Adaptively-Secure Optimal-Resilience Proactive RSA

Yair Frankel*, Philip MacKenzie**, and Moti Yung***

Abstract. When attacking a distributed protocol, an adaptive adversary may determine its actions (e.g., which parties to corrupt), at any time, based on its entire view of the protocol including the entire communication history. In this paper we are concerned with proactive RSA protocols, i.e., robust distributed RSA protocols that rerandomize key shares at certain intervals to reduce the threat of long-term attacks. Here we design the first proactive RSA system that is secure against an adaptive adversaries. The system achieves "optimal-resilience" and "secure space scalability" (namely $O(1)$ keys per user).

1 Introduction

Distributed public-key systems involve public/secret key pairs where the secret key is distributively held by some number of servers. As long as an adversary cannot corrupt a quorum of servers the system remains secure (as opposed to centralized cryptosystems in which the compromise of a single entity breaks the system). Distributed cryptography, including the design of practical distributed cryptosystems, has been an area of extensive research (see surveys [9,23,21]).

In this paper, we present the first proactive RSA system that is both optimally resilient and provably secure against an adaptive adversary. The system is also scalable in that it only requires a number of key shares that is linear in the number of servers. Let us review why such a proactive RSA system is considered one of the hardest amongst the "distributed cryptosystems" problems:

1. RSA is harder to "make distributed" than the discrete logarithm family due to mathematical constraints. The group of RSA exponents (which contains the RSA secret key) has unknown order (as opposed to discrete-log based protocols), which makes distributed manipulations with the partial keys: polynomial interpolation and key re-randomization (typical in "proactive systems" [28]) hard.
2. From an adversarial perspective, the "proactive" system is the most difficult since it must cope with a "mobile" adversary (whereas a regular "robust threshold system" copes only with static adversary)..

* CertCo, yfrankel@cs.columbia.edu
** Information Sciences Research Center, Bell Laboratories, Murray Hill, NJ 07974, philmac@research.bell-labs.com
*** CertCo, moti@cs.columbia.edu

3. For l denoting the number of servers and t the number of misbehaving servers, an "optimally resilient" protocol is the weakest possible constraint, of $l \geq 2t + 1$, (matching the trivial lower bound).
4. The size of keying information per server is an important parameter in determining scalability of a system. For "scalability of secure space," we require each server to store $O(1)$ key shares, which is optimal.
5. The security proof for distributed protocols is usually based on simulating the adversary's view of the protocol. From the point of view of "proving security" against an adaptive adversary using simulatability is the most difficult one because the view of an adaptive adversary changes dynamically based on all information it receives.

Formally, what we show is:

Theorem 1. *There exists an adaptively-secure optimal-resilient space-scalable proactive RSA system.*

2 Background

The "adaptive adversary" challenge:
Let us elaborate more on the problem facing the designers of adaptively-secure distributed protocols. (Very few such systems have been designed e.g., [1,2]). The major difficulty in proving the security of protocols against adaptive adversaries is being able to efficiently simulate (without actually knowing the secret keys) the view of an adversary which may corrupt parties dynamically, depending on its internal "unknown strategy." The adversary's corruption strategy may be based on values of public ciphertexts, other public cryptographic values in the protocol, and the internal states of previously corrupted parties. Let a user's share of the private key be committed by any public value which is based on the share. The commitment may be an encryption of the share, or the computation to generate the signature with the quorum. Suppose the adversary decides to corrupt next the party whose identity matches a one-way hash function applied to the entire communication history of the protocol so far. Now we must simulate the corruption which allows obtaining a share of the key which is consistent with the commitments.

In distributed public-key systems, the problem of adaptive security is exacerbated by the fact that there is generally "public function and related publicly-committed robustness information" available to anyone, which as discussed above, needs to be consistent with internal states of parties that get corrupted. This is the main cause of difficulties in the proof of security. With proactive systems the update has to be done correctly and be connected to the past, which is another source of difficulty when facing an adaptive adversary.
History of proactive RSA:
The work of [24] developed tools to allow proactive security [28] in discrete-log based systems. They also defined the notion of a proactive public-key system. Tools to allow proactive security in RSA-based systems were given in [16,15,30].

(Previous work on threshold and robust threshold RSA systems is given in [10,8,17,22].) None of these tools have been shown to be secure against an adaptive adversary. Recently the notion of security against adaptive adversaries in threshold public-key systems was dealt with (in systems less constrained than ours) [19,3].

Our Contributions and Techniques:
We base our system on the one in [15], since it is optimally resilient, and, as opposed to [16,30], is secure-space scalable (requires only a linear number of RSA key shares). The system in [30] needs $O(l^3)$ RSA key shares. The system in [16] is suitable for small number of participants, but it is not optimally resilient, and may employ an even larger number of keys as the system grows in scale. Our system differs from the one in [15] in that we construct and employ a new set of techniques that make the system secure against an adaptive adversary.

As in previous proactive RSA systems, we perform secret sharing operations over the integers (since for security the order of the group containing the secret key cannot be revealed to the servers). A difficulty with integer operations is maintaining bounded share sizes even after many proactive update (rerandomization) operations. We do this without requiring any zero-knowledge proofs in our update protocol. In fact, we only use zero-knowledge proofs in our function application (signature) protocol, and if one wishes to optimize the system for the corruption-free case, these proofs would only need to be performed if the computed signature is found to be invalid (which is the typical "optimistic" approach to fault-tolerance advocated e.g. in [16]).

