A shorter version of this paper appears in *Proc. 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, Boston (MA), USA.

An Asynchronous Protocol for Distributed Computation of RSA Inverses and its Applications

Christian Cachin

IBM Research
Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
cca@zurich.ibm.com

April 25, 2003

Abstract

This paper presents an efficient asynchronous protocol to compute RSA inverses with respect to a public RSA modulus N whose factorization is secret and shared among a group of parties. Given two numbers x and e, the protocol computes y such that $y^e \equiv x \pmod{N}$. A synchronous protocol for this task has been presented by Catalano, Gennaro, and Halevi (Eurocrypt 2000), but the standard approach for turning this into an asynchronous protocol would require a Byzantine-agreement subprotocol. Our protocol adopts their approach, but exploits a feature of the problem in order to *avoid* the use of a Byzantine agreement primitive. Hence, it leads to efficient asynchronous protocols for threshold signatures and for Byzantine agreement based on the strong RSA assumption, without the use of random oracles.

1 Introduction

RSA [28] is the most widely used public-key cryptosystem today. Methods for sharing an RSA key among a group of parties in a distributed system, and for using the key in a fault-tolerant way have therefore received considerable attention [14, 15, 7, 18]. They are the subject of *threshold cryptography* [13].

For example, it is well-known how to distribute an RSA signature scheme among $n > 2t$ parties in a synchronous network such that a majority of them can securely issue signatures together, despite the fact that up to t may be faulty and misbehave in arbitrary, malicious ways. Given an RSA public key (N, e), where N is the product of two large primes, such schemes work by sharing the RSA "decryption exponent" $d = e^{-1} \mod \varphi(N)$ among the parties, where $\varphi(\cdot)$ is the Euler function. To sign m, the parties jointly compute σ such that $\sigma^e \equiv m \pmod{N}$. The values N, e, and d are chosen when the signature scheme is set up and remain unchanged afterwards.

Several RSA-based cryptosystems have been proposed [17, 12] recently, where e is given dynamically together with a value x, and the problem is to compute y such that $y^e \equiv x \pmod{N}$. We call this the *RSA inversion problem*. Catalano, Gennaro, and Halevi [9] present a protocol that solves it in the threshold setting. Their protocol computes d as the modular inverse of e with the shared modulus $\varphi(N)$, from which $y = x^d \mod N$ is easily obtained.

Most threshold cryptographic protocols, including those mentioned so far, assume a synchronous network with a broadcast channel connecting all parties. Although this assumption is justified in principle by the existence of suitable clock synchronization and Byzantine agreement protocols that provide broadcast, the approach may lead to rather expensive solutions in practice, for example, when deployed

in a distributed system over a wide-area network with only loosely synchronized clocks. Such systems are also vulnerable to timing attacks.

In this paper, we consider the problem of computing an RSA inverse in an *asynchronous* distributed system, consisting of n communicating parties linked only by point-to-point channels, where local clocks are not synchronized and no a priori bound on message delay exists.

We present a protocol for asynchronous distributed RSA inversion that is quite practical, achieves resilience $n > 4t$, and uses $O(n^3)$ messages and $O(n^4(\kappa + \mu))$ communication, where κ is a security parameter and μ the length of the RSA modulus. Moreover, it is deterministic in the sense that it does not rely on a randomized Byzantine agreement primitive; a randomized solution would not only be more expensive but also preclude one of its applications: to *implement* randomized asynchronous Byzantine agreement using cryptography.

The protocol employs one distributed multiplication step like the synchronous RSA inversion protocol of Catalano et al. [9]. In contrast to their protocol and to the generic approach of turning synchronous protocols with broadcast into asynchronous ones [2, 3], however, our protocol does not rely on *Byzantine agreement* for implementing the distributed multiplication step; this works because RSA inversion is *self-verifiable*, i.e., the result can be checked by every participant locally. This surprising observation is one of the main contributions of this paper, and it leads to a very simple and attractive protocol.

The difficulty with the multiplication step is that all parties must use a consistent combination of sharings, excluding, for example, sharings created by apparently faulty parties. This is ensured by broadcast and by Byzantine agreement in the synchronous protocol and also in a generic asynchronous protocol. Instead, we carry out the operation $t + 1$ times in parallel, each time with a different leader who ensures consistency. Since at least one of these leaders is guaranteed to be correct, at least one operation terminates. As soon as a party terminates the first parallel protocol instance with the correct result, it sends the result to all other parties and aborts the remaining protocols. This works only because the RSA operation is deterministic and because the result can be checked locally by every party.

We discuss two applications of our protocol: *threshold RSA signatures* for asynchronous networks and an efficient *common coin* protocol for implementing asynchronous Byzantine agreement, based on the strong RSA assumption. An important feature of the resulting protocols in both cases is that they do *not* use the so-called *random oracle model* for their security analysis. The random oracle model allows to design practical cryptosystems, but only yields heuristic evidence for their security.

Almost all known *threshold RSA* schemes require a synchronous network with broadcast; the only exception is the non-interactive signature scheme of Shoup [29], which makes crucial use of the random oracle model, however. Using our protocol, we obtain asynchronous threshold implementations of the RSA signature schemes by Gennaro, Halevi, and Rabin [17] and by Cramer and Shoup [12], which are based on the strong RSA assumption. They represent the first implementations of threshold signatures in asynchronous networks without random oracles.

Asynchronous Byzantine agreement protocols rely on randomization, which can be implemented by so-called *common coin* protocols [10]. In modern cryptography, such a common coin is known as an unpredictable threshold pseudo-random function. So far, all efficient implementations of this primitive have relied on the random oracle model or on synchronous networks and broadcast channels. We observe that our protocol can be combined with the verifiable pseudo-random function of Micali, Rabin, and Vadhan [21] to yield an efficient threshold pseudo-random function for asynchronous networks and a randomized Byzantine agreement protocol based on the strong RSA assumption. This is the first asynchronous network implementation of such a function in the standard model, and it gives the most efficient cryptographic asynchronous Byzantine agreement protocol without random oracles.

Related work. This work builds on several cryptographic techniques developed over the last years: the "non-interactive" verifiable secret sharing scheme of Pedersen [24] (which actually rely on synchronous clocks and a broadcast channel and therefore involve interaction), the method of sharing the RSA function by using secret sharing over the integers [14, 15, 26, 18], the zero-knowledge proofs of modular

relations under the strong RSA assumption [16], and the synchronous RSA inversion protocol [9].

Of course, RSA inverses may also be computed in an asynchronous network using the *generic* approach of secure multi-party computation [2, 3]. But such methods are prohibitively expensive for practical use and invoke many instances of asynchronous Byzantine agreement sub-protocols.

A threshold pseudo-random function without random oracles may also be obtained under the decisional Diffie-Hellman assumption from the constructions of Naor and Reingold [22] and Nielsen [23]. The function can be evaluated by an asynchronous protocol, but it requires a number of rounds that is directly proportional to the security parameter.

