Simplified Threshold RSA with Adaptive and Proactive Security

Jesús F. Almansa, Ivan Damgård*, and Jesper Buus Nielsen**

BRICS***, Department of Computer Science
University of Aarhus, Denmark
{jfa,ivan,buus}@brics.dk

Abstract. We present the currently simplest, most efficient, optimally resilient, adaptively secure, and proactive threshold RSA scheme. A main technical contribution is a new rewinding strategy for analysing threshold signature schemes. This new rewinding strategy allows to prove adaptive security of a proactive threshold signature scheme which was previously assumed to be only statically secure. As a separate contribution we prove that our protocol is secure in the UC framework.

Keywords: Proactiveness, Adaptive Security, Threshold RSA, Universal Composability.

1 Introduction

The concept of threshold cryptography was first introduced by Desmedt [Des87]. In threshold cryptography n servers run a service in such a way that even if some t servers are corrupted, the service is still available and secure. In a *threshold signature* the servers implement a service for signing messages under a signature key shared between the servers with some threshold t.

The first RSA based threshold signature was given independently by Boyd [Boy89] and Frankel [Fra89]. In both protocols the signing key d is shared additively among the servers. The first RSA threshold scheme was published by Santis et al. [SDFY94], and although the key sharing is polynomial, it does not tolerate actively cheating servers. This restriction was later removed independently by Frankel et al. [FGY96] and Gennaro et al. [GJK96]. All of these protocols are only proved secure against static adversaries, i.e., the set of corrupt parties is fixed before the protocol starts.

In [OY91], Ostrovsky and Yung introduced the notion of *proactive* security, in which the life span of a protocol is divided into separate time periods and we assume that the adversary can corrupt at most t players in each period. However, the set of corrupted players may change from one period to the next, so the protocol must remain secure, even though every player may have been

* Supported by FICS, Foundations in Cryptography and Security, financed by the Danish Research Council

** Supported by FICS and ECRYPT, European Network of Excellence for Cryptology.

*** Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation

corrupt at some point. This can, for instance, be achieved for a protocol based on secret sharing by having players re-randomize the shares they hold between periods, and erase the old shares. This is called *refreshment*.

In [FGMY97b] Frankel et al. published the first *proactive* threshold RSA signature, as a generalization of an unpublished protocol by Jakobsson et al. [JJKY95]. This protocol, does not scale up well: even for a moderately large number of servers it is either highly inefficient or does not tolerate the optimal threshold $t < n/2$, and it is only *statically* secure. Next, in [FGMY97a] Frankel et al. achieved optimal threshold. Later, Rabin [Rab98] gave a simplified static proactive protocol by combining the best of the linear and polynomial sharing techniques. The protocol from [Rab98] was later optimized and simplified by Jarecki and Saxena [JS05], still obtaining a static proactive protocol.

In [CGJ$^+$99] Canetti et al. add mechanisms to the protocol from [Rab98], to obtain an *adaptively* secure protocol, and even though adaptive security is only claimed for the non-proactive version of [Rab98], the protocol in [CGJ$^+$99] seems to be the first adaptive proactive threshold RSA signature.

Later Frankel et al. [FMY01] gave an adaptively secure version of the protocol from [FGMY97b]. The protocols from [CGJ$^+$99,FMY01] seem to be the only published adaptive proactive threshold RSA signatures. Unfortunately both protocols have some practical drawbacks. The linear-sharing based protocol in [FMY01] inherits the problem from [FGMY97b] that it is inefficient or non-optimally secure unless the number of servers is small. The protocol [CGJ$^+$99] modifies the protocol from [Rab98] such that before each signature generation the signing-key shares are refreshed. This adds a considerable performance overhead.

Our contributions

The paper has three main contributions.

1. The first contribution is a novel analysis technique which allows to prove that the protocol from [Rab98] is – with minor modifications – *adaptively* secure, contrary to what was previously believed. Indeed, as mentioned above, the authors of [CGJ$^+$99] add an expensive mechanism to the protocol to make it adaptively secure. Our first contribution shows that this mechanism is unnecessary. The technical problem we need to solve to do this is explained below.
2. Our second contribution is a technique for avoiding so-called key-share exposure. In the protocol from [Rab98], if a server fails to contribute correctly to the signature generation, the key share of that server is reconstructed and thus exposed in public. Such key-share exposure can degrade security in practice. For instance, if a server fails to contribute only because it is temporarily down, this error will be made worse if the remaining servers expose its key share. Our second contribution shows that key-share exposure is not necessary to make the protocol from [Rab98] actively and adaptively secure.
3. Our final contribution consists of two definitions for security of threshold signature schemes. One is cast in the universally composable (UC) framework, the other one is a more standard definition similar to the one from [CGJ$^+$99].

We show that the two are equivalent. This allows simplified proofs that our protocols – and other protocols as well – are secure in the UC framework.

The technical problem we solve with our first contribution is the following: In [CGJ+99] a useful technique known as the *Single Inconsistent Player* technique was introduced. It facilitates proving adaptive security of threshold signature schemes using simulation arguments as follows: we typically want to show that a successful adversary against the protocol can be used to break security of an underlying single server signature scheme. To this end, we build a simulator which does a chosen message attack on the underlying signature scheme while simulating the adversary's view of the protocol with the aim of having him forge a signature. The simulator initially chooses among the currently honest players a single inconsistent player (SIP) and arranges matters such that it knows valid-looking secret keys of *all players except the SIP*. It can therefore simulate successfully even against adaptive corruption, *as long as the SIP is not corrupted*. If there is a non-negligible chance that the SIP stays honest, there is also a non-negligible chance that the adversary produces a forged signature.

It is natural to try using the SIP technique for doing also *proactive* adaptive security. However, in this scenario we must choose a new SIP every time keys are refreshed, since under a proactive attack no single party can hope to stay honest throughout the protocol. But even this will not work: the probability that a single SIP remains honest in a single phase cannot be made arbitrarily close to 1, whence the probability that we are lucky in every phase is negligible. Thus, already with as few as a super-logarithmic number of proactive phases, a simple straight-line simulation will not work. A potential solution is to use rewinding, i.e., every time the SIP is corrupted, rewind back to the last refreshment, choose a new SIP and try again. This turns out to work, if one is willing to refresh keys *every time a message is about to be signed* (as shown in [CGJ+99]).

If we are not willing to pay the price this costs in efficiency, a further technical problem emerges: if we rewind past a point where a message m was given as input, we risk that when we go forward again, the adversary asks us to generate signatures on different messages than before, in particular different from m. The net result of this is that we may end up with a simulated transcript where the adversary apparently breaks the scheme by producing a valid signature on message m, which he did not ask to have signed. But in reality, the simulator had to ask for the signature on m somewhere in the rewinding process, so we did not break the underlying signature scheme after all. In connection with the proof of Theorem 2 we give an example adversary to show that this really is a problem and we show how to solve it. The basic idea is to guess the point in the simulation where the simulator asks for a signature on the "fatal" message, and simply refrain from asking. One then has to show that this does not bias the distribution of the simulation. More details are given later.

How to read this paper

In Section 2.3 we give a sketch of the UC definition of security, and in Section 3 a formal specification of secure threshold signatures as an ideal functionality. Readers who are more interested in the protocol constructions can skip this without loss of continuity, as all protocols are proved according to the more standard definition from Section 3.1, which is equivalent to UC security. However, to read the protocol descriptions and proofs, the notation introduced in Sections 2.2 and 2.1 is necessary.

2 Proactive UC security

In this section we describe our computational and adversarial model. Our work utilizes the Universal-Composability framework by Canetti, introduced in [Can01], and last revised in [Can]. Among the upgrades in the last revision that are relevant to us, it is now possible to model *erasures*, by allowing a party to leak only partial internal state upon a corruption. Since the composition theorem remains valid, we can cast proactiveness in the framework and make proofs of security, while still being able to use the composition theorem. Likewise, it is also shown that w.l.o.g., one may assume that a single entity (the environment) models all activity external to the protocol, including adversarial activity. We use this technical simplification.