Since we must maintain security against an adaptive adversary, we are not able to use Feldman's verifiable secret sharing VSS [14], which is inherently insecure against an adaptive adversary. (Previous protocols, such as [15,30] are based on Feldman VSS, and [30] uses it in particular to avoid the use of zero-knowledge proofs and thus increase some of the efficiency.) Instead, we use a form of Pedersen's VSS [29]. One can think of the commitments in this VSS as **"detached commitments"**. These commitments are used to ensure correct behavior of servers, yet have no "hard attachment" to the rest of the system, even the secret key itself! We show how to work with these detached commitments, e.g., using "function representation transformations" like "poly-to-sum" and "sum-to-poly" (which are basic tools which maintain space-scalability and which we build based on [15]). We also show how to maintain robustness by constructing simulatable "soft attachments" from these detached commitments into the operations of the rest of the system. The soft attachments are constructed using efficient zero-knowledge proofs-of-knowledge. For proactive maintenance we develop update techniques that carefully assure correctness of update, as well as non-growth of the key size as we perform our computations over the integers.

The basic proof technique for security against an adaptive adversary is based on the notion of a "faking server" which is chosen when commitments related to the secret key must be generated. (Fortunately, this is only during a function application phase, where additive sharings are used.) The simulator exploits the "actions" of this server to assure that the view is simulatable while not knowing the secret key. This server is chosen at random and its public actions are indistinguishable from an honest server to the adversary. We have to backtrack the simulation **only if** the adversary corrupts this special server. Since there is only one faking server, and since regardless of its corruption strategy, the adversary has a polynomial chance (at least $1/(t+1)$) of not corrupting this one server, we will be able to complete the simulation in expected polynomial time. We have to assure that this "faking server" technique works in the proactive setting, where the adversary is not just adaptive, but also mobile. Our protocol and simulation techniques also maintain "optimal resilience."

3 Model and Definitions

Participants and Communication: We use the standard model for proactive systems [28]. The system consists of l servers $\mathcal{S} = \{S_1, \ldots, S_l\}$. A server is *corrupted* if it is controlled by the adversary. When corrupted, we assume "for security" that the adversary sees all the information currently on that server. On the other hand, the system should not "open" secrets of unavailable servers (effectively reducing the needed threshold). Namely, we separate availability faults from security faults (and do not cause potential security exposures due to unavailability). Our communication model is similar to [25]. All participants communicate via an authenticated bulletin board in a synchronized manner. We assume that the adversary cannot jam communication.

Time periods: To deal with a mobile adversary, we assume a common global clock (e.g., a day, a week, etc.) that divides time into two types of *time periods* repeated in sequence: an **update period** (odd times) and an **operational period** (even times). During an operational period, the servers can perform functions using the current secret key shares. During the update period the servers engage in an interactive *update protocol* which upon completion the servers hold new shares to be used during the following operational period[1].

System Management: We assume that a server that is determined to be corrupted by a majority of active servers can be *refreshed* (i.e., erased and rebooted, or perhaps replaced by a new server with a blank memory) by some underlying system management. This is a necessary assumption for dealing with corrupted servers in any proactive system.

The adversary: The adversary is t**-restricted**; namely it can, during each period, corrupt at most t servers, under the assumption that if a server is corrupt during an update period it is corrupt in the prior and subsequent operational periods. A **mobile** adversary moves around as servers become corrupted or uncorrupted. The actions of an adversary at any time may include submitting messages to

[1] Technical note: we consider a server that is corrupted during an update phase as being corrupted during both its adjacent periods. This is because the adversary could learn the shares used in both the adjacent operational periods.

the system to be signed, corrupting servers, and broadcasting arbitrary information on the communication channel. The adversary is **adaptive**; namely it is allowed to base its actions not only on previous function outputs, but on all the information that it has previously obtained.
Distributed Public-Key Systems: Here we formally define our notions of security and robustness for proactive public-key systems. For simplicity, we will assume that the function application operation computes signatures.

Definition 1. (Robustness of a Proactive System) *A (t, l)-proactive public-key system S is robust if for any polynomial-time t-restricted adaptive mobile adversary $\mathcal{A}$, with all but negligible probability, after polynomially-many update protocols for each input m which is submitted to the signing protocol during an operational period, the resulting signature s is valid.*

Definition 2. (Security of a Proactive System) *A (t, l)-proactive public-key system S is secure if for any polynomial-time t-restricted adaptive mobile adversary $\mathcal{A}$, after polynomially-many signing protocols performed during operational periods on given values, and polynomially-many update protocols, given a new value m and the view of $\mathcal{A}$, the probability of being able to produce in polynomial time a signature s on m that is valid is negligible.*

Remark: The choice of the inputs to the signing protocol prior to the challenge m defines the *tampering power* of the adversary (i.e., "known message," "chosen message", "random message" attacks). The choice depends on the implementation within which the distributed system is embedded. In this work, we assume that the (centralized) cryptographic function is secure with respect to the tampering power of the adversary. We note that the provably secure signature and encryption schemes typically activate the cryptographic function on random values (decoupled from the message choice of the adversary).