Outline. The paper is organized as follows. Section 2 presents the system model and the cryptographic assumptions used. It also contains the definition of the RSA inversion problem. A variation of asynchronous verifiable secret sharing is introduced in Section 3; this protocol is used by the preliminary inversion protocol presented in Section 4. The complete inversion protocol is given in Section 5, and the applications to threshold signatures and verifiable random functions are described in Section 6.

2 System Model and Problem Statement

We adopt the basic system model from [6, 5], which describes an asynchronous network of parties with a computationally bounded adversary.

The computational model is parameterized by a security parameter κ; a function $\epsilon(\kappa)$ is called *negligible* if for all $c > 0$ there exists a κ_0 such that $\epsilon(\kappa) < \frac{1}{\kappa^c}$ for all $\kappa > \kappa_0$.

We say that two probability distributions P_X and P_Y are *statistically indistinguishable* if their distance $d(P_X, P_Y) = \frac{1}{2}\sum_x \left|P_X(x) - P_Y(x)\right|$ is negligible. The distance of two random variables is defined as the distance between the associated probability distributions.

The notation $a \leftarrow_R \mathcal{S}$ denotes the (uniformly) random choice of an element a from a set $\mathcal{S}$, and $[\cdot, \cdot]$ denotes an interval of $\mathbb{Z}$.

Cryptographic assumptions. An *RSA modulus* N is the product of two primes of equal length. W.l.o.g. the length of N is $\mu \geq \kappa$ bits. A *safe* prime P is a prime such that $\frac{P-1}{2}$ is prime. A *safe RSA modulus* N is the product of two safe primes P and Q, which comprise the secret key. The *RSA operation* is to compute $m^e \mod N$ for given m and e. The *RSA inversion operation* is to compute y such that $y^e = x \mod N$ for given x and e; y is called the *RSA inverse* of x and e.

The *RSA assumption* is that for a given $e > 1$ a randomly generated RSA modulus N, any probabilistic polynomial-time algorithm computes the RSA inverse of e and a random $x \in \mathbb{Z}_N$ at most with negligible probability.

The *flexible RSA problem* is for a randomly generated RSA modulus N and a random $x \in \mathbb{Z}_N$ to find $e > 1$ and $y \in \mathbb{Z}_N^*$ such that $y^e \equiv x \pmod N$. The *strong RSA assumption* is that any probabilistic polynomial-time algorithm solves the flexible RSA problem at most with negligible probability. Solving the flexible RSA problem is potentially easier than computing RSA inverses because e may depend on the choice of y. The strong RSA assumption has first been used by Barić and Pfitzmann [1] and by Fujisaki and Okamoto [16].

Given the secret key[1] $\varphi(N) = (P-1)(Q-1)$, RSA inversion is easily carried out by computing $d = e^{-1} \mod \varphi(N)$ and then raising x to the power d modulo N. We will show how to compute this in an asynchronous distributed system where $\varphi(N)$ is shared among a group of parties.

2.1 System Model

Network. The network consists of n parties $P_1, \ldots, P_n$, which are probabilistic interactive Turing machines (PITM) [20] that run in polynomial time (in κ). There is an adversary, which is a PITM that

[1] Knowing $\varphi(N)$ is equivalent to the factors of N.

runs in polynomial time in κ. Some parties are controlled by the adversary and called *corrupted*; the remaining parties are called *honest*. An adversary that corrupts at most t parties is called *t-limited*. There is also an initialization algorithm, which is run by a trusted party before the system starts. On input κ, n, t, and further parameters, it generates the state information used to initialize the parties. The initial state information for the corrupted parties is given to the adversary. In other words, the adversary is static, which simplifies the protocol presentation.

We assume that every pair of honest parties is linked by a *secure asynchronous channel* that provides privacy and authenticity with scheduling determined by the adversary. Formally, we model such a network as follows. All communication is driven by the adversary. There exists a global set of messages $\mathcal{M}$, whose elements are identified by a *label* (s, r, l) denoting the sender s, the receiver r, and the length l of the message. The adversary sees the labels of all messages in $\mathcal{M}$, but not their contents. The adversary may also add messages originating from corrupted senders. $\mathcal{M}$ is initially empty. The system proceeds in steps. At each step, the adversary performs some computation, chooses an honest party P_i, and selects some message $m \in \mathcal{M}$ with label (s, i, l). P_i is then *activated* with m on its communication input tape. When activated, P_i reads the contents of its communication input tape, performs some computation, and generates one or more response messages, which it writes to its communication output tape. A response message m may contain a destination address, which is the index j of a party. Such an m is added to $\mathcal{M}$ with label $(i, j, |m|)$ if P_j is honest; if P_j is corrupted, m is given to the adversary. In any case, control returns to the adversary. This step is repeated arbitrarily often until the adversary halts.

These steps define a sequence of events, which we view as logical time. We sometimes use the phrase "at a certain point in time" to refer to an event like this.

The *view* of a party consists of its initialization data, all messages it has received, and the random choices it made so far.

Termination. We define *termination* of a protocol instance only to the extent that the adversary chooses to deliver messages among the honest parties [6]. Technically, termination of a protocol follows from a bound on the number of messages that honest parties generate on behalf of a protocol, which must be independent of the adversary.

We say that a message is *associated* to a particular protocol instance if it was generated by an honest party on behalf of the protocol.

The *message* and *communication complexities* of a protocol are defined as the number and as the bit length, respectively, of all associated messages, generated by honest parties. They are random variables that depend on the adversary and on κ.

Recall that the adversary runs in time polynomial in κ. We assume that the parameter n is bounded by a fixed polynomial in κ, independent of the adversary, and that the same holds for all messages in the protocol (longer messages are simply ignored).

For a particular protocol, a *protocol statistic* X is a family of real-valued, non-negative random variables $\{X_A(\kappa)\}$, parameterized by adversary A and security parameter κ, where each $X_A(\kappa)$ is a random variable induced by running the system with A. (Message complexity is an example of such a statistic.) We restrict ourselves to protocol statistics that are bounded by a polynomial in the adversary's running time.

We say that a protocol statistic X is *uniformly bounded* if there exists a fixed polynomial $p(\kappa)$ such that for all adversaries A, there is a negligible function ϵ_A, such that for all $\kappa \geq 0$,

$$\Pr[X_A(\kappa) > p(\kappa)] \leq \epsilon_A(\kappa).$$

A protocol statistic X is called *probabilistically uniformly bounded* if there exists a fixed polynomial $p(\kappa)$ and a fixed negligible function δ such that for all adversaries A, there is a negligible function ϵ_A, such that for all $l \geq 0$ and $\kappa \geq 0$,

$$\Pr[X_A(\kappa) > lp(\kappa)] \leq \delta(l) + \epsilon_A(\kappa).$$

If X is probabilistically uniformly bounded by p, then for all adversaries A, we have $\mathrm{E}[X_A(\kappa)] = O(p(\kappa))$, with a hidden constant that is independent of A. Additionally, if Y is probabilistically uniformly bounded by q, then $X \cdot Y$ is probabilistically uniformly bounded by $p \cdot q$, and $X + Y$ is probabilistically uniformly bounded by $p + q$. Thus, (probabilistically) uniformly bounded statistics are closed under polynomial composition, which is their main benefit for analyzing the composition of randomized protocols [5].