Finally, it is possible to specialize [Can] to the case of synchronous networks, which we will do here. Thus we do not need to define a new synchronous model and reprove the composition theorem as was done in [DN03] and [Nie04].

We now give a brief description of our instance of the UC framework. For a more detailed description of the proactive UC framework, see [Alm05].

2.1 Computations

All entities are PPT Interactive Turing machines (ITM). An n-party protocol π in the $\mathcal{G}$-hybrid model is a set of n ITMs, whose identities $P_1, \ldots, P_n$ are all different, and an ideal functionality $\mathcal{G}$ to which parties are granted use.

In general, $\mathcal{G} = (\mathcal{G}_1, \ldots, \mathcal{G}_m)$, $1 < m$, may include one or more ideal functionalities, and we will assume $\mathcal{G}_1 = \mathcal{F}_{\texttt{aut}}$ provides authenticated transmission, to be used for communication between parties. In some cases, we will use instead a functionality $\mathcal{F}_{\texttt{SMT}}$ modeling secure point-to-point channels. These abstractions allow us to focus on the high-level properties of our protocols, yet they can be implemented using well known techniques. Note, however, that for proactive security, care should be taken with refreshing the key material used for message transmission.

The protocol runs while interacting with an *environment*, an ITM $\mathcal{Z}$ that models external (adversarial) activity, and which provides inputs to honest parties and receives their corresponding outputs. We use $\texttt{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}$ to denote the entire process of running π while interacting with $\mathcal{Z}$ and $\mathcal{G}$.

The execution proceeds in communication rounds, that we denote by $r = 0, 1, \ldots$.

A *proactive protocol* proceeds in *phases*. A phase consists of a number of consecutive rounds, and every round belongs to exactly one phase. There are two kinds of phases, *refreshment* and *operational*, which occur alternately. Finally, a *stage* consists of an *opening* refreshment phase, an operational phase in the middle and a *closing* refreshment phase. Thus, each refreshment is the closing of one stage and the opening of another. We use $u = 0, 1, \ldots$ to denote stages.

The intuition is that during the operational phases, the protocol provides whatever service it was designed for, whereas refreshment phases are used to rerandomize various representations of data so that attacks in different phases will not be able to benefit from each other.

We allow $\mathcal{Z}$ to decide when refreshment starts (equivalently, when a new stage begins), by sending a command to each party. Refreshment ends when all honest parties have output a special symbol indicating end of refreshment.

2.2 Adversaries

As mentioned, $\mathcal{Z}$ also models the *adversary*. As such, $\mathcal{Z}$ may corrupt parties adaptively throughout the protocol, subject to the limitation that no more than t parties can be corrupt *in every stage*. In particular, this means that if a party is corrupt during a refreshment phase, he is considered to be corrupt in both of the two stages to which the phase belongs. After corruption, $\mathcal{Z}$ acts on behalf of the corrupted player. Corruption may be passive, where $\mathcal{Z}$ internally executes the correct protocol on behalf of the corrupted player, or active where $\mathcal{Z}$ decides on its own the actions of the corrupted player.

If player P_i is corrupted during an operational phase, $\mathcal{Z}$ is given the view of P_i starting from his state at the beginning of the current operational phase. This models the assumption that all randomness and data used in the previous refreshment phase is erased, except for the information that the protocol specifies should be used afterwards.

If the corruption is made during a refreshment phase, say, the closing refreshment of stage u, $\mathcal{Z}$ receives the view of P_i starting from his state at the beginning of the operational phase of stage u, and P_i is assumed to be corrupt for stage $u + 1$.

If P_i is corrupt when a refreshment begins, $\mathcal{Z}$ may decide to *leave* him, which may allow $\mathcal{Z}$ to corrupt new parties, subject to the bound of t corruptions per stage. In this case, we say P_i is *decorrupted*.

A decorrupted player immediately starts taking part in the protocol as any honest player. In the passive corruption case, he starts from the correct state specified by the protocol at this point. In the active corruption case, he starts from a *default state after round* r. This state is application-dependent in general.

2.3 UC Security

Security is defined by comparing protocol π's execution with an *ideal protocol execution*. There, instead of parties, an ideal functionality $\mathcal{F}$ is used to *specify* the desired input/output behavior of π. It also specifies the information allowed to be leaked from π to the environment.

Security loosely speaking means that whatever $\mathcal{Z}$ could achieve by attacking π, it could also achieve by interacting with $\mathcal{F}$. To make this precise, a special ITM $\mathcal{T}$ is introduced. The goal of $\mathcal{T}$ is to *simulate* the adversary's view of π, based only on the information $\mathcal{F}$ is willing to exchange with the environment.

We declare π secure in the $\mathcal{G}$-hybrid model if no environment can distinguish interactions with π from those with $\mathcal{F}$ and $\mathcal{T}$. More formally:

The environment is assumed to always end by outputting a bit which we think of as its guess at whether it works in the ideal or the hybrid scenario.

When $\mathcal{Z}$ interacting with π in the $\mathcal{G}$-hybrid model, on security parameter k, auxiliary input z to $\mathcal{Z}$, and the random coins of all machines are uniformly chosen, this output of $\mathcal{Z}$ is a random variable denoted $\mathtt{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}(k,z)$. We denote by $\mathtt{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}()$ the ensemble $\{\mathtt{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$.

Similarly, $\mathtt{IDEAL}_{\mathcal{F},\mathcal{T},\mathcal{Z}}(k,z)$ and $\mathtt{IDEAL}_{\mathcal{F},\mathcal{T},\mathcal{Z}}()$ are the random variable and ensemble produced when $\mathcal{Z}$ interacts with $\mathcal{F}$ and $\mathcal{T}$ in the ideal process.

Using $\stackrel{c}{\approx}$ to denote computational indistinguishability, we then have:

Definition 1 (UC Security). *A protocol π proactively t-realizes a functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, if there exists a simulator $\mathcal{T}$ such that for all environments $\mathcal{Z}$ corrupting at most t parties per stage it holds that* $\mathtt{IDEAL}_{\mathcal{F},\mathcal{T},\mathcal{Z}}() \stackrel{c}{\approx} \mathtt{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}()$.

3 Defining Proactive Threshold Signatures

We define threshold signatures by giving a functionality, $\mathcal{F}_{\mathtt{ThSig}}$, that is a version of Canetti's signature functionality[Can04], adapted for the threshold case.

FUNCTIONALITY $\mathcal{F}_{\mathtt{ThSig}}$

Key Generation, initiate Having received the same message $(\mathit{KeyGen}, \mathit{sid})$ from all honest parties in a set $S = \{S_1, \ldots, S_n\}$ in the same round, and $\mathit{sid} = (S, \mathit{sid}')$ for some sid', send $(\mathit{KeyGen}, \mathit{sid})$ to $\mathcal{Z}$.

Key Generation, finalize Upon receiving $(\mathit{KeyGen}, \mathit{sid}, v)$ from $\mathcal{Z}$, if $(\mathit{KeyGen}, \mathit{sid})$ was sent earlier, record v and send (sid, v) to all $S_i \in S$. All further commands that do not contain the *sid* established here are ignored.

Signature Generation, initiate Having received $(\mathit{Sign}, \mathit{sid}, m)$ from all honest $S_i \in S$ in the same round, store $(\mathit{Sign}, \mathit{sid}, m)$ and send it to $\mathcal{Z}$. There might be several identical $(\mathit{Sign}, \mathit{sid}, m)$ stored.

Signature Generation, finalize Upon receiving $(\mathit{Signature}, \mathit{sid}, m, \sigma)$ from $\mathcal{Z}$, if $(\mathit{Sign}, \mathit{sid}, m)$ is stored and an entry of the form $(m, \sigma, 0)$ was not recorded, delete an entry $(\mathit{Sign}, \mathit{sid}, m)$, record the entry $(m, \sigma, 1)$ and send $(\mathit{Signature}, \mathit{sid}, m, \sigma)$ to all $S_i \in S$.