3.1 Range Notation

We will use the following notation to define ranges:

1. A range is defined as $[a, b]$, denoting all integers between a and b inclusive. (Similarly, we could define (a, b), $(a, b]$, and $[a, b)$.)
2. We can multiply a range by a positive scalar: $x \cdot [a, b]$ denotes the range $[xa, xb]$.
3. We can multiply a range by a negative scalar: $-x \cdot [a, b]$ denotes the range $[-xb, -xa]$.
4. We can add two ranges: $[a, b] + [c, d]$ denotes the range $[a + c, b + d]$.
5. For a range that covers the point zero, we can multiply that range by a positive or negative scalar: $\pm x \cdot [a, b]$, where $a \leq 0 \leq b$, denotes the range $[-xc, xc]$, where $c = \max(|a|, |b|)$.

4 Basics for Our System

Let k be the security parameter. Let key generator GE define a family of RSA functions to be $(e, d, N) \leftarrow GE(1^k)$ such that N is a composite number $N = P * Q$ where P, Q are prime numbers of $k/2$ bits each. The exponent e and modulus N are made public while $d \equiv e^{-1} \bmod \lambda(N)$ is kept private.[2] The **RSA encryption function** is public, defined for each message $M \in Z_N$ as: $C = C(M) \equiv M^e \bmod N$. The **RSA decryption function** (also called signature function) is the inverse: $M = C^d \bmod N$. It can be performed by the owner of the private key d. Formally the RSA Assumption is stated as follows.

RSA Assumption Let k be the security parameter. Let key generator GE define a family of RSA functions (i.e., $(e, d, N) \leftarrow GE(1^k)$ is an RSA instance with security parameter k). For any probabilistic polynomial-time algorithm A, $\Pr[u^e \equiv w \bmod N : (e, d, N) \leftarrow GE(1^k); w \in_R \{0,1\}^k; u \leftarrow A(1^k, w, e, N)]$ is negligible.

Recall that the RSA assumption implies the intractability of factoring products of two large primes.

Next we describe variants of Shamir secret sharing and Pedersen VSS that we use. They differ in that operations on the shares are performed over the integers, instead of in a modular subgroup of integers.

(t,l)-secret sharing over the integers [15]: This primitive is based on Shamir secret sharing [32]. Let $L = l!$ and let β, K be positive integers. For sharing a secret $s \in [0, K]$, a random polynomial $a(x) = \sum_{j=0}^{t} a_j x^j$ is chosen such that $a_0 = L^2 s$, and each other $a_j \in_R \{0, L, 2L, \ldots, \beta L^3 K\}$.[3] Each shareholder $i \in \{1, \ldots, l\}$ receives a secret share $s_i = a(i)$, and verifies[4] that (1) $0 \le s_i \le \beta L^3 K l^t (t+1)$, and (2) L divides s_i. Any set of shareholders of cardinality $t+1$ can compute s using Lagrange interpolation, i.e. $s = a(0) = \sum_{i \in \Lambda} a(i) z_{i,\Lambda}$, where $z_{i,\Lambda} = \prod_{j \in \Lambda \setminus \{i\}} (i-j)^{-1}(0-j)$.

(t,l)-Unconditionally-Secure VSS over the Integers (INT-(t,l)-US-VSS): This primitive is based on Pedersen Unconditionally-Secure (t,l)-VSS [29], and is slightly different than the version in [20]. Let N be an RSA modulus and let g and h be maximal order elements whose discrete log modulo N with respect to each other is unknown. The protocol begins with two (t,l)-secret sharings over the integers with $\beta = N$, the first sharing secret $s \in [0, K]$, and the second sharing $s' \in [0, N^2 K]$. Let $a(x) = \sum_{j=0}^{t} a_j x^j$ be the random polynomial used in

[2] $\lambda(N) = \mathrm{lcm}(P-1, Q-1)$ is the smallest integer such that $x^{\lambda(N)} \equiv 1 \bmod N$ for any $x \in Z_N^*$. We use $\lambda(N)$, rather than more traditionally $\phi(N)$, because it explicitly describes an element of maximal order in Z_N^*.

[3] We note that $L^2 s$ is actually the secret component of the secret key, which when added to a public leftover component (in $[0, L^2 - 1]$), forms the RSA secret key.

[4] These tests only verify the shares are of the correct form, not that they are correct polynomial shares.

sharing s and let $a'(x) = \sum_{j=0}^{t} a'_j x^j$ be the random polynomial used in sharing s'. For all i, S_i receives shares $s_i = a(i)$ and $s'_i = a'(i)$. (We refer to the pair $(a(i), a'(i))$ as *dual-share* i.) Also, the *verification shares* $\{\alpha_j (= g^{a_j} h^{a'_j})\}_{0 \le j \le t}$, are published.[5] Call *check share* $A_i = \prod_{j=0}^{t} \alpha_j^{i^j}$. S_i can verify the correctness of his shares by checking that $A_i \overset{?}{=} g^{s_i} h^{s'_i}$. Say s and s' are the shares computed using Lagrange interpolation from a set of $t+1$ shares that passed the verification step. If the dealer can reveal different secrets $\hat{s}$ and $\hat{s}'$ that also correspond to the zero coefficient verification share, then the dealer can compute an α and β such that $g^\alpha \equiv h^\beta$, which implies factoring (and thus breaking the RSA assumption).

Looking ahead, we will need to simulate an INT-(t, l)-US-VSS. We can do this by constructing a random polynomial over an appropriate simulated secret (e.g., a random secret, or a secret obtained as a result of a previously simulated protocol) in the zero coefficient, and a random companion polynomial with a totally random zero coefficient.