Protocol execution and notation. We now introduce our notation for writing asynchronous protocols. Recall that a party is always activated with an input message; this message is added to an internal input buffer upon activation.

In our model, protocols are invoked by the adversary. Every protocol *instance* is identified by a unique string ID, also called the *tag*, which is chosen by the adversary when it invokes the instance.

There may be several protocol instances executing at a given party, but no more than one is active concurrently. When a party is activated, all instances are in *wait states*. A wait state specifies a condition defined on the received messages contained in the input buffer and other local state variables. If one or more instances are in a wait state whose condition is satisfied, one such instance is scheduled arbitrarily, and this instance runs until it reaches another wait state. This process continues until no more instances are in a wait state whose condition is satisfied. Then, the activation of the party is terminated, and control returns to the adversary.

There are two types of messages that protocols process and generate: The first type contains *input actions*, which represent a local activation and carry input to a protocol, and *output actions*, which signal termination and potentially carry output of a protocol; such messages are called *local events*. The second message type is an ordinary point-to-point network message, which is to be delivered to the peer protocol instance running on another party; such messages are also called *protocol messages*.

All messages are denoted by a tuple $(ID, \ldots)$; the tag ID denotes the protocol instance to which this message is *associated*. Input actions are of the form $(ID, \texttt{in}, \textit{type}, \ldots)$, and output actions are of the form $(ID, \texttt{out}, \textit{type}, \ldots)$, with *type* defined by the protocol specification. All other messages of the form $(ID, \textit{type}, \ldots)$ are protocol messages, where *type* is defined by the protocol implementation.

We describe protocols in a modular way: A protocol instance may invoke another protocol instance by sending it a suitable input action and obtain its output via an output action of the sub-protocol. This is realized by a party-internal mechanism, which, for any message generated by the calling protocol that contains an input action for a sub-protocol, creates the corresponding protocol instance (if not already running) and delivers the input action; furthermore, it passes all output actions of the sub-protocol to the calling protocol by adding them to the input buffer.

The pseudo-code notation used for describing our protocols is as follows. To enter a wait state, an instance may execute a command of the form **wait for** *condition*, where *condition* is an ordinary predicate on the input buffer and other state variables. Upon executing this command, an instance enters a wait state with the given *condition*.

We specify a *condition* in the form of *receiving messages* or *events*. In this case, *messages* describes a set of one or more protocol messages and *events* describes a set of local events (e.g., outputs from a sub-protocol) satisfying a certain predicate, possibly involving other state variables. Upon executing this command, an instance enters a wait state, waiting for the arrival of messages satisfying the given predicate.

There is a global implicit **wait for** statement that every protocol instance repeatedly executes; it matches any of the *conditions* given in the clauses of the form **upon** *condition block*. Every time a *condition* is satisfied, the corresponding *block* is executed. If there is more than one satisfied *condition*, all corresponding *blocks* are executed in an arbitrary order.

2.2 Computing RSA Inverses with a Shared Secret Key

A (n, k)-*sharing* of a secret s is an encoding of s into a set of *shares* $S_1, \ldots, S_n$ such that any set of at least k shares uniquely defines s and any set of less than k shares does not give information about s. Associated with a sharing is an efficient algorithm reconstruct that reconstructs s from any set of k shares.

A sharing is called *(non-interactively) verifiable* if there exists public information V and an algorithm verify-share such that P_i can determine if a value S represents a "valid" share with respect to V by computing verify-share(V, S). More precisely, a sharing is verifiable if for any adversary that generates a string V and a set of strings $\mathcal{S}^*$ that satisfy the predicate verify-share(V, S^*) for all $S^* \in \mathcal{S}^*$, for any two subsets $\mathcal{S}_0^*, \mathcal{S}_1^* \subset \mathcal{S}^*$ of cardinality k each, the probability that reconstruct$(\mathcal{S}_0^*) \neq$ reconstruct$(\mathcal{S}_1^*)$ is negligible. Given V, the value returned by reconstruct with all but negligible probability is called the *secret associated to* V.

Let N be an RSA modulus. Suppose $P_1, \ldots, P_n$ hold the shares of an (n, k)-sharing of the corresponding RSA secret key $\varphi(N)$. A protocol for *RSA inversion* of x and e with tag ID, for some $e > n$, is started when a party is activated on $(ID, \texttt{in}, \texttt{start}, e, N, x, S_i)$. A party terminates the protocol by generating an output of the form $(ID, \texttt{out}, \texttt{inverse}, y)$. All honest parties must be activated like this and all should output y such that $y^e \equiv x \pmod N$. Recall that our formal system model defines the notion the *associated* messages for every instance; they include all messages with tag ID or a tag starting with ID generated by honest parties analogous to [5].

The formal definition is divided into liveness, correctness, privacy, and efficiency.

Definition 1. A protocol for *RSA inversion over N with a shared secret key $\varphi(N)$* as described above satisfies the following conditions for any t-limited adversary:

Liveness: If all honest parties start the protocol and all associated messages are delivered, then all honest parties terminate except with negligible probability.

Correctness: If an honest party terminates the protocol and outputs y, then $y^e \equiv x \pmod N$ except with negligible probability.

Privacy: The adversary gains no useful information about $\varphi(N)$.

Efficiency: For every ID, the number messages associated to ID is uniformly bounded.

The formalization of *privacy* calls for the existence of a simulator that interacts with the adversary and produces a view that is indistinguishable from a real protocol execution.

Termination of a protocol in our computational model follows analogously to [5] from the combination of *liveness* and *efficiency*.

A protocol in this model is also called *robust* because it tolerates an *active* adversary, i.e., one that exhibits Byzantine faults. It is sometimes useful to restrict the adversary to *passive* corruptions, which means that corrupted parties follow the protocol, but the adversary observes their internal state and obtains information that may lead to a violation of privacy.

3 Verifiable Secret Sharing

Verifiable secret sharing (VSS) is an important primitive in distributed cryptography [11]. We introduce the notion of weak asynchronous verifiable secret sharing and give a protocol for verifiably sharing a secret over the integers.

3.1 Weak Asynchronous Verifiable Secret Sharing

In *weak asynchronous verifiable secret sharing*, the *agreement* property of standard asynchronous verifiable secret sharing (AVSS) [8, 4] is relaxed as follows. When the dealer is faulty, some honest parties may terminate a weak AVSS protocol and others may not, but those who terminate hold consistent shares and are guaranteed that there are enough honest parties holding shares in order to reconstruct the secret. In contrast, AVSS guarantees that either all honest parties terminate the protocol successfully or none, which ensures agreement on the success of the sharing. This difference is analogous to the difference between *consistent broadcast* and *reliable broadcast* in asynchronous networks, using the terminology of Cachin et al. [5].

We consider *dual-threshold sharings*, which generalize the standard notion of secret sharing by allowing the reconstruction threshold to exceed the number of corrupted parties by more than one. In an (n, k, t) dual-threshold sharing, there are n parties holding shares of a secret, of which up to t may be corrupted by an adversary, and any group of k or more honest parties may reconstruct the secret ($n - t \geq k > t$). Such dual-threshold sharings are useful for distributed computation and agreement problems [6].