Signature Verification Upon receiving a message $(\mathit{Verify}, \mathit{sid}, m, \sigma, v')$ from some party P, give $(\mathit{Verify}, \mathit{sid}, m, \sigma, v')$ to $\mathcal{Z}$. Upon receiving $(\mathit{Verified}, \mathit{sid}, m, \sigma, \phi)$ from $\mathcal{Z}$, send $(\mathit{Verified}, \mathit{sid}, m, \sigma, f)$ to P, where f is determined as follows:

1. If $v' = v$ and the entry $(m, \sigma, 1)$ is recorded, then set $f = 1$ (guarantees that if v' is the registered public key and σ is legitimately generated, then verification succeeds).
2. Else, if $v' = v$ and no entry $(m, \sigma, 1)$ is recorded, set $f = 0$ (guarantees that if v' is the registered public key and m was not legitimately signed, then verification fails). Record the entry $(m, \sigma, 0)$.
3. Else, if $v \neq v'$, set $f = \phi$.

Refreshment On input signaling that a refreshment phase starts in this round, record this and signal end of refreshment in the next round (this reflects that our protocol implementing the functionality takes one round to do the refreshment).

Note that all our functionalities receive initially a session id $sid = (S, sid')$ where S is the set of players who participate in realizing the functionality and sid' is a number identifying this particular instance of the functionality.

The functionality defines a player set S called the *servers* and a *verification key* v. Only the servers can ask $\mathcal{F}_{\texttt{ThSig}}$ to sign messages, but any player with the correct key v can use $\mathcal{F}_{\texttt{ThSig}}$ to verify a signature. For simplicity we also assume some external mechanism for the servers to agree on which message to sign and in which round. We model this by assuming throughout that all our environments behave such that if an honest player gets a message to sign as input, all honest players get the same message as input in the same round.

We note that the logic in Canetti's signature functionality is slightly more complicated than ours because it has to deal with the case where the signer is corrupted. In our case the single signer is replaced by the set of servers, and hence we can demand by bounding the number of corrupted servers that things will always work as if "the signer" is honest.

For a protocol π (in the $\mathcal{G}$-hybrid model) we can then say it is a *secure UC threshold signature scheme for the class* $\mathscr{Z}$ if π realizes $\mathcal{F}_{\texttt{ThSig}}$ when quantifying over $\mathcal{Z} \in \mathscr{Z}$ in the definition of security.

3.1 Equivalence to a more standard notion

In [Can04] it was proved that for a (non-threshold) signature scheme, implementing the signature functionality in [Can04] is equivalent to the scheme being correct (i.e. signed messages are accepted by the verification algorithm), consistent (two verifications of the same message and signature give the same result) and unforgeable under chosen message attack.

Threshold Signature Scheme $\mathcal{F}_{\texttt{ThSig}}$

Key Generation (well-formed) If all honest $S_i \in S$ receive the same message $(KeyGen, (S, sid'))$ in the same round, then after some rounds all honest parties $S_i \in S$ output one common message (sid, v).

Signature Generation (well-formed) If all honest $S_i \in S$ receive the same message $(Sign, sid, m)$ in the same round, then after some rounds all honest $S_i \in S$ output one common message $(Signature, sid, m, \sigma)$.

Signature Verification (well-formed) If an honest party P_i receives input $(Verify, sid, m, \sigma, v')$ in round r, then P_i outputs one corresponding message of the form $(Verified, sid, m, v', f)$ in round r.

No other messages (well-formed) No honest party outputs a message not described above.

Signature Verification (correct) If an honest party P_i receives input $(Verify, sid, m, \sigma, v)$ in round r and some honest party once output $(Signature, sid, m, \sigma)$, then P_i outputs $(Verified, sid, m, v, 1)$ in round r.

Signature Verification (consistent) If two honest parties P_i and P_j (not necessarily distinct) outputs $(Verified, sid, m, \sigma, v, f_i)$ respectively $(Verified, sid, m, \sigma, v, f_j)$, then $f_i = f_j$.

Signature Verification (unforgeable) If an honest party P_i outputs $(Verified, sid, m, v, 1)$ in round r, then in some round $r' \leq r$ an honest party received the input $(Sign, sid, m)$.

In this section we do a similar "sanity check" of our definition of a UC threshold signature scheme, by giving a property based definition of what it means for a protocol to be a secure threshold signature scheme and then proving that this notion is equivalent to the UC notion.

Let $\mathscr{Z}$ be a set of environments. We say that π has one of the properties in the figure above (relative to $\mathscr{Z}$) if for all $\mathcal{Z} \in \mathscr{Z}$, the probability that the property fails when executing $\mathtt{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}$ is negligible.

Theorem 1. *If π is well-formed, correct, consistent and unforgeable relative to $\mathscr{Z}$, then π is a secure UC threshold signature scheme for $\mathscr{Z}$.*

Briefly, this result is shown by constructing a UC simulator $\mathcal{T}$, which will generate on its own a set of keys for the signature scheme by executing internally an instance of π. Then, using the private key(s), it can trivially simulate $\mathcal{Z}$'s view of π by simply following the protocol to generate signatures. One then observes that the only way this could differ from actual executions is if $\mathcal{Z}$ can produce a valid signature that was not legally generated. Such a signature would be accepted in the hybrid process, but rejected in the ideal one. However, the unforgeability of π ensures that such events occur with negligible probability. The (tedious but straightforward) details can be found in [Alm05].

4 Passive Security

In this section we give a UC threshold signature scheme under the class $\mathscr{Z}$ of passive, adaptive environments which corrupts at most $t = n-1$ players in each stage. In Section 5 we describe how to obtain active security. To simplify matters, we will assume a trusted dealer who distributes keys to the servers initially. The dealer is modeled as an ideal functionality $\mathcal{F}_{\mathtt{KeyGen}}$, in other words, we operate in the $\mathcal{F}_{\mathtt{KeyGen}}$-hybrid model.

FUNCTIONALITY $\mathcal{F}_{\mathtt{KeyGen}}$

Key Generation, initiate Having received the same message $(KeyGen, sid)$ from all honest parties in the same round, parse sid as (S, sid'), perform RSA key generation with security parameter k to obtain modulus N and exponents e, d. Next, for $i = 1, \ldots, n$, where n is the size of the set S, choose at random d_i in $[-nN^2..nN^2]$ and set $d_{public} = d - \sum_i d_i$. Then send $(KeyGen, sid, v, d_{public})$ to $\mathcal{Z}$, where v is the RSA public key (N, e).

Key Generation, finalize (To avoid having to specify how many rounds key generation will take, we let the environment decide when to return results) Upon receiving $(KeyGenFinish, sid)$ from $\mathcal{Z}$, send $(KeyGen, sid, v, d_{public}, d_i)$ to each $S_i \in S$.

The keys generated will be used in an RSA signature scheme, where we assume (as usual) that binary strings of length up to some polynomial in the security parameter k can be signed, where the signature on m is of form $H(m)^d \bmod N$, and where H is some preprocessing that typically involves a hash function. The details of this are left out of scope here.

We also assume secure point-to-point channels, i.e., we assume a functionality $\mathcal{F}_{\mathtt{SMT}}$ for secure message transmission. This functionality will accept inputs containing message and sender/receiver id, and will in the next round deliver the message to the intended receiver, revealing only the message length to the adversary. Whenever we speak about sending a message privately in the following, this refers to calling $\mathcal{F}_{\mathtt{SMT}}$.

PROTOCOL π

All parties in the protocol run the following code:

Key Generation, initiate On input $(KeyGen, sid)$, parse sid as (S, sid') and send $(KeyGen, sid)$ to $\mathcal{F}_{\mathtt{KeyGen}}$.

Key Generation, finalize Wait to receive $(KeyGen, sid, v, d_i, d_{public})$ from $\mathcal{F}_{\mathtt{KeyGen}}$, store this information, and output (sid, v). Here v is the RSA public key (N, e).

Signature Generation, initiate On input $(Sign, sid, m)$, send $(sid, m, H(m)^{d_i} \bmod N)$ to all $S_j \in S$.

Signature Generation, finalize Upon receiving $(sid, m, H(m)^{d_j} \bmod N)$ from all $S_j \in S$ (i.e. for $j = 1, \ldots, n$), compute the signature

$$\sigma = H(m)^{d_1} H(m)^{d_2} \cdots H(m)^{d_n} H(m)^{d_{public}} \bmod N,$$

and output $(Signature, sid, m, \sigma)$.