5 Techniques

In order to "detach" ciphertexts from their cleartext values we employ semantically-secure non-committing encryption [1]. In fact, our (full) security proofs first assume perfectly secret channels and then add the above (a step we omit here).

A more involved issue concerns the public commitments. The collection of techniques needed to underly distributed public-key systems include: distributed representation methods (polynomial sharing, sum (additive) sharing), representation transformers which move between different ways to represent a function (poly-to-sum, sum-to-poly), as well as a set of "elementary" distributed operations (add, multiply, invert). For example, the "poly-to-sum" protocol is executed by $t+1$ servers at a time, and transforms a (t, l)-secure polynomial-based sharing to an additive sharing with $t+1$ shares. We need to have such techniques (motivated by [20,15]) which are secure and robust against adaptive adversaries. We will rely on new zero-knowledge proof techniques (see Appendix A), as well as on shared representation of secrets as explained in Section 4. The notation "2poly" refers to a polynomial and its companion polynomial shared with INT-(t, l)-US-VSS (which is "unconditionally secure VSS"). The notation "2sum" refers to two additive sharings, with check shares that contain both additive shares of a server (similar to the check shares in INT-(t, l)-US-VSS). In describing the protocols, unless otherwise noted we will assume multiplication is performed mod N and addition (of exponents) is performed over the integers (i.e., not "mod" anything).

We define the following ranges, using the notation described in Section 3.1.

1. Let $[\text{RANGE}_1] = [0, (t+1)L^3N^3l^t]$ and $[\text{RANGE}'_1] = [0, (t+1)L^3(t+1)N^7l^t]$.
2. Let $[\text{RANGE}_2] = \pm(t+1)L \cdot [\text{RANGE}_1]$ and $[\text{RANGE}'_2] = \pm(t+1)L \cdot [\text{RANGE}'_1]$.
3. Let $[\text{RANGE}_3] = \pm(t+1) \cdot [\text{RANGE}_2] + [0, L^2N] + (t+1) \cdot [0, L^2]$ and $[\text{RANGE}'_3] = \pm(t+1) \cdot [\text{RANGE}'_2] + [0, L^2N^3] + (t+1) \cdot [0, L^2]$.

[5] We implicitly assume all verification operations are performed in Z_N^*.

4. Let $[\textsc{Range}_4] = (t+1)\cdot[\textsc{Range}_2] + (t+1)\cdot[0,L^2] + [\textsc{Range}_3]$ and $[\textsc{Range}'_4] = (t+1)\cdot[\textsc{Range}'_2] + (t+1)\cdot[0,L^2] + [\textsc{Range}'_3]$.
5. Let $[\textsc{Range}_5] = L\cdot[\textsc{Range}_4] - (t+1)\cdot[0, L^2N^2/(t+1)]$ and $[\textsc{Range}'_5] = L\cdot[\textsc{Range}'_4] - (t+1)\cdot[0,L^2N^5]$.

5.1 2poly-to-2sum

The goal of 2poly-to-2sum (Figure 1) is to transform t-degree polynomials $a()$ and $a'()$ used in INT-(t,l)-US-VSS into $t+1$ additive shares for each secret $a(0)$ and $a'(0)$, with corresponding check shares. The idea is to perform interpolation.[6] We note that in Step 2 each s_i and s'_i is a multiple of L, so S_i can actually compute b_i and b'_i over the integers.

1. **Initial configuration:** INT-(t,l)-US-VSS (parameters: (N,g,h)) with t-degree polynomials $a()$ and $a'()$, and a set Λ of $t+1$ server indices. For all $i\in\Lambda$, recall S_i holds shares s_i and s'_i with corresponding check share $A_i = g^{s_i}h^{s'_i}$.
2. For all $i\in\Lambda$, S_i computes the additive shares $b_i = s_i z_{i,\Lambda}$ and $b'_i = s'_i z_{i,\Lambda}$ and publishes $B_i = g^{b_i}h^{b'_i} = A_i^{z_{i,\Lambda}}$.
3. All servers verify B_i for all $i\in\Lambda$ using $(A_i)^{V_{i,\Lambda}} \equiv (B_i)^{V'_{i,\Lambda}}$ where $V_{i,\Lambda} = \prod_{j\in\Lambda\setminus\{i\}}(0-j)$ and $V'_{i,\Lambda} = \prod_{j\in\Lambda\setminus\{i\}}(i-j)$. If the verification for a given B_i fails, each server broadcasts a (Bad,i) message and quits the protocol.

Fig. 1. 2poly-to-2sum

Note the following ranges:

– For $i\in\Lambda$, if S_i is good then $s_i\in[\textsc{Range}_4]$ and $s'_i\in[\textsc{Range}'_4]$. (This will be shown later.)

5.2 2sum-to-2sum

The goal of 2sum-to-2sum is to randomize additive dual-shares (most likely obtained from a 2poly-to-2sum) and update the corresponding check shares. The scheme is in Figure 2.

Note the following ranges:

– For $i\in\Lambda$, if S_i is good then $d_i\in[0,L^2N^2]$ and $d'_i\in[0,(t+1)L^2N^4]$.
– We will show that $d_*\in[\textsc{Range}_3]$ and $d'_*\in[\textsc{Range}'_3]$ (unless RSA is broken).

[6] In [15], poly-to-sum also performed a rerandomization of the additive shares. We split that into a separate protocol for efficiency, since sometimes 2poly-to-2sum is used without a rerandomization of additive shares.