An AVSS protocol establishes a sharing of a secret $s \in [0, M-1]$ with tag $ID.d$ as follows. Every party is initialized on the protocol, and a special party P_d, called the *dealer*, is activated on $(ID.d, \texttt{in}, \texttt{share}, s)$. When this occurs, we say P_d *shares* $s \in [0, M-1]$ *with tag* $ID.d$. A party is said to *complete the sharing with tag* $ID.d$ when it generates an output of the form $(ID.d, \texttt{out}, \texttt{shared})$. Subsequently, a party *starts the reconstruction of the secret with tag* $ID.d$ when it is activated on $(ID.d, \texttt{in}, \texttt{reconstruct})$; it terminates the reconstruction when it outputs $(ID.d, \texttt{out}, \texttt{reconstructed}, s)$.

More precisely, a *weak asynchronous verifiable dual-threshold secret sharing* protocol establishes the following for any t-limited adversary:

Liveness: If the dealer P_d is honest, all honest parties are initialized on a sharing $ID.d$, and all associated messages are delivered, then all honest parties complete the sharing except with negligible probability.

Correctness: If some honest party completes the sharing, there exists a unique value V such that every honest party who completes the sharing holds a valid share with respect to V except with negligible probability; if the dealer is honest then the secret associated to V is s. Moreover, if at least k honest parties have completed the sharing and start the reconstruction, every one of them reconstructs the secret associated to V except with negligible probability, provided all associated messages are delivered.

Privacy: If the dealer is honest and shares a value s with tag $ID.d$, and less than $k - t$ honest parties have started the reconstruction, then the adversary gains no useful information about s.

Efficiency: For every $ID.d$, the communication complexity is uniformly bounded.

This definition is adapted from [4]. It guarantees the uniqueness of the shared value, but not that k honest parties actually complete the sharing as required for reconstruction; even when an honest party completes the sharing, it may be that only $k - t$ honest parties hold proper shares. To address this problem, we introduce the notion of *semi-weak AVSS*, which is a weak AVSS protocol that guarantees additionally:

Weak Agreement: If an honest party completes the sharing, it may send a message to all other honest parties such that at least k honest parties complete the sharing upon receiving this message or have already completed it.

Weak agreement guarantees that a single honest party who completed the sharing can cause k honest parties to complete the sharing, as necessary for reconstruction, simply by sending a suitable message.

We say that this message *completes* the sharing. Such a *completing message* does not contain secret information; the receiver must already hold a correct share but may not be aware of it before receiving the completing message. The *weak agreement* property is related to the concept of "verifiable" (or "transferable") broadcast from [5].

3.2 Secret Sharing over the Integers

Polynomial secret sharing is usually done in a finite field, but it works also over $\mathbb{Z}$, provided that extra randomization is added. This is a well-known technique developed in the context of threshold RSA [14, 15]. Let $L = n!$. To share a secret $s \in [0, M-1]$ over $\mathbb{Z}$ with security parameter $K = 2^{O(\kappa)}$, choose $k-1$ random values $F_1, \ldots, F_{k-1}$ in $[-KL^2M, KL^2M]$ and let $f(z) = L\big(Ls + \sum_{i=1}^{k-1} F_i z^i\big)$. Denote the coefficients of the *sharing polynomial* f by f_0, f_1, ..., f_{k-1}; they are divisible by L and their absolute value is bounded by KL^3M.

The share of P_i is $f(i)$ for $i = 1, \ldots, n$ computed in $\mathbb{Z}$. It is easy to see that these values form a (n,k)-*sharing* because $sL = \sum_{i \in \mathcal{S}} \lambda_i^{\mathcal{S}} f(i)$ for any $\mathcal{S} \subset \{1, \ldots, n\}$ of cardinality k, where $\lambda_i^{\mathcal{S}} = \prod_{j \in \mathcal{S} \setminus \{i\}} \frac{-j}{i-j}$ are the Lagrange interpolation coefficients for $\mathcal{S}$ and position 0. This can be computed in $\mathbb{Z}$ because every $\prod_{j \in \mathcal{S} \setminus \{i\}} (i-j)$ divides $i!(n-i)!$, which divides $n! = L$.

3.3 A Protocol for Weak AVSS over $\mathbb{Z}_N$

Protocol w-AVSS combines the verification method of Pedersen's VSS [24] with secret sharing over the integers and with the method of Fujisaki and Okamoto [16] to achieve robustness based on the strong RSA assumption in $\mathbb{Z}_N$ to an asynchronous VSS protocol.

Let $K = 2^{O(\kappa)}$ be a security parameter, N a safe RSA modulus, g and h two random squares in $\mathbb{Z}_N^*$ and $s \in [0, M-1]$ the secret to share.

The dealer first computes (n,k,t)-sharings of s and of a random $s_0 \in \mathbb{Z}_M$ over the integers, defining two sharing polynomials f and f', respectively. It also computes verification values $C_j = g^{f_j} h^{f'_j} \mod N$ for $j = 0, \ldots, k-1$, where f_j and f'_j denote the coefficients of f and f'.

The communication follows the approach of "echo broadcast" [27] (which is called "consistent broadcast" in [5]) with a non-interactive (n,m,t) dual-threshold signature scheme $\mathcal{S}_1$ [6] for $m = \max\{k, \lceil \frac{n+t+1}{2} \rceil\}$. Recall that such a threshold signature scheme tolerates up to t corrupted parties and requires m valid signature shares for assembling the threshold signature. First, the dealer sends a share of the secret to all parties and every party answers with an $\mathcal{S}_1$-signature share if the share is valid. Then, upon receiving m $\mathcal{S}_1$-shares, the dealer computes the threshold signature and sends it to all parties. Finally, a party accepts the sharing when it has a valid share and receives a valid threshold signature. Reconstruction is straightforward and omitted (cf. [4]).

A detailed description is given in Figure 1. The predicate verify-point(C, i, a, b) checks that the given values a and b correspond to the points $f(i)$ and $f'(i)$, respectively, committed to in $C = [C_0, \ldots, C_{k-1}]$; it is true if and only if $g^a h^b \equiv \prod_{j=0}^{k-1} (C_j)^{i^j} \pmod N$. Note that verify-point is the share validation algorithm of the resulting sharing with public information C. Common inputs to the protocol are N, g, and h. The local output of every party includes its share (s_i, t_i).

The message complexity of Protocol w-AVSS is $O(n)$. Assuming that a threshold signature and a threshold signature share are $O(\kappa)$ bits, the communication complexity is dominated by the send messages, one of which is of length $k\mu + 2(k + \log(KL^2M)) = O(k\mu + \kappa + \log M)$, since $\log K = O(\kappa)$ and $\log L = O(\kappa)$. When $k = \Theta(n)$, the total communication complexity is $O(n^2\mu + n(\kappa + \log M))$.

It can be shown using standard methods [15, 16, 26, 9] that Protocol w-AVSS implements weak AVSS over $\mathbb{Z}_N$ for $n > k + t$ under the strong RSA assumption.