Signature Verification On input $(Verify, sid, m, \sigma, v')$, where v' is an RSA key (N', e'), define $f \in \{0, 1\}$ by $f = 1$ iff $\sigma^{e'} \bmod N' = H(m)$, and output $(Verified, sid, m, \sigma, v', f)$.

Refreshment For each decorrupted P_i, his default state after round r is d_i, i.e., his actual share. Each $S_i \in S$ reshares d_i, i.e., chooses $d_{i,j}$ at random in $[-N^2..N^2]$, sets $d_{i,public} = d_i - \sum_j d_{i,j}$, sends $d_{i,public}$ to all $S_j \in S$ and $d_{i,j}$ privately to S_j. In the next round, each $S_i \in S$ computes $d_i^{new} = \sum_j d_{j,i}$. Finally, all $S_i \in S$ compute $d_{public}^{new} = d_{public} + \sum_i d_{i,public}$. Everyone signals end of refreshment. Only d_{public}^{new} and the private d_i^{new} are remembered in the next phase.

By Theorem 1 it is sufficient to show the following:

Theorem 2. *If the underlying signature scheme is unforgeable under chosen message attack, then the protocol π is well-formed, correct, consistent and unforgeable relative to the class $\mathscr{Z}$ of environments which corrupt, passively and adaptively, at most $n-1$ players in each proactive stage.*

Proof. It is straight-forward to verify that the protocol is well-formed, correct and consistent (in fact these properties hold unconditionally). What remains is to prove that it is unforgeable. So, assume for the sake of contradiction that there exists $\mathcal{Z} \in \mathscr{Z}$ such that with some non-negligible probability $P(\mathcal{Z})$ it happens in $\mathtt{HYB}_{\pi,\mathcal{Z}}^{(\mathcal{F}_{\mathtt{SMT}},\mathcal{F}_{\mathtt{KeyGen}})}$ that an honest party P_i outputs $(Verified, sid, m, \sigma, v, 1)$ in round r without m being signed in some round $r' \leq r$ (in the following we say that m *was signed* in round r' if $(Sign, sid, m)$ was input to all honest parties). We use this to construct a PPT reduction $\mathtt{Red}'(\mathcal{Z})$ which breaks the underlying signature scheme with some non-negligible probability P' related to $P(\mathcal{Z})$. It is given a random RSA verification key (N, e) and is given an oracle $\mathcal{O}(N, d) : m \mapsto H(m)^d \bmod N$. It then tries to compute a *forgery*, i.e. a value (m, σ) where $\sigma^e \bmod N = H(m)$ and where $\mathcal{O}(N, d)$ was not queried on m. The algorithm $\mathtt{Red}(\mathcal{Z})$ described on the next page is used as a sub-routine.

The strategy of $\mathtt{Red}(\mathcal{Z})$ is to run $\mathcal{Z}$ while simulating its view of the protocol. More precisely, $\mathtt{Red}(\mathcal{Z})$ runs $\mathtt{HYB}_{\pi,\mathcal{Z}}^{(\mathcal{F}_{\mathtt{SMT}},\mathcal{F}_{\mathtt{KeyGen}})}$, but it simulates itself the actions of $(\mathcal{F}_{\mathtt{SMT}}, \mathcal{F}_{\mathtt{KeyGen}})$ and the (currently) honest players, using the verification key and oracle it is given, but of course without knowing the secret RSA key. The hope is that $\mathcal{Z}$ will behave (approximately) as in a real attack and will hence produce a forgery that can help us break the signature scheme.

The reduction $\mathtt{Red}(\mathcal{Z})$ uses the single inconsistent player (SIP) technique explained in the introduction. A new SIP is chosen at random after every refreshment phase. We use S_{j_u} to denote the SIP chosen after the opening refreshment of stage u. If the current SIP S_{j_u} is corrupted, $\mathtt{Red}(\mathcal{Z})$ rewinds to the beginning of stage u and tries again.

We now analyze $\mathtt{Red}(\mathcal{Z})$: Let an *attempt* for stage u be a run of $\mathtt{Red}(\mathcal{Z})$ from state State^{u-1} at the beginning of the opening refreshment of u, until S_{j_u} is corrupted or the closing refreshment of u begins. Let a *failed attempt* (*successful attempt*) be an attempt where S_{j_u} is (not) corrupted. Notice that $\mathtt{Red}(\mathcal{Z})$ is trying to create a sequence of successful attempts, closing with $\mathcal{Z}$ terminating or the unforgeability property being violated. Call such a sequence a *successful sequence*. Let d denote the signing key corresponding to the input verification key (N, e), and let $d_1^u, \ldots, d_n^u, d_{public}^u$ be the shares used by $\mathtt{Red}(\mathcal{Z})$ in successful attempt u, and similarly $d_{i,j}^u, d_{i,public}^u$ the values used in the refreshment in successful attempt u. These are called the *real shares* in the following.We first prove:

Claim 1: the view of $\mathcal{Z}$ in a successful sequence is statistically indistinguishable from its view in $\mathtt{HYB}_{\pi,\mathcal{Z}}^{(\mathcal{F}_{\mathtt{SMT}},\mathcal{F}_{\mathtt{KeyGen}})}$.

Note that $\mathtt{Red}(\mathcal{Z})$, when it creates and updates the shares d_i, follows exactly the protocol, except that the secret is zero, instead of the correct d. We now want to argue that if we modify the shares generated by $\mathtt{Red}(\mathcal{Z})$ so they are consistent with d, $\mathcal{Z}$ will still see essentially the same view. To this end, define a new set of shares ${d'}_1^u, \ldots, {d'}_n^u, {d'}_{public}^u, {d'}_{i,j}^u, {d'}_{i,public}^u$ that are equal to the shares generated by $\mathtt{Red}(\mathcal{Z})$, except

$$ {d'}_{j_u}^u = d_{j_u}^u + d, \quad {d'}_{j_u,j_{u+1}} = d_{j_u,j_{u+1}} + d \ . $$

We call these the *virtual* shares. Note that the new set of values is consistent with secret exponent d, but if we restrict to the subset seen by $\mathcal{Z}$ the virtual shares equal the real ones. Moreover, except with negligible probability, the virtual shares are legal, i.e., all shares are in the intervals specified in the protocol. This follows immediately from the fact that the size of the intervals is larger than d by an exponential factor. Note also that when signatures are generated, the contribution from the SIP, σ_{j_u}, as generated by $\mathtt{Red}(\mathcal{Z})$ satisfies $\sigma_{j_u} = H(m)^{{d'}_{j_u}^u}$, since $\sigma = H(m)^d$ and $-d_{public}^u = \sum_{i \in S} d_i^u$. In other words, $\mathtt{Red}(\mathcal{Z})$ already generates signatures consistently with the virtual shares.

Reduction $\mathtt{Red}(\mathcal{Z})$

Run a copy of $\mathtt{HYB}_{\pi,\mathcal{Z}}^{(\mathcal{F}_{\mathtt{SMT}},\mathcal{F}_{\mathtt{KeyGen}})}$ while simulating $(\mathcal{F}_{\mathtt{SMT}},\mathcal{F}_{\mathtt{KeyGen}})$ and the honest parties as follows:

Key Generation, initiate On input $v=(N,e)$ and a set of players S, choose d_i at random in $[0..nN^2]$ for all $S_i \in S$ and set $d_{public} = -\sum_i d_i$. Then, choose a player S_{j_0} at random among the honest players in S and call S_{j_0} the *single inconsistent party* (SIP) for stage 0.

Key Generation, finalize When $\mathcal{Z}$ gives the command to generate keys, send $(KeyGen, sid, v, d_i, d_{public})$ to each S_i on behalf of $\mathcal{F}_{\mathtt{KeyGen}}$. Store the current state State^0 of $\mathtt{HYB}_{\pi,\mathcal{Z}}^{(\mathcal{F}_{\mathtt{SMT}},\mathcal{F}_{\mathtt{KeyGen}})}$.