1. **Initial configuration:** There is a set Λ of $t+1$ server indices. For all $i \in \Lambda$, S_i holds additive dual-share (b_i, b_i'), with corresponding check share $B_i = g^{b_i} h^{b_i'}$.
2. For all $i \in \Lambda$, S_i chooses $r_{i,j} \in_R Z_{L^2 N^2/(t+1)}$ and $r_{i,j}' \in_R Z_{L^2 N^4}$, for $j \in \Lambda$.
3. For all $i \in \Lambda$, S_i publishes $r_{i,*} = b_i - \sum_{j \in \Lambda} r_{i,j}$ and $r_{i,*}' = b_i' - \sum_{j \in \Lambda} r_{i,j}'$.
4. For all $i \in \Lambda$, S_i privately transmits $r_{i,j}$ and $r_{i,j}'$ to all S_j for $j \in \Lambda \setminus \{i\}$.
5. For all $i \in \Lambda$, S_i publishes $R_{i,j} = g^{r_{i,j}} h^{r_{i,j}'}$ for $j \in \Lambda \setminus \{i\}$.
6. All servers can compute $R_{i,i} = B_i / g^{r_{i,*}} h^{r_{i,*}'} \prod_{j \in \Lambda \setminus \{i\}} R_{i,j}$ for all $i \in \Lambda$.
7. For all $j \in \Lambda$, S_j verifies that each $r_{i,j} \in Z_{L^2 N^2/(t+1)}$, each $r_{i,j}' \in Z_{L^2 N^4}$, each $r_{i,*} \in [\text{RANGE}_5]$, each $r_{i,*}' \in [\text{RANGE}_5']$, and that $R_{i,j} \equiv g^{r_{i,j}} h^{r_{i,j}'}$. If the verification fails, S_j broadcasts an (Accuse,i,j) message, to which S_i responds by broadcasting $r_{i,j}$ and $r_{i,j}'$. If S_i does not respond, $r_{i,j}$ or $r_{i,j}'$ is not in the correct range, or $R_{i,j} \not\equiv g^{r_{i,j}} h^{r_{i,j}'}$ (which all servers can now test), then each server broadcasts a (Bad,i) message and quits the protocol.
8. For all $j \in \Lambda$, S_j computes $d_j = \sum_{i \in \Lambda} r_{i,j}$, $d_j' = \sum_{i \in \Lambda} r_{i,j}'$, and $D_j = \prod_{i \in \Lambda} R_{i,j}$.
9. All servers compute *leftover shares* $d_* = \sum_{i \in \Lambda} r_{i,*}$ and $d_*' = \sum_{i \in \Lambda} r_{i,*}'$

Fig. 2. 2sum-to-2sum

5.3 2sum-to-1sum

2sum-to-1sum (Figure 3) is employed in computing partial signatures using the first parts of the additive dual-shares (obtained from 2sum-to-2sum) as the exponents, and in proving the partial signatures correct. These proofs form the "soft attachments" from the information-theoretically secure check shares to the computationally secure check shares that must correspond to the actual secret.

1. **Initial configuration:** Parameters (N, e, g, h). There is a set Λ of $t+1$ server indices. For all $i \in \Lambda$, S_i holds additive dual-share (d_i, d_i'), with corresponding check share $D_i = g^{d_i} h^{d_i'}$. Also, all servers S_i with $i \in \Lambda$ have performed a ZK-proof-setup protocol ZKSETUP-RSA(N, e, g) with all other servers.
2. For all $i \in \Lambda$, S_i broadcasts $E_i = m^{d_i}$, where m is the message to be signed, or more generally, the value to which the cryptographic function is being applied.
3. For all $i \in \Lambda$, S_i performs a ZK-proof of knowledge ZKPROOF-DL-REP$(N, e, m, g, h, E_i, D_i)$ with all other servers. Recall that this is performed over a broadcast channel so all servers can check if the ZK-proof was performed correctly.
4. If a server detects that for some $i \in \Lambda$, S_i fails to perform the ZK-proof correctly, that server broadcasts a message (Bad,i) and quits the protocol.

Fig. 3. 2sum-to-1sum

5.4 2sum-to-2poly

The goal of 2sum-to-2poly is to transform $t+1$ additive dual-shares with their corresponding check shares (obtained from 2sum-to-2sum) into polynomial sharings of the same secrets. The idea is for each shareholder to perform an INT-(t,l)-US-VSS with its shares as the secrets, and then to sum the resulting polynomials. The protocol is given in Figure 4