For example, in *correctness*, the uniqueness of the shared secret follows from the strong RSA assumption as follows. Towards a contradiction, suppose an honest P_i terminates the protocol with a valid share $S = (a, b)$ of s and some honest P_j terminates it with a valid share $S' = (a', b')$ of $s' \neq s$. By the properties of a valid share, verify-share(V_s, S) and verify-share$(V_{s'}, S')$ are both true. Recall that

Protocol w-AVSS **for party** P_i **and tag** $ID.d$

upon *initialization*:

$s_i \leftarrow \perp; t_i \leftarrow \perp$
$V \leftarrow \emptyset; w \leftarrow \perp$

upon $(ID.d, \texttt{in}, \texttt{share}, s)$**:** // only P_d

$F_1, \ldots, F_{k-1}, F'_1, \ldots, F'_{k-1} \leftarrow_R [-KL^2M, KL^2M]$
$s_0 \leftarrow_R [0, M-1]$
let $f(z) = L\big(Ls + \sum_{j=1}^{k-1} F_j z^j\big)$
and $f'(z) = L\big(Ls_0 + \sum_{j=1}^{k-1} F'_j z^j\big)$
$C \leftarrow [C_0, \ldots, C_{k-1}]$, where $C_j = g^{f_j} h^{f'_j} \mod N$
for $j = 0, \ldots, k-1$
for $j = 1, \ldots, n$ **do**
send message $(ID.d, \texttt{send}, C, f(j), f'(j))$ to P_j

upon receiving message $(ID.d, \texttt{send}, C, a, b)$ from P_d for the first time:

if $s_i = \perp$ **and** verify-point(C, i, a, b) **then**
$s_i \leftarrow a; t_i \leftarrow b$
compute an $\mathcal{S}_1$-signature share u
on $(ID.d, \texttt{ready}, C)$
send message $(ID.d, \texttt{ready}, C, u)$ to P_d

upon receiving message $(ID.d, \texttt{ready}, C, u_j)$ from P_j for the first time:

if $i = d$ **and** u_j is a valid $\mathcal{S}_1$-signature share
on $(ID.d, \texttt{ready}, C)$ **then**
$V \leftarrow V \cup \{u_j\}$
$r \leftarrow r + 1$
if $r = m$ **then**
assemble the shares in V to an $\mathcal{S}_1$-threshold
signature v
send message $(ID.d, \texttt{final}, C, v)$ to all parties

upon receiving message $(ID.d, \texttt{final}, C, v)$:

if $w = \perp$ **and** v is a valid $\mathcal{S}_1$-signature on $(ID.d,$
$\texttt{ready}, C)$ **and** verify-point(C, i, s_i, t_i) **then**
$w \leftarrow v$
output $(ID.d, \texttt{out}, \texttt{shared})$

Figure 1: Protocol w-AVSS for weak K-random AVSS of a secret $s \in [0, M-1]$ over $\mathbb{Z}_N$.

the public verification information V_s corresponds to the vector C of the protocol. But the consistency property of the underlying "echo broadcast" (cf. [27]) ensures that $V_s = V_{s'}$ and therefore $g^a h^b \equiv g^{a'} h^{b'} \pmod N$. It follows easily that $g^{\frac{a-a'}{b'-b}} \equiv h \pmod N$ with all but negligible probability since N is a safe RSA modulus, contradicting the strong RSA assumption.

The *privacy* of the secret can be shown using a standard simulation argument (cf. Lemma 1 in [15]).

Protocol w-AVSS can be modified to implement *semi-weak AVSS* over the integers by setting

$$m \;=\; \max\Big\{k+t, \Big\lceil \frac{n+t+1}{2} \Big\rceil\Big\}.$$

The *completing* message, i.e., the message that may cause an honest party to complete the sharing, is the `final` message, which is known to all parties who have completed the sharing. The choice of m then ensures that the $\mathcal{S}_1$-signature included in the message is sufficient evidence that at least $m-t \geq k$ honest parties hold a share of the secret. Since $n - t \geq m$, the protocol can only create a sharing polynomial of degree less than $n - 2t$.

Suitable implementations of the non-interactive threshold signature scheme $\mathcal{S}_1$ exist only in the random oracle model so far [29]. To avoid the random oracle model, one may replace $\mathcal{S}_1$ with a vector of ordinary digital signatures, one for each party, and distribute the verification keys during setup. This increases the size of the `final` message and the communication complexity of the protocol by a factor n.

The same protocol allows to share multiple secrets $s, s', \ldots$ *in parallel* with the same dealer in sharings with reconstruction thresholds $k, k', \ldots$. All properties are satisfied if the reconstruction threshold of $\mathcal{S}_1$ is set to $m = \max\{m, m', \ldots\}$. This not only decreases message and computation complexities compared to separate executions of the sharing protocol, but ensures also that every honest party holding a share of s holds also a share of $s', \ldots$. The latter property is important for the inversion protocol in the next section.

4 A Preliminary Protocol

We start with a preliminary protocol for computing an RSA inverse. The protocol requires a correct leader and tolerates crashes and a passive adversary. That is, all parties apart from the leader may crash and collude to gain knowledge about the secret, but otherwise they follow the protocol.

Protocol AINV1 uses the approach of the synchronous protocol for computing modular inverses by Catalano et al. [9], to obtain a sharing of $d = e^{-1} \mod \varphi(N)$. Given the shared d, the result $y = x^d \mod N$ is reconstructed easily.

More precisely, the protocol works as follows. The input of every P_i includes e, N, x, and S_i, where S_i is P_i's share of $\phi = \varphi(N)$ in a $(n, t+1, t)$-sharing over $\mathbb{Z}_N$, using a polynomial $S(z) = L\big(L\phi + \sum_{j=1}^{t} a_j z^j\big)$ for $a_j \leftarrow_R [-KL^2N, KL^2N]$. W.l.o.g., assume $N \geq K \geq L^2$ and $K > e > n$.

The parties first compute a K-random $(n, t+1, t)$-sharing of a random $Q_0 \in [0, KN-1]$, an analogous sharing of a random $R_0 \in [0, K^2N^2 - 1]$, and a K-random $(n, 2t+1, t)$-sharing of $0 \in [0, K^4L^3N^2 - 1]$. These sharings are executed in parallel as mentioned at the end of Section 3.3 and define integer polynomials $Q(z)$ and $R(z)$ of degree t and $H(z)$ of degree $2t$, where $H(0) = 0$. As usual, the shares of P_i are $Q(i)$, $R(i)$, and $H(i)$.

Next, the parties obtain a sharing of $F(0)$ for $F(z) = S(z)Q(z) + eR(z) + H(z)$ by local multiplication and addition of shares only, and collaboratively reconstruct $F(0) = L^4\phi Q_0 + L^2 e R_0$. Every party applies the extended Euclidean algorithm to compute locally a and b such that $aF(0) + be = 1$, which works if $\gcd(F(0), e) = 1$. Then, $d = aL^2R_0 + b$ is the inverse of e modulo ϕ, as is easy to verify. However, d is not reconstructed since this would reveal ϕ and the factorization of N. Instead, P_i computes a share of d as $d_i = aL^2R(i) + b$ using its share of R_0; then it reveals $y_i = x^{d_i} \mod N$. From $t+1$ such y_i, the result $y \equiv x^d \pmod N$ can easily be recovered.