Refreshment On a signal that opening refreshment of u starts in this round, record state State^{u-1} of $\mathtt{HYB}_{\pi,\mathcal{Z}}^{(\mathcal{F}_{\mathtt{SMT}},\mathcal{F}_{\mathtt{KeyGen}})}$ and set State^u := State^{u-1}. Then execute the refreshment on behalf of the honest players according to the protocol, using as input the current d_i. This results in a new set of d_i's for all the players, and a new d_{public}.[a] Update and record State^u. Finally $\mathtt{Red}(\mathcal{Z})$ picks a new SIP S_{j_u} among the $S_i \in S$ still honest after the refreshment phase.

Signature Generation, initiate When $\mathcal{Z}$ inputs $(Sign, sid, m)$ to all honest $S_i \in S$, call $\mathcal{O}(N,d)$ to obtain $\sigma = H(m)^d \bmod N$.

Signature Generation, finalize In the next round, for each $S_i \in S \setminus \{S_{j_p}\}$, set $\sigma_i = H(m)^{d_i} \bmod N$, and for the SIP S_{j_p}, compute

$$\sigma_{j_p} = \sigma \cdot H(m)^{-d_{public}} \cdot \prod_{S_i \in S\setminus\{S_{j_p}\}} \sigma_i^{-1}.$$

Then for all honest $S_i \in S$, send σ_i to all parties in S.

Corruption When $\mathcal{Z}$ corrupts a server S_i, $\mathtt{Red}(\mathcal{Z})$ sends the d_i it holds for S_i to $\mathcal{Z}$, or both d_i and its older share if corruption is made in opening refreshment of u. If $S_i = S_{j_u}$ (where S_{j_u} denotes the current SIP for stage u), then $\mathtt{Red}(\mathcal{Z})$ gives up this attempt to simulate stage u and restarts the simulation from the recorded state State^u at the beginning of the appropriate phase, using fresh randomness (notice that this involves choosing a new random SIP). To ensure that $\mathtt{Red}(\mathcal{Z})$ runs in PPT it will rerun each operational phase at most kn times and then give up the reduction completely.

Signature Verification $\mathtt{Red}(\mathcal{Z})$ does not need to do anything special here, since verification is just done as in the protocol using v'.

Termination If it ever happens that the unforgeability property is violated by some party P_i outputting $(\mathit{Verified}, sid, m, \sigma, v, 1)$, then $\mathtt{Red}(\mathcal{Z})$ terminates with output (m,σ). If $\mathcal{Z}$ terminates first, then $\mathtt{Red}(\mathcal{Z})$ terminates with an empty output.

[a] Notice that by inspecting the messages that $\mathcal{Z}$ sends privately on behalf of the corrupted parties in $\mathtt{HYB}_{\pi,\mathcal{Z}}^{(\mathcal{F}_{\mathtt{SMT}},\mathcal{F}_{\mathtt{KeyGen}})}$, $\mathtt{Red}(\mathcal{Z})$ can also compute the d_i of all corrupted $S_i \in S$.

This means that the mapping from real to virtual shares creates (except for a negligibly small set of cases) a 1-1 correspondence between successful sequences generated by $\mathtt{Red}(\mathcal{Z})$ and executions of $\mathtt{HYB}_{\pi,\mathcal{Z}}^{(\mathcal{F}_{\mathtt{SMT}},\mathcal{F}_{\mathtt{KeyGen}})}$. Since $\mathcal{Z}$'s view is unchanged under this correspondence, Claim 1 follows.

Since no SIP is even defined in $\mathtt{HYB}_{\pi,\mathcal{Z}}^{(\mathcal{F}_{\mathtt{SMT}},\mathcal{F}_{\mathtt{KeyGen}})}$ it follows from Claim 1 that all j_u are statistically independent of the view of $\mathcal{Z}$ in any attempt until S_{j_u} is corrupted. Since j_u is chosen uniformly at random and $\mathcal{Z}$ corrupts at most $n-1$ parties, it follows that in any given attempt, with probability at least statistically close to $1/n$ the environment $\mathcal{Z}$ does not corrupt S_{j_u}. From this it easily follows that after kn reruns we get a successful attempt except with negligible probability. Since the number of operational phases is polynomial it follows that $\mathtt{Red}(\mathcal{Z})$ also gets a successful sequence, except with negligible probability. From Claim 1 it also follows that the unforgeability property fails with probability statistically close to $P(\mathcal{Z})$ in this successful sequence. Now, every time the unforgeability property fails in $\mathtt{Red}(\mathcal{Z})$, by some party P_i outputting $(\mathit{Verified}, \mathit{sid}, m, \sigma, v, 1)$, it by definition holds that $\sigma^e \bmod N = H(m)$ and that m was not signed in the the successful sequence. Therefore $\mathtt{Red}(\mathcal{Z})$ never queried $\mathcal{O}(N,d)$ on m in the successful attempts used to produce the successful sequence.

It is tempting to believe that $\mathtt{Red}(\mathcal{Z})$ could just output (m,σ) and break the signature scheme with probability statistically close to $P(\mathcal{Z})$. However, this may not work as $\mathtt{Red}(\mathcal{Z})$ also makes queries to $\mathcal{O}(N,d)$ in the failed attempts. If m was queried in a failed attempt, $\mathtt{Red}(\mathcal{Z})$ does not break the signature scheme by outputting (m,σ). Below we will say that a message on which the simulator queried $\mathcal{O}(N,d)$ during a failed attempt and for which $\mathcal{Z}$ did not request a signature generation in the successful sequence is a *dirty message*. When the environment outputs a forgery on a dirty message m in the final state, then we have a situation where $\mathcal{Z}$ produced a successful forgery, but where $\mathtt{Red}(\mathcal{Z})$ cannot use this forgery as its own. Accordingly, successful forgeries by $\mathcal{Z}$ on dirty messages are called *useless forgeries*.

To see that useless forgeries are a real problem, consider the following environment $\mathcal{Z}$: it runs for k operational phases and in phase i picks a random message m_i from the set $\{0,1,\ldots,k\}$ which was not signed already, and then inputs $(\mathit{Sign}, \mathit{sid}, m_i)$ to all parties. After the signature is generated, $\mathcal{Z}$ corrupts all parties except one (at the end of any operational phase, it leaves all parties). After k phases it outputs a forgery (m_{k+1},σ) on the single message $m_{k+1} \in \{0,1,\ldots,k\}$ which was not signed yet. It is easy to see that $\mathtt{Red}(\mathcal{Z})$ will have to rerun each operational phase an expected n times, and that the probability that m_{k+1} was not signed in any failed attempt thus is negligible. This shows that the reduction $\mathtt{Red}$ does not work for all $\mathcal{Z}$. So, we must come up with a better simulation strategy.

First of all we can assume that $\mathtt{Red}(\mathcal{Z})$ never queries $\mathcal{O}(N,d)$ on the same message m twice by having it remember previous queries (here we use that RSA signatures are unique). For a run of $\mathtt{Red}(\mathcal{Z})$ we then use $(m_1,\ldots,m_L)$ to denote the distinct messages on which $\mathtt{Red}(\mathcal{Z})$ queried $\mathcal{O}(N,d)$, in the order of query. Furthermore, when $\mathtt{Red}(\mathcal{Z})$ produces a useless forgery on some dirty message m we define l_0 by $m_{l_0} = m$, and when $\mathtt{Red}(\mathcal{Z})$ produces a useful forgery or no forgery we let $l_0 = 0$. Clearly, given $\mathcal{Z}$ and the randomness $\mathfrak{r}$ used by $\texttt{Red}(\mathcal{Z})$, the value l_0 is uniquely defined by some function $l_0 = l_0(\mathcal{Z}, \mathfrak{r})$.