1. **Initial configuration:** Parameters (N,e,g,h). There is a set Λ of server indices. For all $i \in \Lambda$, S_i holds additive dual-share (d_i, d'_i), with corresponding check share $D_i = g^{d_i} h^{d'_i}$, and all servers know leftover shares d_* and d'_*.
2. For $i \in \Lambda$, S_i broadcasts $r_i = d_i \bmod L^2$ and $r'_i = d'_i \bmod L^2$.
3. For $i \in \Lambda$, S_i sets $e_i = d_i - r_i$, $e'_i = d'_i - r'_i$, and $E_i = g^{e_i} h^{e'_i}$.
4. Note that all e_i and e'_i are multiples of L^2. These are then shared using INT-(t,l)-US-VSS (over the appropriate ranges, and without any additional L^2 factor), say with polynomials $v_i()$ and $v'_i()$.
5. For all $j \in \Lambda$, S_j verifies that each $r_i, r'_i \in Z_{L^2}$, and verifies its shares of each INT-(t,l)-US-VSS. (For ranges, S_j must verify that each $v_i(j) \in [\textsc{Range}_1]$ and each $v'_i(j) \in [\textsc{Range}'_1]$.) If a verification fails for the INT-(t,l)-US-VSS from S_i, S_j broadcasts an (Accuse,i,j) message, to which S_i responds by broadcasting $v_i(j)$ and $v'_i(j)$. If S_i does not respond, $v_i(j)$ or $v'_i(j)$ is not in the correct range, or the verification shares do not match for $v_i(j)$ and $v'_i(j)$ (which all servers can now test), then each server broadcasts a (Bad,i) message and quits the protocol.
6. For all j, S_j computes the sums $v(j) = d_* + \sum_{i\in\Lambda} r_i + \sum_{i\in\Lambda} v_i(j)$ and $v'(j) = d'_* + \sum_{i\in\Lambda} r'_i + \sum_{i\in\Lambda} v'_i(j)$.
   The verification shares for $v()$ and $v'()$ can be computed from the verification shares for $v_i()$ and $v'_i()$, for $i \in \Lambda$.

Fig. 4. 2sum-to-2poly

Note the following ranges:

- For $i \in \Lambda$, if S_i is good then $e_i \in [0, L^2N^2]$ and $e'_i \in [0, (t+1)L^2N^4]$. Also, there is a range of size L^2N^4 for which an additive part of e'_i was randomly chosen by S_i itself, and this provides the simulatability of $v_i()$.
- Assuming $(t+1)$ good servers check the ranges of their shares of $v_i()$ and $v'_i()$, then it can be deduced that $v_i(0) \in [\textsc{Range}_2]$ and $v'_i(0) \in [\textsc{Range}'_2]$.
- Since $d_i = v_i(0)$ and $d'_i = v'_i(0)$ (unless RSA is broken, as we will prove), and since $d_* + \sum_{i\in\Lambda} d_i = L^2 s$ and $d'_* + \sum_{i\in\Lambda} d'_i = L^2 s'$ (again, unless RSA is broken), we have that $d_* \in [\textsc{Range}_3]$ and $d'_* \in [\textsc{Range}'_3]$.
- Using the facts above, we conclude that for $j \in \Lambda$, if S_j is good then $v(j) \in [\textsc{Range}_4]$ and $v'(j) \in [\textsc{Range}'_4]$, i.e., for the next operational round, $s_j \in [\textsc{Range}_4]$ and $s'_j \in [\textsc{Range}'_4]$, which is what we stated in the 2poly-to-2sum protocol.

6 RSA Protocols

We now present protocols for threshold function application and proactive maintenance. Their security and robustness proofs are available from the authors [18].

6.1 Threshold Function Application

The protocol is given in Figure 5

1. **Initial configuration:** INT-(t,l)-US-VSS (parameters: (N,e,g,h)) with t-degree polynomials $a()$ and $a'()$. Also, each server maintains a list $\mathcal{G}$ of server indices for servers that have not misbehaved (i.e., they are considered good). A message m needs to be signed.
2. A set $\Lambda \subseteq \mathcal{G}$ with $|\Lambda| = t+1$ is chosen in some public way.
3. 2poly-to-2sum is run. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 2.
4. 2sum-to-2sum is run. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 2.
5. 2sum-to-1sum is run. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 2. If there is no misbehavior, the signature on m can be computed from the partial signatures generated in this step, along with the leftover shares.
6. All values created during the signing protocol for m are erased.

Fig. 5. Function Application Protocol

6.2 Proactive Maintenance

For proactive maintenance, we perform an update by running 2poly-to-2sum on the secret polynomials, followed by 2sum-to-2sum and 2sum-to-2poly. After 2sum-to-2poly, each server erases all previous share information, leaving just the new polynomial shares and verification shares. If there is misbehavior by any server, the procedure is restarted with new participants (here restarts do not introduce statistical biases and do not reduce the protocol's security). The protocol is given in Figure 6, where the full Proactive RSA protocol is in Figure 7.

References

1. D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Advances in Cryptology—EUROCRYPT 92*, volume 658 of *Lecture Notes in Computer Science*, pages 307–323. Springer-Verlag, 24–28 May 1992. 181, 186
2. R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In STOC'96 [33], pages 639–648. 181
3. R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security of threshold systems. to appear in Crypto'99, 1999. 182
4. R. Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, University of Amsterdam, 1995. 193

1. **Initial configuration:** INT-(t,l)-US-VSS (parameters: (N,e,g,h)) with t-degree polynomials $a()$ and $a'()$
2. Each server maintains a list $\mathcal{G}$ of server indices for servers that have not misbehaved (i.e., they are considered good).
3. Each (ordered) pair of servers (S_i,S_j) performs ZKSETUP-RSA$_{S_i,S_j}(N,e,g,h)$ (using new commitment values). This setup will be used for all proofs performed during the following operational period.
4. A set $\Lambda \subseteq \mathcal{G}$ with $|\Lambda| = t+1$ is chosen in some public way.
5. 2poly-to-2sum is run. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 4.
6. 2sum-to-2sum is run. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 4.
7. 2sum-to-2poly is run. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 4.
8. All previous share information is erased.