So far the description parallels [9], but the difference is in the computation of the sharings. To create $Q(z)$, every P_i creates a sharing polynomial $Q_i(z)$ with a random constant term using the semi-weak AVSS protocol from the previous section. $Q(z)$ is now the sum of up to n sharing polynomials $Q_i(z)$ with dealer P_i. The problem is that all parties must arrive at the same $Q(z)$. In the synchronous model with broadcast, agreement on faulty parties P_j who did not properly share a polynomial is immediately available. In our asynchronous model, this is not the case, so we use a single party, the *leader*, to choose a set $\mathcal{S}$ of parties whose sharings should be combined; this works only because we assume the leader to be correct. Thus, $Q(z)$ is set to $\sum_{i\in\mathcal{S}} Q_i(z)$, and $R(z)$ and $H(z)$ are computed in the same way.

It will be shown in Section 5 how to remove the assumption of a correct leader.

Since $H(z)$ needs to be of degree $2t$ and Protocol w-AVSS for semi-weak sharing creates only sharings of degree less than $n-2t$, the maximum resilience of Protocol AINV1 is limited to $n > 4t$.

The message complexity of Protocol AINV1 is $O(n^2)$ because every party runs an instance of weak AVSS as a dealer and the weak AVSS protocol creates $O(n)$ messages. The communication complexity of the inversion protocol is dominated by the share messages and by weak AVSS, which contain an integer of length $M = 2t + 5\log K + 5\log L + 2\log N$ bits. With the assumptions $N \gg K \geq L \geq n$, $\log N = O(\mu)$, and $\log K = O(\kappa)$, we have $M = O(\kappa + \mu)$. Since each weak AVSS instance incurs $O(n^2\mu + n(\kappa + \log M))$ communication, the total communication of the inversion protocol is $O(n^3(\kappa + \mu))$.

Theorem 1. *Under the RSA assumption, Protocol AINV1 implements RSA inversion with a shared secret key for $n > 4t$ with a correct leader and a passive adversary.*

The proof of Theorem 1 is based on the following lemma, which strengthens a lemma of Catalano et al. [9].

Lemma 2. *Let X be a random variable with uniform distribution over $[0, n_x - 1]$ and Y a random variable with uniform distribution over $[0, n_y - 1]$ for $n_y \ll n_x$; let $f, e, b \in \mathbb{N}$ be relatively prime to each other and bounded from above by some $m \ll n_y$. Then*

$$d(Xe + Yf, Xe + Yb) \;\leq\; \frac{n_y m}{n_x e} + \frac{1}{2}\left|\frac{m}{f} - \frac{m}{b}\right|.$$

Proof. Recall that

$$2d(Xe + Yf, Xe + Yb) \;=\; \sum_{0\leq x<n_x e+n_y m} \Big|\Pr[Xe + Yf = x] - \Pr[Xe + Yb = x]\Big|. \qquad (1)$$

Split the sum into three parts: two minor parts, for $0 \leq x < n_y m$ and $n_x e \leq x < n_x e + n_y m$, and a major part, for $n_y m \leq x < n_x e$. It is straightforward to verify that the contribution of the first minor part is bounded by

$$\Pr[Xe < n_y m] \;<\; \frac{n_y m}{n_x e}. \qquad (2)$$

Analogously, the contribution of the second minor part is bounded by

$$\Pr[Xe \geq n_x e] \;<\; \frac{n_y m}{n_x e}. \qquad (3)$$

Protocol AINV1 **with correct leader** P_ℓ **for party** P_i **and tag** $ID.\ell$

upon *initialization*:
$F_i \leftarrow \perp$

upon $(ID.\ell, \texttt{in}, \texttt{start}, e, N, x, S_i)$**:** // $S_1, \ldots, S_n$ form an $(n, t+1)$ sharing of $\varphi(N)$ over $\mathbb{Z}$
$q_i \leftarrow_R [0, KN-1]$; $r_i \leftarrow_R [0, K^2N-1]$
share in parallel with semi-weak K-random sharing and tag $ID|\texttt{sh}.i$:
– $q_i \in [0, KN-1]$ with $(n, t+1, t)$-sharing over $\mathbb{Z}_N$ // this defines $Q_i \in \mathbb{Z}[z]$
– $r_i \in [0, K^2N^2-1]$ with $(n, t+1, t)$-sharing over $\mathbb{Z}_N$ // this defines $R_i \in \mathbb{Z}[z]$
– $0 \in [0, K^4L^3N^2-1]$ with $(n, 2t+1, t)$-sharing over $\mathbb{Z}_N$ // this defines $H_i \in \mathbb{Z}[z]$
if $i = \ell$ **then**
wait for the *completion* of the parallel sharings
from a set $\mathcal{S}$ of $t+1$ parties, i.e.,
for some $|\mathcal{S}| = t+1$, all sharings
with tags $ID|\texttt{sh}.j$ for $j \in \mathcal{S}$
denote the received shares by Q_{ji}, R_{ji}, and H_{ji}, respectively
let $\mathcal{M}$ be the collection of completing messages
from the sharings in $\mathcal{S}$
send message $(\texttt{subset}, \mathcal{S}, \mathcal{M})$ to all parties

upon receiving message $(ID.\ell, \texttt{subset}, \mathcal{S}, \mathcal{M})$ from P_ℓ:
if $i \neq \ell$ **then**
deliver the completing messages in $\mathcal{M}$ to the
parallel sharings $ID|\texttt{sh}.j$ for $j \in \mathcal{S}$
wait for the *completion* of the parallel
sharings from the parties in $\mathcal{S}$, i.e., all
sharings with tags $ID|\texttt{sh}.j$ for $j \in \mathcal{S}$
denote the received shares by Q_{ji}, R_{ji},
and H_{ji}, respectively
$F_i \leftarrow S_i \sum_{j \in \mathcal{S}} Q_{ji} + e \sum_{j \in \mathcal{S}} R_{ji} + \sum_{j \in \mathcal{S}} H_{ji}$
send message $(\texttt{share}, F_i)$ to all parties

upon receiving message $(ID.\ell, \texttt{share}, F_j)$ from P_j for the first time **and** $F_i \neq \perp$:
$\mathcal{F} \leftarrow \mathcal{F} \cup \{(j, F_j)\}$
if $|\mathcal{F}| = 2t+1$ **then**
interpolate $F \in \mathbb{Z}[z]$ of degree $2t$ such that
$F(j) = F_j$ for all $(j, F_j) \in \mathcal{F}$
apply the extended Euclidean algorithm to
compute a and b such that $aF(0) + be = 1$
$d_i \leftarrow a \sum_{j \in \mathcal{S}} R_{ji} + b$
$y_i \leftarrow x^{d_i} \mod N$
send message $(\texttt{partial}, y_i)$ to all parties

upon receiving message $(ID.\ell, \texttt{partial}, y_j)$ from P_j for the first time:
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(j, y_j)\}$
if $|\mathcal{Y}| = t+1$ **then**
$y \leftarrow \prod_{j:(j,y_j) \in \mathcal{Y}} {y_j}^{\lambda_j^{\mathcal{Y}}} \mod N$
output $(ID.\ell, \texttt{out}, \texttt{inverse}, y)$

Figure 2: Protocol AINV1 for asynchronous RSA inversion with a shared secret.