Consider now the following reduction $\texttt{Red}^{l_0}(\mathcal{Z})$ which has access to an oracle for the function l_0. When running with randomness $\mathfrak{r}$, $\texttt{Red}^{l_0}(\mathcal{Z};\mathfrak{r})$ runs $\texttt{Red}(\mathcal{Z};\mathfrak{r})$, but tries to keep m_{l_0} clean. First it queries $l_0 = l_0(\mathcal{Z}, \mathfrak{r})$ and proceeds as follows: When $l_0 = 0$ it just runs $\texttt{Red}(\mathcal{Z};\mathfrak{r})$ (so, $\texttt{Red}^{l_0}(\mathcal{Z};\mathfrak{r}) = \texttt{Red}(\mathcal{Z};\mathfrak{r})$ when $l_0 = 0$). When $l_0 > 0$ it runs $\texttt{Red}(\mathcal{Z};\mathfrak{r})$ with the following changes: initially it just counts on how many distinct messages it queried $\mathcal{O}(N, d)$, until it is about to query on the l_0'th message m_{l_0}. Then it remembers m_{l_0} and *does not query* $\mathcal{O}(N, d)$ on m_{l_0}. After m_{l_0} is defined $\texttt{Red}^{l_0}(\mathcal{Z})$ still runs $\texttt{Red}(\mathcal{Z};\mathfrak{r})$, except that in addition to rerunning when the SIP S_{j_u} is corrupted it also reruns when it is about to query on m_{l_0}, *so that it never queries* $\mathcal{O}(N, d)$ *on* m_{l_0}. Notice that by definition of $l_0 > 0$ the message m_{l_0} would be dirty in $\texttt{Red}(\mathcal{Z};\mathfrak{r})$. So, if $\texttt{Red}(\mathcal{Z};\mathfrak{r})$ was run, the message m_{l_0} would by definition not be requested signed by $\mathcal{Z}$ in the successful sequence. So, all requests by $\mathcal{Z}$ to signed m_{l_0} would occur in failed attempts, because the SIP was corrupted. Therefore the modification in $\texttt{Red}^{l_0}(\mathcal{Z};\mathfrak{r})$ of aborting when m_{l_0} is requested signed only aborts attempts which would also have been aborted by $\texttt{Red}(\mathcal{Z};\mathfrak{r})$. In particular, the successful sequence of $\texttt{Red}^{l_0}(\mathcal{Z};\mathfrak{r})$ is identical to the successful sequence of $\texttt{Red}(\mathcal{Z};\mathfrak{r})$ (so, $\texttt{Red}^{l_0}(\mathcal{Z};\mathfrak{r}) = \texttt{Red}(\mathcal{Z};\mathfrak{r})$ when $l_0 > 0$). It follows that independent of l_0, $\texttt{Red}^{l_0}(\mathcal{Z};\mathfrak{r}) = \texttt{Red}(\mathcal{Z};\mathfrak{r})$. However, since $\texttt{Red}^{l_0}(\mathcal{Z})$ by construction never queries $\mathcal{O}(N, d)$ on m_{l_0} it follows that $\texttt{Red}^{l_0}(\mathcal{Z})$ produces no useless forgeries, so $\texttt{Red}^{l_0}(\mathcal{Z})$ outputs a forgery (m, σ) with probability statistically close to $P(\mathcal{Z})$.

Consider finally the algorithm $\texttt{Red}'(L, \mathcal{Z})$ which runs as follows: It first sample a uniformly random number $l_0' \in [L]$. Then it runs $\texttt{Red}^{l_0}(\mathcal{Z};\mathfrak{r})$ with uniformly random $\mathfrak{r}$, except that when $\texttt{Red}^{l_0}(\mathcal{Z})$ queries the oracle l_0, $\texttt{Red}'(L, \mathcal{Z})$ replies with l_0'. If $L > l_0$, then $l_0' = l_0(\mathcal{Z}, \mathfrak{r})$ with probability $1/L$. Since $\texttt{Red}^{l_0}(\mathcal{Z})$ outputs a forgery with probability statistically close to $P(\mathcal{Z})$, it follows that when $L > l_0$, the algorithm $\texttt{Red}'(L, \mathcal{Z})$ outputs a forgery with probability statistically close to $P(\mathcal{Z})/L$. Assume then that there exists a PPT environment $\mathcal{Z}$ which violates the unforgeability property in $\texttt{HYB}_{\pi,\mathcal{Z}}^{(\mathcal{F}_{\texttt{SMT}}, \mathcal{F}_{\texttt{KeyGen}})}$ with non-negligible probability $P(\mathcal{Z})$. Then there also exists a polynomial bound $L(k)$ on the running time of $\mathcal{Z}$ and thus there exists a PPT algorithm $\texttt{Red}' = \texttt{Red}'(L(k), \mathcal{Z})$ which breaks the unforgeability under chosen message attack of the RSA signature scheme with probability P' statistically close to the non-negligible $P(\mathcal{Z})/L$, a contradiction. $\diamondsuit$

5 Active Security

In this section we sketch how to make the protocol robust. We follow the approach from [Rab98] and [CGJ+99] with some modifications to avoid share exposure.

5.1 The Protocol

Preliminaries

We need a statistically hiding integer commitment scheme *com*, where a commitment to integer a is denoted by $com(a)$ (we suppress here the random coins need to produce the commitment). We assume that the initial key setup generates parameters for such a scheme. We require that the scheme is *linear*. Informally this means that there exists a method to compute from two commitments $com(a)$ and $com(b)$ and an integer c a new commitment $com(a) + c \cdot com(b)$, and if one can open $com(a)$ to a and $com(b)$ to b, one can compute an opening of $com(a)+c\cdot com(b)$ to $z = a+cb$. This opening should reveal essentially no information about a and b except that $z = a + cb$. A commitment scheme with these properties exists, where binding is based on the factoring assumption [FD02].

Using this commitment scheme we can construct a statistically private VSS scheme as follows. Given a secret integer $s \in [0..B]$ in some known interval, pick a degree t polynomial f with $f(0) = sL$ by letting $a_0 = sL$, picking integer coefficients $a_1, \ldots, a_t$ as in [Rab98] and letting $f(x) = \sum_{j=0}^{t} a_j x^j$. Then for $j = 0, 1, \ldots, t$ compute a commitment $c_j = com(a_j)$ and broadcast $c_0, c_1, \ldots, c_t$. Then for $i = 1, \ldots, n$ compute $d_i = \sum_{j=0}^{t} i^j c_j$. Then compute an opening of d_i to $f(i)$ and send this opening to P_i. If P_i does not receive an opening of $\sum_{j=0}^{t} i^j c_j$ it complains and the dealer must broadcast an opening of $\sum_{j=0}^{t} i^j c_j$. If the dealer fails to do so, the VSS is rejected. It is straight-forward to verify that this is a secure integer VSS scheme that hides the shared value information theoretically.

A VSS to a secret s is given by the commitments d_i and we use $[s] = (com(f(1)), \ldots, com(f(n)))$ to denote a VSS to s. Given a VSS $[a] = (com(f(1)), \ldots, com(f(n)))$ and a VSS $[b] = (com(g(1)), \ldots, com(g(n)))$ and an integer c we can compute a VSS $[a] + c \cdot [b] = (com(f(1)) + c \cdot com(g(1)), \ldots, com(f(n)) + c \cdot com(g(n)))$. Clearly, from openings of $[a]$ and $[b]$ the parties can compute an opening of $[a] + c \cdot [b]$.

Generation of Challenges

We will be using several interactive proofs of the standard public coin 3-move form (Σ-protocols), where the prover must answer a challenge. For us, it will always be the case that all players have to give proofs simultaneously. After the opening messages of the proofs have been sent, the challenges are generated as follows: each party picks uniformly random k-bit values $a_i, b_i \in \{0,1\}^k$, deals a VSS of a_i and then broadcasts b_i. The parties open all the VSS's (that were successfully generated) and compute the k-bit values $c_i = b_i \oplus \bigoplus_j a_j$. The string c_i is used as challenge in the proof given by P_i. The rationale for this method is that, although a corrupt P_i will not be able to predict the challenge ahead of time, a simulator can put itself in a position where it knows all a_j before b_i is chosen, and can therefore force c_i to be any desired value.

Key Setup

We have the following requirements on the key setup. First, let $p = 2p' + 1$ and $q = 2q'+1$ be safe primes and let $N = pq$ and let SQ_N be the subgroup of squares in $\mathbb{Z}_N$ (which has order $p'q'$). We let $L = n!$ and require that $\gcd(e, L) = 1$.

