Yael Tauman. Improved Online/Offline Signature Schemes. In *CRYPTO*, 2001.

[20] Victor Shoup. Practical Threshold Signatures. In *EUROCRYPT*, pages 207–220, 2000.

[21] Zhiyong Xu, Yingwu Zhu, Rui Min, and Yiming Hu. Achieving Better Load Balance in Distributed Storage System. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, June 2002.

[22] Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. COCA: A Secure Distributed Online Certification Authority. *ACM Trans. Computer Systems*, 20(4):329–368, 2002.

A Acknowledgments

We thank Lea Kissner, Emil Ong, Naveen Sastry, Umesh Shankar, and Hoeteck Wee for providing helpful feedback on an earlier draft of this work.

B Proof of Robustness

Proof. We need to show completeness, soundness, and zero knowledge simulatability of the signature share verification protocol when verifying $\textsf{col-2}_i$ from player $P_i \in \mathcal{P}'$.

- **Completeness**: An honest player $P_i \in \mathcal{P}'$ should convince any verifier that the protocol was followed with high probability. In fact, if the signature share verification protocol is correctly followed, then the verifier will accept with probability 1.
- **Soundness**: No corrupted player $P_i \in \mathcal{P}'$ should be able to fool any verifier into accepting incorrect shares with high probability. Using the definitions for e, d, and f from Section 3.5, we require that both

$$\begin{array}{ll} g^v d^{-c} \equiv g^r & \pmod p \\ e^v f^{-c} \equiv h^r & \pmod p \end{array}.$$

  Therefore, $g^v d^{-c} \equiv g^{v - y_i c} \equiv g^r \pmod p$ if and only if $v \equiv y_i c + r \pmod{p'}$. In addition, $e^v f^{-c} \equiv g^{m_i v} g^{(\textsf{col-2}_i - z_i)(-c)} \equiv (g^{m_i})^r \pmod p$, which implies that $m_i v - c(\textsf{col-2}_i - z_i) \equiv m_i r \pmod{p'}$. By using $e^v f^{-c} \equiv h^r \pmod p$ from above, we see that $m_i y_i c \equiv c(\textsf{col-2}_i - z_i) \pmod{p'}$. If $c \not\equiv 0 \pmod{p'}$, then clearly $\textsf{col-2}_i$ is the correct share. If $c \equiv 0 \pmod{p'}$, then $\textsf{col-2}_i$ may be incorrect. By the Discrete Logarithm Assumption, no probabilistic polynomial time adversary can produce such a v with non-negligible probability.
- **Zero Knowledge Simulatability**: No cheating verifier should learn anything useful after running the protocol. We can easily construct a simulator S which simulates the view of the verifier when verifying P_i's $\textsf{col-2}_i$. To do so, S selects c and v uniformly at random and fixes $H(g, d, e, f, g^v d^{-c}, e^v f^{-c})$ to be c, since we are working in the Random Oracle model. Thus, S has recreated the view of the verifier without knowing P_i's secret key share y_i, so the signature share verification protocol has zero knowledge.

As a result, our on-line/off-line threshold signature scheme is robust. We sketch an alternative approach without random oracles in Appendix D □

C Proof of Existential Unforgeability

The proof of existential unforgeability will be in a similar style to the proof in Shamir and Tauman [19]. First we make use of the following Lemma to show that our One-More-R assumption is implied by a standard assumption:

Lemma 1. *Suppose that there exists an adversary $\mathcal{B}$ that breaks the One-More-R assumption for the discrete logarithm chameleon hash with advantage ε. We show how to construct an algorithm A that breaks the One-More-Discrete-Log assumption with advantage $\geq \varepsilon$.*

Proof. We let $\mathcal{A}$ respond to $\mathcal{B}$'s queries in the One-More-R problem. $\mathcal{A}$ is given as input g and $\{z_1, z_2, \ldots, z_{n(k)+1}\}$. Let $\mathcal{A}$ be described as follows:

1. Pick y uniformly in $\mathbb{Z}_{p'}$
2. Let $h = g^y$, and initialize $\mathcal{B}$ with g and h.
3. For $1 \leq i \leq n(k)+1$, pick m_i uniformly in $\mathbb{Z}_{p'}$ and let $v_i = z_i h^{m_i}$.
4. Send $\mathcal{B}$ the set $\{v_1, v_2, \ldots, v_{n(k)+1}\}$.
5. Whenever $\mathcal{B}$ makes a $\mathsf{Get\text{-}An\text{-}R}(v, m)$ query, receive $t = \mathsf{Dlog}(g, v)$. Return the value $t - ym$ to $\mathcal{B}$.
6. If $\mathcal{B}$ successfully outputs $\{(m'_1, r'_1), (m'_2, r'_2), \ldots, (m'_{n(k)+1}, r'_{n(k)+1})\}$ such that $CH_{HK}(m'_i, r'_i) = v_i$ for all i, $\mathcal{A}$ returns $\{x_1, x_2, \ldots, x_{n(k)+1}\}$ where $x_i = r'_i + y(m'_i - m_i)$. Otherwise, abort.