Now

$$\begin{aligned}
&\sum_{n_y m \le x < n_x e} \Big| \Pr[Xe + Yf = x] - \Pr[Xe + Yb = x] \Big| \\
&\quad = \sum_{n_y m \le x < n_x e} \left| \sum_{0 \le y < n_y m} \Pr[Xe = x - y, Yf = y] - \Pr[Xe = x - y, Yb = y] \right| \\
&\quad\quad = \sum_{n_y m \le x < n_x e} \left| \sum_{0 \le y < n_y m} \Pr[Xe = x - y] \big(\Pr[Yf = y] - \Pr[Yb = y] \big) \right| \\
&\quad\quad\quad = \sum_{n_y m \le x < n_x e} \left| \sum_{\substack{0 \le y < n_y m \\ y : e \mid (x-y)}} \frac{1}{n_x} \Pr[Yf = y] - \Pr[Yb = y] \right| \\
&\quad\quad\quad\quad = \frac{1}{n_x} \sum_{n_y m \le x < n_x e} \left| \sum_{\substack{0 \le y < n_y m \\ y : e \mid (x-y) \\ f \mid y}} \Pr[Yf = y] - \sum_{\substack{0 \le y < n_y m \\ y : e \mid (x-y) \\ b \mid y}} \Pr[Yb = y] \right| \qquad (4)
\end{aligned}$$

All probabilities in the inner sums are equal to $\frac{1}{n_y}$. There are approximately $\frac{n_y m}{ef}$ terms in the first inner sum and approximately $\frac{n_y m}{eb}$ terms in the second one. Thus, the difference of the sums is approximately

$$\frac{n_y m}{ef} \frac{1}{n_y} - \frac{n_y m}{eb} \frac{1}{n_y} \;=\; \frac{1}{e} \left(\frac{m}{f} - \frac{m}{b} \right). \qquad (5)$$

Since the outer sum in (4) includes no more than $n_x e$ terms, (4) is bounded by $|\frac{m}{f} - \frac{m}{b}|$. Combining this with (1)–(3) gives

$$2d(Xe + Yf, Xe + Yb) \;\le\; 2\frac{n_y m}{n_x e} + \left| \frac{m}{f} - \frac{m}{b} \right|$$

and the lemma follows. □

Proof of Theorem 1. Let $S \in \mathbb{Z}[z]$ denote the polynomial associated with the sharing of $\varphi(N)$, and let Q_i, R_i, H_i be the polynomials used in the protocol. Recall the maximum absolute values of their coefficients and of the constant term:

polynomial	coefficients	constant term
S	KL^3N	L^2N
Q_i	K^2L^3N	KL^2N
R_i	$K^3L^3N^2$	$K^2L^2N^2$
H_i	$K^5L^5N^2$	0

Furthermore, the coefficients of all polynomials are divisible by L and their constant terms by L^2.

Liveness holds because the leader P_ℓ does not crash by assumption, and, therefore, all honest parties receive the message $(\texttt{subset}, \mathcal{S}, \mathcal{M})$ that determines which sharings to combine. Because the three sharings created by the parties in $\mathcal{S}$ are executed in parallel, an honest party completes them together, or not at all. By the *weak agreement* property of the semi-weak AVSS protocol, at least $2t + 1$ honest parties are guaranteed to complete the sharings given in $\mathcal{S}$. These honest parties receive all shares and output a `share` message in the `subset` activation. Observe that $n - 2t \ge 2t + 1$ is necessary for the sharing of H_i by the properties of the modified protocol w-AVSS. Since at least $2t+1$ `share` messages are released, every honest party receives enough of them to compute a `partial` message and to make progress. Applying the same argument again, every honest party receives enough `partial` messages to terminate the protocol.

Correctness follows from the same argument as used by Catalano et al. [9] and the modifications for the asynchronous case mentioned in the protocol description. In particular, e must be large enough such that the probability that the Euclidean algorithm fails is negligible. *Efficiency* is clear from protocol inspection.

To prove *privacy*, we describe a simulator *SIM* that interacts with the adversary. The simulator takes as input e, x, y, and N such that $y^e \equiv x \pmod N$, and may invoke the simulator for the weak AVSS protocol.

We may assume that the adversary corrupts $P_1, \ldots, P_t$ and that *SIM* is given an $(n, t+1, t)$ sharing of N (in the place of $\phi = \varphi(N)$), defining an integer polynomial $\hat{S}(z)$. Its shares are also held by the corrupted parties.

Suppose $y = x^D$ for $D \in [0, \varphi(N)]$; *SIM* chooses t values $D_1, \ldots, D_t$ at random from $[-KL^2N, KL^2N]$ and computes y_i such that there is a polynomial $d(z) = L\bigl(\sum_{i=1}^{t} D_i z^i + LD\bigr)$ of degree t that satisfies $y_i \equiv x^{d(i)} \pmod N$. This may be done with standard techniques (cf. Lemma 2 in [26]).

Now *SIM* executes the `start` activation on behalf of an honest P_i as prescribed by the protocol and invokes the simulator for the weak AVSS protocol. This defines integer polynomials $\hat{Q}_i$, $\hat{R}_i$, and $\hat{H}_i$.

W.l.o.g. assume P_ℓ chooses $\mathcal{S}$ such that P_ℓ is the only honest party in $\mathcal{S}$. This defines also $\hat{Q}^{\mathcal{S}}(z) = \sum_{j\in\mathcal{S}} \hat{Q}_j(z)$, $\hat{R}^{\mathcal{S}}(z) = \sum_{j\in\mathcal{S}} \hat{R}_j(z)$, and $\hat{H}^{\mathcal{S}}(z) = \sum_{j\in\mathcal{S}} \hat{H}_j(z)$.

When executing a `subset` activation for P_i, *SIM* answers with

$$\hat{F}_i^{\mathcal{S}} = \hat{S}(i)\hat{Q}^{\mathcal{S}}(i) + e\hat{R}^{\mathcal{S}}(i) + \hat{H}^{\mathcal{S}}(i).$$

When executing a `share` activation for P_i, *SIM* answers with y_i, and in `partial` activations, *SIM* proceeds as prescribed by the protocol.

We now show that the simulated execution is statistically indistinguishable from the real execution.

First, note that the values $\hat{q}_d$ and $\hat{r}_d$ shared by P_ℓ are statistically indistinguishable from arbitrary values in the respective intervals by the privacy condition of AVSS.