The key generator now additionally broadcasts a random element g of order $p'q'$, for $i = 1, \ldots, n$ broadcasts the value $h_i = g^{d_i} \bmod N$, broadcasts public parameters for a commitment scheme as described above, and finally deals a VSS $\alpha_i = [d_i]$.

In the following, let EDL_N be the language for equality of discrete logarithms, where $(a, A, b, B) \in EDL_N$ iff $a, A, b, B \in SQ_N$ and there exists w such that $A = a^w \bmod N$ and $B = b^w \bmod N$.

Signature Generation, with Share Exposure

The only difference from the passive protocol is that after generating an alleged signature σ', the parties check whether $\sigma'^e \bmod N = H(m)$. If this is not the case, then each party P_i has to prove that $(\sigma_i^2 \bmod N, H(m)^2 \bmod N, h_i, g) \in EDL_N$. This is done using the same standard Σ-protocol that was used in [Rab98], but with the above method for generating the challenges. The protocol requires that the inputs are in SQ_N, but this is guaranteed by the key setup and the squarings done.

Let I be the set of i for which P_i failed this proof. The parties then compute the VSS $\alpha_I = \sum_{i \in I} \alpha_i$ and opens it to some value d_I. We have that $d_I = \sum_{i \in I} d_i$. Therefore, the correct signature σ satisfies $\sigma^2 = H(m)^{2(d_{public}+d_I)} \prod_{i \notin I} \sigma_i^2 \bmod N$. Finally, from $\sigma^2 \bmod N, H(m) = \sigma^e \bmod N$, we can easily compute σ, since 2 and e are relatively prime.

Refreshment

At the beginning of refreshment, decorrupted parties may not have reliable information determining their key shares. Therefore, each party P_j sends to each other party all the public information he holds on α_i, for all i. In other words, the commitments to the shares of d_i are sent. This means each player receives n suggestions for α_i, but since a majority will be correct, we may assume that all honest parties now agree on each α_i. Then for each P_i the key share d_i is privately reconstructed from the VSS α_i, i.e., players send the opening information for the commitments in α_i privately to P_i. Decorrupted players may not be able to send correct opening information, but a majority will be able to do so, and this is sufficient.

Next, the refreshment protocol proceeds as in the passive case, with the following changes:

1. In addition to sending $d_{i,j}$ to P_j, party P_i will also broadcast $h_{i,j} = g^{d_{i,j}} \bmod N$, deal a VSS $\alpha_{i,j} = [d_{i,j}]$ and give a zero-knowledge proof that $\alpha_{i,j}$ can be opened to a value $d_{i,j}$ for which $h_{i,j} = g^{d_{i,j}} \bmod N$. The details of this proof is given below.
2. If any of the proofs fails or $h_i \neq g^{d_{i,public}} \prod_{j=1}^{n} h_{i,j} \bmod N$, then P_i is detected as a cheater. Also, if $h_{i,j} \neq g^{d_{i,j}} \bmod N$, then P_j broadcasts a complaint. Then P_i must broadcast $d_{i,j}$ such that $h_{i,j} = g^{d_{i,j}} \bmod N$, and P_j adopts this values. If P_i fails to do so, then P_i is detected as a cheater.
3. For each party P_i which was detected as a cheater, the other parties simulate P_i, as follows. They define $d_{i,i} = d_i$ and let $d_{i,j} = 0$ for $j \neq i$ and let $d_{i,public} = 0$. Notice that $d_i = d_{i,public} + \sum_{j=1}^{n} d_{i,j}$ and that P_j knows $d_{i,j}$, as desired. Then they let $h_{i,i} = h_i$ and let $\alpha_{i,i} = \alpha_i$, and let $h_{i,j} = g^0 \bmod N$ and let $\alpha_{i,j}$ be a default secret sharing of 0. Notice that $h_{i,j} = g^{d_{i,j}} \bmod N$ and $\alpha_{i,j}$ is a secret sharing of $d_{i,j}$, for $j = 1, \ldots, n$, as desired.

4. Finally each P_i computes $d_i^{new} = \sum_{j=1}^{n} d_{j,i}$ and all parties compute $h_i^{new} = \prod_{j=1}^{n} h_{i,j}$ and $\alpha_i^{new} = \sum_{j=1}^{n} \alpha_{i,j}$.

The proof mentioned above proceeds by having each party run the following (in the role of prover) k times in parallel[1].

1. The prover knows some secret $s \in [0..B]$ and has broadcast $h = g^s \bmod N$ and dealt a VSS $\alpha = [s]$.
2. The prover broadcasts $H = g^r \bmod N$ for a uniformly random $r \in_R [0..(B + 2^k)]$ and deals a VSS $\beta = [r]$.
3. The prover is given a challenge $c \in \{0, 1\}$, generated as described earlier.
4. The parties open the VSS $c\alpha+\beta$ to some value z. If $h^c H \bmod N = g^z \bmod N$, then the parties accept the proof.

Using standard techniques, one can argue that this protocol is honest verifier zero-knowledge, and sound relative to the binding property of the commitments used.

5.2 Analysis

We give a sketch of the security analysis. We want to reprove Theorem 2 for the class of actively cheating adversaries. Except for unforgeability, the required properties are straight-forward to verify. In particular, the correctness follows directly from the binding property of the commitment scheme, the soundness of the applied proof systems and the observation that for a decorrupted party P_i, by virtue of the VSS α_i, there is always sufficient backup information at the beginning of refreshment in order to reconstruct the correct value d_i to P_i as his share. Formalizing this requires a rewinding argument to demonstrate that an adversary breaking correctness can break the binding property of the commitments. This rewinding does not cause any problems since first, the reduction is not part of the UC simulator and second, we may assume that we know the factorization of N (but not the trapdoor for the commitment scheme), and so we can simulate perfectly the actions of honest players in all cases, by just following the protocol.

The proof of unforgeability follows the proof from the passive case, with a few additions to the reduction to unforgeability of the signature scheme, as detailed below.

The key generation is simulated as in the passive case with the following addition. Pick a random square $h \bmod N$ (which will have order $p'q'$ except with negligible probability, by choice of N). Let $g = h^e \bmod N$. Then g is also a random element of order $p'q'$, as desired. Notice that $h = g^d \bmod N$. Now, for the consistent parties P_i, let $h_i = g^{d_i} \bmod N$ and let $\alpha_i = [d_i]$, and for the SIP P_{j_0}, let $h_{j_0} = h(\prod_{i \neq j_0} h_i)^{-1} \bmod N$ and let $\alpha_{j_0} = [d_{j_0}]$, such that $h_{j_0} = g^{d'_{j_0}} \bmod N$ where d'_{j_0} is the virtual share of the SIP. It can be seen that all additional values introduced in the simulation have the same distribution as in the protocol, except

[1] We use parallel repetition of a standard protocol with a 1-bit challenge since this gives us soundness with no extra assumptions. If one is willing to make the strong RSA assumption, 1 repetition with a k-bit challenge is sufficient, this follows from results in [FD02].

that α_i is a VSS of the incorrect share d_{j_0} instead of the virtual share d'_{j_0}; This is however unnoticeable as long as the SIP is honest, as the VSS is statistically hiding.

The refreshment protocol is simulated as in the passive case. Additionally, all parties broadcast the values $h^u_{i,j} = g^{d^u_{i,j}} \bmod N$ and deal VSS's $\alpha^u_{i,j} = [d^u_{i,j}]$. The value $h^u_{j_{u-1},j_u} = g^{{d'}^u_{j_{u-1},j_u}} \bmod N$ is computed using the virtual contribution. Since ${d'}^u_{j_{u-1},j_u}$ is defined to be ${d'}^u_{j_{u-1},j_u} = d^{u-1}_{j_{u-1}} - \sum_{i \neq j_{u-1}} d^u_{j_{u-1},i}$, this can be computed as $h^u_{j_{u-1},j_u} = h^{u-1}_{j_u}(\prod_{i \neq j_u} h^u_{j_{u-1},i})^{-1} \bmod N$.