Clearly, we have $\mathsf{Adv}\ \mathcal{B} \leq \mathsf{Adv}\ \mathcal{A}$. □

Using the One-More-R assumption, we can prove that our on-line/off-line threshold signature scheme is secure in the adaptive chosen message attack model.

Theorem 3. *Let $\mathcal{TS}$ = (Thresh-Key-Gen, Thresh-Sig, Ver) be a given simulatable threshold signature scheme. Then we let $\mathcal{TS}^{On/Off}$ = (On/Off-Thresh-Key-Gen, Thresh-Sig-Off-line, Thresh-Sig-On-line, Ver) be the resulting On-line/Off-line Threshold Signature scheme. If $\mathcal{TS}^{On/Off}$ is existentially forgeable by an q-adaptive chosen message attack with success probability ε, then one of the following must hold:*

1. *There exists a probabilistic algorithm that breaks the One-More-R assumption with probability at least $\varepsilon/2$.*
2. *The underlying threshold signature scheme $\mathcal{TS}$ is existentially forgeable by a n-chosen message attack with probability $\varepsilon/2$.*

Proof. Suppose that an adversary $\mathcal{A}$ forges a signature in the $\mathcal{TS}^{On/Off}$ scheme with a q-chosen message attack with probability ε. Now let $\{m_1, m_2, \ldots, m_q\}$ be the q messages chosen by $\mathcal{A}$ to be signed by the $\mathcal{TS}^{On/Off}$ scheme. Let $\{(\sigma_1, m_1, r_1), \ldots, (\sigma_q, m_q, r_q)\}$ be the signatures produced in this fashion by the $\mathcal{TS}^{On/Off}$ scheme. Then our assumption is that $\mathcal{A}$ outputs a signature forgery

(σ, m, r), where $V_{PK}(CH_{HK}(m,r),\sigma) = \mathsf{Accept}$ and $m \neq m_i$ for all i, with probability ε. Then either there exists an i such that $CH_{HK}(m_i, r_i) = CH_{HK}(m,r)$ or there does not exist such an i. One of these cases occurs with probability at least $\varepsilon/2$.

If the first case holds with probability at least $\varepsilon/2$, then we define a simulator $\mathcal{S}$ that breaks the One-More-R assumption. $\mathcal{S}$ is given as input the public bases g and h, as well as the set of challenges $\{v_1, v_2, \ldots, v_{n(k)+1}\}$.

$\mathcal{S}$ then simulates the On/Off-Thresh-Key-Gen phase with $\mathcal{A}$. When the simulation gets to the point where h is to be generated by using the DKG protocol, $\mathcal{S}$ uses $\mathsf{SIM}^{DKG}(h)$, the DKG simulator, to "fix" the result of the DKG run to be h.

On the i^{th} run of the Thresh-Sig-Off-line phase, $\mathcal{S}$ simulates the phase as normal. However, when it reaches the point where h^m is to be generated using the DKG protocol, it uses $\mathsf{SIM}^{DKG}(v_i/g^r)$ to fix the value of h^m so that the resulting chameleon hash $g^r h^m$ equals the given v_i value. $\mathcal{S}$ then simulates the rest of the phase as normal.

On the j^{th} run of the Thresh-Sig-On-line phase, with input m'_j specified by $\mathcal{A}$, $\mathcal{S}$ simulates the phase as normal. Suppose that the players involved are $\mathcal{P}' \subset \mathcal{P}$. Of the players in $\mathcal{P}'$, without loss of generality let $\{P_1, P_2, \ldots, P_t\} \subset \mathcal{P}'$ be the players controlled by the adversary $\mathcal{A}$. Since $\mathcal{S}$ "controls" more than t players, it is able to reconstruct the values of r_i, y_i, m_i, and z_i for $1 \leq i \leq t$ from its own shares, since all were generated by the DKG protocol. Hence $\mathcal{S}$ is able to recover $\mathsf{col\text{-}1}_i$ and $\mathsf{col\text{-}2}_i$ for all $1 \leq i \leq t$. Now $\mathcal{S}$ fixes P_n. For each P_i, $t < i < n$, $\mathcal{S}$ picks $\mathsf{col\text{-}1}_i$ and $\mathsf{col\text{-}2}_i$ uniformly at random and broadcasts them. In addition, $\mathcal{S}$ queries the Get-An-R oracle on m_j and v_j to receive r'_j. With this information $\mathcal{S}$ can simply fix the value of $(\mathsf{col\text{-}1}_n, \mathsf{col\text{-}2}_n)$ such that the interpolation of all the $\mathsf{col\text{-}1}_i + \mathsf{col\text{-}2}_i$ values comes out to be r'_j.