Fig. 6. Proactive Maintenance (Key Update) Protocol

5. R. Cramer, I. Damgård, and P. MacKenzie. Zk for free: the case of proofs of knowledge. manuscript, 1999. 193
6. *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990, 20–24 Aug. 1989. 191, 193
7. *Advances in Cryptology—CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992, 11–15 Aug. 1991. 191, 193
8. A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely (extended summary). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 522–533, Montréal, Québec, Canada, 23–25 May 1994. 182
9. Y. Desmedt. Threshold cryptosystems. In J. Seberry and Y. Zheng, editors, *Advances in Cryptology—AUSCRYPT '92*, volume 718 of *Lecture Notes in Computer Science*, pages 3–14, Gold Coast, Queensland, Australia, 13–16 Dec. 1992. Springer-Verlag. 180
10. Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures (extended abstract). In CRYPTO'91 [7], pages 457–469. 182
11. C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. In STOC'98 [34], pages 409–428. 193
12. C. Dwork and A. Sahai. Concurrent zero-knowledge: Reducing the need for timing constraints. In Krawczyk [27], pages 442–457. 193
13. U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. In CRYPTO'89 [6], pages 526–545. 193
14. P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science*, pages 427–437, Los Angeles, California, 12–14 Oct. 1987. IEEE. 182
15. Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Optimal-resilience proactive public-key cryptosystems. In *38th Annual Symposium on Foundations of Computer Science*, pages 384–393, Miami Beach, Florida, 20–22 Oct. 1997. IEEE. 181, 182, 185, 186, 187

1. The dealer generates an RSA public/private key (N, e, d), and computes public value x_{pub} and secret value $x \in [0, N]$ such that $d \equiv x_{pub} + L^2 x \bmod \phi(N)$, as in [16].[a] Then the dealer chooses generators $g, h \in_R Z_N^*$, $x' \in_R Z_{N^3}$, and an INT-(t, l)-US-VSS on secrets x, x' with parameters (N, g, h).
2. Each (ordered) pair of servers (S_i, S_j) performs ZKSETUP-RSA$_{S_i,S_j}(N, e, g, h)$.
3. Each server maintains a list $\mathcal{G}$ of server indices for servers that have not misbehaved (i.e., they are considered good). It also maintains public parameters: $(N, e, g, h, x_{pub}, l, t)$ and the verification shares of the INT-(t, l)-US-VSS polynomials $a()$ and $a'()$ which have $a(0) = x$ and $a'(0) = x'$.
4. When a message m needs to be signed, the servers agree on the public parameters, then the Function Application protocol is run.
5. When an update is scheduled to occur, the servers agree on the public parameters, then the Proactive Maintenance protocol is run.

[a] Recall that x_{pub} is computed using only the public values N, e, L.

Fig. 7. Proactive Threshold Protocol

16. Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Proactive RSA. In *Advances in Cryptology—CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 440–454. Springer-Verlag, 17–21 Aug. 1997. 181, 182, 192
17. Y. Frankel, P. Gemmell, and M. Yung. Witness-based cryptographic program checking and robust function sharing. In STOC'96 [33], pages 499–508. 182
18. Y. Frankel, P. D. MacKenzie, and M. Yung. Manuscript of current paper with complete proof. 190
19. Y. Frankel, P. D. MacKenzie, and M. Yung. Adaptively-secure distributed public-key systems. In *European Symposium on Algorithms—ESA '99*, volume 1643 of *Lecture Notes in Computer Science*, pages 4–27. Springer-Verlag, 16–18 July 1998. 182
20. Y. Frankel, P. D. MacKenzie, and M. Yung. Robust efficient distributed rsa-key generation. In STOC'98 [34], pages 663–672. 185, 186, 193
21. Y. Frankel and M. Yung. Distributed public-key cryptosystems. In H. Imai and Y. Zheng, editors, *Advances in Public Key Cryptography—PKC '98*, volume 1431 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, Feb. 1998. invited talk. 180
22. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of RSA functions. In *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 157–172. Springer-Verlag, 18–22 Aug. 1996. 182
23. S. Goldwasser. Multi-party computations: Past and present. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–6, 1997. invited talk. 180
24. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public-key and signature schemes. In *Proceedings of the Third Annual Conference on Computer and Communications Security*, pages 100–110, 1996. 181
25. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing, or: How to cope with perpetual leakage. In *Advances in Cryptology—CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352. Springer-Verlag, 27–31 Aug. 1995. 183

26. J. Kilian, E. Petrank, and C. Rackoff. Lower bounds for zero knowledge on the internet. In *39th Annual Symposium on Foundations of Computer Science*, pages 484–492. IEEE, Nov. 1998. 193
27. H. Krawczyk, editor. *Advances in Cryptology—CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*. Springer-Verlag, 17–21 Aug. 1998. 191, 193
28. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 51–61, 1991. 180, 181, 183
29. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In CRYPTO'91 [7], pages 129–140. 182, 185
30. T. Rabin. A simplified approach to threshold and proactive rsa. In Krawczyk [27], pages 89–104. 181, 182
31. C. P. Schnorr. Efficient identification and signatures for smart cards. In CRYPTO'89 [6], pages 239–252. 193
32. A. Shamir. How to share a secret. *Commun. ACM*, 22:612–613, 1979. 185
33. *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, Philadelphia, Pennsylvania, 22–24 May 1996. 190, 192
34. *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, Dallas, Texas, 23–26 May 1998. 191, 192

A Proofs

We use efficient ZK proofs of knowledge (POKs) derived from [20] and [5]. These are composed of combinations of Σ-protocols [4] (i.e., Schnorr-type proofs [31]). For each ZK proof that we need, we will have a separate "proof" protocol, but there will be a single "setup" protocol used for all ZK proofs. Say A wishes to prove knowledge of "X" to B. Then the setup protocol will consist of B making a commitment and proving that he can open it in a witness indistinguishable way [13], and the proof protocol will consist of A proving to B either the knowledge of "X" or that A can open the commitment. (See [5] for details.) This construction allows the proof protocols to be run concurrently without any timing constraints, as long as they are run after all the setup protocols have completed. (For more on the problems encountered with concurrent ZK proofs see [26,11,12].)