Notice that $\alpha^u_{j_{u-1},j_u} = [d^u_{j_{u-1},j_u}]$ is still computed using the incorrect contribution. This means that the simulator does not know a witness for the proof that $\alpha^u_{j_{u-1},j_u}$ can be opened to a value x such that $h^u_{j_{u-1},j_u} = g^x \bmod N$. Therefore this proof is simulated, as follows. Using the honest verifier zero-knowledge property, the first message in the k proofs are set up such that there exists exactly one string of challenges $c_{j_{u-1}} \in \{0,1\}^k$ which the simulator can answer. Then the simulator waits for the VSS's of the a_i values to be dealt, and using the shares of the honest parties it computes each a_i and broadcasts $b_{j_{u-1}} = c_{j_{u-1}} \oplus \bigoplus_i a_i$. Note that this simulation introduced *no new rewinding.* As a consequence of $\alpha^u_{j_{u-1},j_u}$ being incorrect, the VSS $\alpha^u_{j_u}$ will be incorrect. Again this is not a problem as long as the SIP P_{j_u} is not corrupted.

The signature generation is simulated as in the passive protocol. Additionally, for the consistent parties a proof that $(\sigma_i, H(m), h_i, g) \in EDL_N$ is simulated by following the protocol (as d_i is known). Notice that the signature share of the SIP is computed as to make it $\sigma_{j_u} = H(m)^{-{d'}^u_{j_u}} \bmod N$. Therefore $(g, h^u_{j_u}, H(m)^2, \sigma^2_{j_u}) \in EDL_N$. So we can run the honest verifier simulator for the proof of membership in EDL_N, and in this way generate an opening message for which we can answer one challenge value. As above, we can make the challenge equal this value without rewinding, and hence complete the simulation. As long as the SIP is not corrupted this will give $\mathcal{Z}$ a view statistically close to that of the protocol. As for simulating the value α_I, notice that it is not a problem that the VSS $\alpha^u_{j_u}$ is not correct, as it will never enter the sum α_I when the SIP P_{j_u} is honest.

As in the passive case, the simulation is statistically close to the protocol until the SIP is corrupted, and as argued during the description, we introduced no more rewinding. Therefore the reduction goes through as in the passive case, using the same rewinding technique.

As for efficiency, note that although we introduced some changes compared to Rabin's original protocol, to make our proof go through, the performance is essentially the same: signature generation is constant round and requires broadcasting $O(n(k + \log n))$ bits.

5.3 Signature Generation, without Share Exposure

Because of the model it is considered secure to open the VSS α_I to reveal the value d_I in the signing protocol, as the parties P_i for $i \in I$ are considered corrupted. In practice a party P_i might, however, end up in I just because a network plug was pulled or its network was congested because of a denial of service attack. In such a situation it might not be such a good idea to reveal d_i, as it constitutes a value which the 'adversary' does not know already. We can indeed do better.

Instead of opening the VSS α_I to the value d_I the parties notice that this VSS defines a polynomial f of degree at most t such that $f(0) = d_I L$ and the party P_i is holding an opening of a public commitment $com(f(i))$.

Each party P_i can therefore broadcast the value $h_i = H(m)^{f(i)} \bmod N$ and prove (using standard techniques similar to what we described above) that $com(f(i))$ can be opened to a value $f(i)$ such that $h_i = H(m)^{f(i)} \bmod N$.

This gives the parties at least $t + 1$ of the values $H(m)^{f(i)} \bmod N$. Therefore the parties can use interpolation as described in [Sho00] to compute $H(m)^{f(0)L} \bmod N = H(m)^{d_I L^2} \bmod N$. Then they compute $\sigma' = H(m)^{d_{public} L^2} H(m)^{\sigma_I L^2} \prod_{i \notin I} (\sigma_i)^{L^2} \bmod N = H(m)^{dL^2} \bmod N$. Using that $\gcd(e, L) = 1$ they then compute $H(m)^d \bmod N$ from $H(m)^{dL^2} \bmod N$.

It might seem puzzling that this is adaptively secure, given the similarity to the protocol from [Sho00] which is not known to be adaptively secure. The crucial point is that we applied the technique to compute $H(m)^{d_I L^2} \bmod N$ and not $H(m)^{dL^2} \bmod N$. Since the value d_I can be computed from the shares d_i of the corrupted parties, d_I is known to the simulator in the reduction (as opposed to d). Therefore it can 'simulate' the computation of $H(m)^{d_I L^2} \bmod N$ by simply running the protocol honestly.

References

[Alm05] Jesús F. Almansa. *A Study for Cryptologic Protocols*. PhD thesis, BRICS, University of Aarhus, Department of Computer Science, IT-parken, Aabogade 34, DK-8200 Århus N, Denmark, 2005.

[Boy89] C. Boyd. Digital multisignatures. In Oxford University Press, editor, *Cryptography and Coding*, pages 241–246, 1989.

[Can] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, page 136, 2001. FOCS'01.

[Can04] R. Canetti. Universally composable signature, certification, and authentication. Cryptology ePrint Archive, August 2004. Corrected version of the paper in Proceedings of the 17th IEEE Computer Security Foundations Workshop, pages 219–235, 2004.

[CGJ+99] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. In *LNCS*, volume 1666, pages 98–115, 1999. CRYPTO'99.

[Des87] Yvo Desmedt. Society and group oriented cryptography: A new concept. In *LNCS*, volume 293, pages 120–127, 1987. CRYPTO'87.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *LNCS*, volume 2729, pages 247–264, 2003. CRYPTO'03.

[FD02] E. Fujisaki and I. Damgård. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Proceedings of Asiacrypt 2002: 125-142*, pages 125–142, 2002. ASIACRYPT'02.

[FGMY97a] Y. Frankel, P. Gemmell, P. Mackenzie, and Moti Yung. Optimal resilience proactive public-key cryptosystems. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, page 384, 1997. FOCS'97.

[FGMY97b] Y. Frankel, P. Gemmell, P. Mackenzie, and Moti Yung. Proactive RSA. In *LNCS*, volume 1294, pages 440–454, 1997. CRYPTO'97.

[FGY96] Yair Frankel, Peter Gemmell, and Moti Yung. Witness-based cryptographic program and robust function sharing. In *28th Annual ACM Symposium on Theory of Computing*, pages 499–508, 1996. STOC'96.

[FMY01] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Adaptive security for the additive-sharing based proactive RSA. In *LNCS*, volume 1992, pages 240–263, 2001. PKC'01.

[Fra89] Yair Frankel. A practical protocol for large group oriented networks. In *LNCS*, volume 434, pages 56–61, 1989. EUROCRYPT'89.

[GJK96] Rosario Gennaro, Stanislaw Jarecki, and Hugo Krawczyk. Robust and efficient sharing of RSA functions. In *LNCS*, volume 1109, pages 157–172, 1996. CRYPTO'96.

[JJKY95] M. Jakobsson, S. Jarecki, H. Krawczyk, and Moti Yung. "proactive RSA for constant-size thresholds". Unpublished manuscript, 1995.

[JS05] Stanislaw Jarecki and Nitesh Saxena. Further simplifications in proactive RSA signature schemes. In *LNCS*, volume 3378, pages 510–528, 2005. TCC'05.

[Nie04] Jesper Buus Nielsen. *On Protocol Security in the Cryptographic Model.* PhD thesis, BRICS, University of Aarhus, Department of Computer Science, IT-parken, Aabogade 34, DK-8200 Århus N, Denmark, 2004.

[OY91] R. Ostrovsky and M. Yung. How to withstand mobile virus attack. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 51–59, 1991. PODC'91.

[Rab98] T. Rabin. A simplified approach to threshold and proactive RSA. In *LNCS*, volume 1462, pages 89–104, 1998. CRYPTO'98.

[SDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *26th Annual ACM Symposium on Theory of Computing*, pages 522–533, 1994. STOC'94.

[Sho00] Victor Shoup. Practical threshold signatures. In *LNCS*, volume 1807, pages 207–220, 2000. EUROCRYPT 2000.