At the end, $\mathcal{A}$ produces (σ, m, r) such that $V_{PK}(CH_{HK}(m,r),\sigma) = \mathsf{Accept}$ and there exists an i such that $CH_{HK}(m,r) = v_i$ and v_i was not used by $\mathcal{S}$ in a run of Thresh-Sig-On-line. But if this is the case, $\mathcal{S}$ has produced One-More-R value, namely r.

If the second case holds with probability at least $\varepsilon/2$, then we define a simulator $\mathcal{S}$ that existentially forges a signature on the underlying threshold signature $\mathcal{TS}$. In addition, we let $\mathsf{SIM}_1^{\mathcal{TS}}$ and $\mathsf{SIM}_2^{\mathcal{TS}}$ be defined as in Definition 7.

$\mathcal{S}$ simulates the On/Off-Thresh-Key-Gen phase as normal, except during Thresh-Key-Gen. In this case, $\mathcal{S}$ uses $\mathsf{SIM}_1^{\mathcal{TS}}$ to fix the public key for $\mathcal{TS}$ to be the public key for the signing oracle $\mathsf{Sig}_{\mathcal{TS}}$

On the i^{th} run of the off-line phase, let $\mathcal{S}$ simulate it as normal up until the point where it needs to perform Thresh-Sig on $v_i = CH_{HK}(m_i, r_i)$. When this occurs, $\mathcal{S}$ queries the signing oracle $\mathsf{Sig}_{\mathcal{TS}}$ to get σ_i, the signature of v_i. $\mathcal{S}$ then uses $\mathsf{SIM}_2^{\mathcal{TS}}$ to simulate a run of Thresh-Sig with $\mathcal{S}$ on input v_i, such that the output is fixed to σ_i. We can do this because our assumption is that Thresh-Sig is simulatable.

Each run of the on-line phase is simulated as normal by $\mathcal{S}$.

At the end, $\mathcal{A}$ produces (σ, m, r) such that $V_{PK}(CH_{HK}(m,r),\sigma) = \mathsf{Accept}$ and for all i, $v_i \neq CH_{HK}(m,r)$. But in this case, $\mathcal{S}$ has forged a signature σ on a message $CH_{HK}(m,r)$ not queried to the signing oracle $\mathsf{Sig}_{\mathcal{TS}}$. $\square$

D Extensions

D.1 Using Merkle Trees for Batching

We explained earlier that computing a threshold signature when performing writes for small files in Pond [15] is expensive, while for large files, the time spent computing the threshold signature is negligible compared to the actual write. In the event that a threshold signature must be quickly computed on demand, our scheme immediately becomes attractive over other known signature schemes. This is especially true for Pond when computing threshold signatures for small writes.

One way of improving performance is to batch messages using Merkle hash trees [14]. Instead of signing messages one by one, we wait for n messages to arrive and then build a Merkle tree over these messages. After computing the hash h_r on the root node, we sign h_r using our on-line/off-line threshold signature scheme. To verify the validity of any message m that was included in the batch, we hash m and using hashes from sibling nodes as well as other internal nodes, we can compute h_r. We then verify that h_r is in fact the hash that was signed. If there are a total of n messages and the batch size is B, then a total of $\lceil \frac{n}{B} \rceil$ signature stamps are needed. This approach does trade latency for throughput, and it depends on how much time can be spent waiting for messages to arrive on-line. In fact, Merkle trees for batching has been applied to Shoup's scheme in OceanStore in order to increase throughput for small updates [16].

D.2 Eliminating Random Oracles

Our signature share verification algorithm requires the use of a random oracle H because it allows the verification to be non-interactive. Moreover, the random oracle can actually be eliminated, but at the cost of interaction. This is done by using the techniques in [5], which eliminates the random oracle from the verification step in Shoup's RSA threshold scheme, and can be easily adapted to our scheme.

E Further Work

– *Other On-line/Off-line threshold signature schemes.* Our signature scheme uses a Chameleon Hash function because of its elegant property for allowing interpolation in the exponents. An interesting extension would be to consider elliptic curves over a finite field $\mathbb{F}_p$. Moreover, do there exist other kinds of families of Chameleon Hash functions that can successfully be used for constructing an on-line/off-line threshold signature scheme?

- *Optimality.* In the on-line phase, our optimistic threshold signature scheme is an order of magnitude faster than Shoup's RSA threshold signature scheme. Can our signature scheme be further optimized or is it optimal? Furthermore, both schemes require modular exponentiations for ensuring robustness. Can this be avoided?
- *Proactive threshold signatures.* A useful extension of threshold signatures is making them proactive [10]. This requires that the secret key shares held by each player are periodically refreshed such that the old shares can no longer be used and the public and private keys remain unchanged. Making our on-line/off-line threshold signature scheme be proactive can be useful because OceanStore's design [15] calls for the use of proactive threshold signatures.
- *Bursty traffic analysis.* Our threshold signature schemes becomes useful only if enough signature stamps can be created off-line. In order to better understand when on-line/off-line threshold signatures are beneficial, can we create a suitable traffic model?