The (RSA-based) ZK-proof-setup protocols are exactly the Σ-protocols for commitments over q-one-way-group-homomorphisms (q-OWGH), given in [5]. Recall the q-OWGH for an RSA system with parameters (N, e) is $f(x) = x^e \bmod N$ (with $q = e$ in this case).

Let KE denote the "knowledge error" of a POK.

We define $\text{ZK}\textsc{setup}\text{-RSA}_{A,B}(N, e, g)$ as a protocol in which A generates a commitment C and engages B in a WH POK ($KE = 1/e$)[7] of (σ, σ') (with $\sigma \in Z_e$, $\sigma' \in Z_N^*$) where $C \equiv g^\sigma(\sigma')^e \bmod N$.

We define $\text{ZK}\textsc{proof}\text{-DL-REP}_{A,B}(N, e, m, g, h, E, D)$ as a protocol in which A, who knows integers $d \in (-a, a]$ and $d' \in (-b, b]$ such that $E \equiv m^d \bmod$

[7] This implies e must be exponentially large in the security parameter k in order to obtain a sound proof. However, if e is small (say $e = 3$) we can use different setup and proof protocols described in [5] to obtain provably secure and robust RSA-based protocols.

N and $D \equiv g^d h^{d'} \bmod N$, engages B in a WH POK ($KE = 1/e$) of either $\Delta \in Z_e$, $\delta \in (-2ae(N+1), 2ae(N+1)]$, and $\delta' \in (-2be(N+1), 2be(N+1)]$ where $D^\Delta \equiv g^\delta h^{\delta'} \bmod N$ and $E^\Delta \equiv g^\delta \bmod N$, or (τ, τ') (with $\tau \in Z_e$, $\tau' \in Z_N^*$) where $C_{B,A} \equiv g^\tau (\tau')^e \bmod N$ and $C_{B,A}$ is the commitment generated in ZKSETUP-RSA$_{B,A}(N, e, g)$. This protocol is honest-verifier statistical zero-knowledge with a statistical difference between the distribution of views produced by the simulator and in the real protocol bounded by $2/N$.

A.1 Proof of Representations

Here we give the main Σ-protocol used in ZKPROOF-DL-REP$_{A,B}(N, e, m, g, h, E, D)$.[8]

1. Initially, the parameters (N, e, m, g, h, E, D) are public, and A knows integers $d \in (-a, a]$ and $d' \in (-b, b]$ such that $E \equiv m^d \bmod N$ and $D \equiv g^d h^{d'} \bmod N$.
2. A generates $r \in_R (-aeN, aeN]$ and $r' \in (-beN, beN]$, computes $V = m^r \bmod N$ and $W = g^r h^{r'} \bmod N$, and sends V, W to B.
3. B generates $c \in_R Z_e$ and sends c to A.
4. A computes $z = cd + r$ and $z' = cd' + r'$, and sends z, z' to B.
5. B checks that $m^z \equiv E^c V \bmod N$ and $g^z h^{z'} = D^c W \bmod N$.

In all steps, A and B also check that the values received are in the appropriate ranges.

The above is a POK of $\Delta \in Z_e$, $\delta \in (-2ae(N+1), 2ae(N+1)]$, and $\delta' \in (-2be(N+1), 2be(N+1)]$ in which $m^\delta \equiv E^\Delta \bmod N$ and $g^\delta h^{\delta'} = D^\Delta \bmod N$. The knowledge error is $1/e$, and the protocol is honest-verifier statistical zero-knowledge, with a statistical difference between views produced by the simulator and those in the real protocol bounded by $2/N$.

A.2 Security Proof Summary

We reduce the security of RSA to the security of our Proactive RSA protocol. Say there exists an adversary, after watching polynomially many messages signed and polynomially many update protocols run, can sign a new challenge message with non-negligible probability ρ. Then we will give a polynomial-time algorithm to break RSA with probability close to ρ. We run the adversary against a simulation of the protocol, and then present m^* to be signed. We will show that the probability that an adversary can distinguish the simulation from the real protocol is negligible, and thus the probability that it signs m^* is negligibly less than ρ.

A major tool that enables us to claim simulatability of secret-key function applications is the notion of a "faking server." The simulator exploits the "actions" of this server to assure that the view is simulatable while not knowing the secret key. This server can be chosen at random and its public actions are

[8] Recall that this main protocol is combined with a Σ-protocol proving knowledge of a commitment generated in a setup protocol, using an "OR" construction.

indistinguishable from an honest server to the adversary. We have to backtrack the simulation **only if** the adversary corrupts this special server (which is a polynomial probability implying expected poly-time simulation).