From the `share` messages, the adversary learns the polynomial $\hat{F}^{\mathcal{S}}(z) = \hat{S}^{\mathcal{S}}(z)\hat{Q}^{\mathcal{S}}(z) + e\hat{R}^{\mathcal{S}}(z) + \hat{H}^{\mathcal{S}}(z)$, of which $\hat{S}$ and the additive contributions from $\hat{Q}_d$, $\hat{R}_d$, and $\hat{H}_d$ are unknown.

By comparing the size of the coefficients in $\hat{H}_d(z)$ with $\hat{S}(z)\hat{Q}_d(z)$ and $e\hat{R}_d(z)$ in the table above, observe that the coefficients of $\hat{H}_d(z)$ exceed the magnitude of the others by at least a factor K (this is where we need $K \geq L^2$ and $K > e$). (The same applies to $H_d(z)$ with respect to $S(z)Q(z)$ and $eR(z)$ in the real execution.) It follows using a standard argument that $\hat{F}^{\mathcal{S}}(z)$ is statistically indistinguishable from $\hat{H}_d(z)$, except for the constant term (and likewise for $F^{\mathcal{S}}(z)$).

Thus, the only difference is in the constant terms. They are given by $\hat{F}^{\mathcal{S}}(0) = L^4NU_{KN} + L^2eU_{K^2N^2}$ and $F^{\mathcal{S}}(0) = L^4\phi U_{KN} + L^2eU_{K^2N^2}$, where U_M denotes a random variable with uniform distribution on $[0, M-1]$. After dividing out L^2, we may apply Lemma 2 with $n_x = K^2N^2$, $n_y = KN$, $f = \phi$, $m = b = N$ and obtain

$$d\Bigl(\hat{F}^{\mathcal{S}}(0), F^{\mathcal{S}}(0)\Bigr) \quad \leq \quad \frac{KN}{K^2N^2}\frac{N}{e}\left|\frac{N}{\phi} - 1\right| \quad = \quad \frac{1}{Ke}\left|\frac{P+Q-1}{(P-1)(Q-1)}\right| \quad = \quad O(\sqrt{N}), \quad (6)$$

recalling that $N = PQ$ is an RSA modulus and $\phi = (P-1)(Q-1)$ for the second step. In other words, they are statistically indistinguishable for the adversary and the theorem follows. □

5 The Main Inversion Protocol

This section presents the complete protocol, which is also *robust*, i.e., tolerates Byzantine faults.

Let us first discuss how to make Protocol AINV1 robust, still under the assumption that P_ℓ is honest. Recall from Section 3.3 that the AVSS sub-protocol yields a commitment vector C to its sharing polynomial. Let C_S denote such a commitment to the polynomial used to share ϕ, corresponding to the

share S_i of P_i, and assume C_S is made available initially to every party. Let C_{Qj}, C_{Rj}, and C_{Hj} for $j \in [1, n]$ denote the commitments resulting from the AVSS sub-protocols invoked by Protocol AINV1.

The semi-weak AVSS sub-protocol is already robust and the `subset` message may remain unchanged. However, additional steps are needed in the inversion protocol to prevent honest parties from accepting `share` and `partial` messages with incorrect data. Since these are point-to-point messages, the standard two-party techniques for proving statements about relations modulo a composite N are sufficient [16]; the method works under the strong RSA assumption.

First, for the message $(\texttt{share}, F_i)$, the sender P_i carries out a zero-knowledge proof of knowledge with every receiver that F_i has been computed correctly with respect to C_S, C_{Qj}, C_{Rj}, and C_{Hj} for $j \in \mathcal{S}$. The receiver accepts the message only if the proof is correct. Second, for the message $(\texttt{partial}, y_i)$, the sender P_i carries out a zero-knowledge proof of knowledge with every receiver that y_i has been computed correctly with respect to a, b, and C_{Rj} for $j \in \mathcal{S}$ (the receiver uses its own values a and b computed upon receiving `share` messages). The receiver accepts the message only if the proof is correct.

It remains to show how to cope with a corrupted leader P_ℓ. We employ the well-known method of running the preliminary inversion protocol $t+1$ times in parallel with different leaders, which guarantees that at least one leader is honest and the corresponding protocol terminates. As soon as a party terminates the first parallel protocol instance with the correct result, it sends the result to all other parties, aborts the remaining instances, and halts. Every party who receives the correct result like this also aborts all inversion protocols. Note that every party may verify that a claimed result y is correct by checking that $y^e \equiv x \pmod{N}$. (The idea of stopping early with the correct, self-verifiable result can be traced back to early work on agreement, e.g., [25].)

This approach works only because RSA inversion is deterministic and the result is self-verifiable; thus, all protocol copies output the same result for all parties and every party can check locally that the output is correct if they receive the result from another party. Note that privacy is maintained since all $t + 1$ copies of AINV1 run completely independently of each other.

This idea is realized in Protocol AINV2, which implements robust RSA inversion. The formal description is given in Figure 3, and the proof of the following theorem is now immediate.

Theorem 3. *Under the strong RSA assumption, Protocol AINV2 implements (robust) RSA inversion with a shared secret key for $n > 4t$.*

Since Protocol AINV2 involves running $O(t)$ copies of Protocol AINV1, the message complexity of the robust RSA inversion protocol is $O(tn^2)$, or $O(n^3)$ in the likely case that $t = \Theta(n)$. The communication complexity is $O(tn^3(\kappa + \mu))$ or $O(n^4(\kappa + \mu))$, respectively.

It is an open problem to devise an efficient RSA inversion protocol with optimal resilience $n > 3t$.

6 Applications

6.1 Threshold RSA Signatures

Standard RSA signatures can be proved secure only in the *random oracle model*. Random oracles are a heuristic tool to analyze the security of cryptographic primitives by pretending that a hash function is implemented by a distributed oracle, which answers with a random value for every distinct point on which it is queried. They are used because the cryptosystems in this model are typically more efficient than the corresponding systems in the standard model, where proofs must be based only on intractability assumptions. The scheme of Shoup [29] implements non-interactive threshold RSA signatures in the random oracle model.

Two related RSA signature schemes that avoid the random oracle model have been proposed recently by Gennaro, Halevi, and Rabin [17] (GHR) and by Cramer and Shoup [12] (CS). Both rely on the *strong RSA assumption* and involve the RSA-inversion of an element x with a fresh e for every signature. These

Protocol AINV2 **for party** P_i **and tag** ID

upon $(ID, \texttt{in}, \texttt{start}, e, N, x, S_i)$**:** // $S_1, \ldots, S_n$ form a $(t+1, n)$ sharing of sL over $\mathbb{Z}$

for $j = 1, \ldots, t+1$ **do**

start robust protocol AINV1 with leader j,

inputs e, N, x, S_i and tag $ID|\texttt{sub}.j$

upon $(ID|\texttt{sub}.j, \texttt{out}, \texttt{inverse}, y)$**:**

if $y^e \equiv x \pmod N$ **then**

send message $(ID, \texttt{inverse}, y)$ to all other parties

abort all protocols AINV1 with tags $ID|\texttt{sub}.j$

for $j \in [1, t+1]$

output $(ID, \texttt{out}, \texttt{inverse}, y)$

halt

upon receiving message $(ID, \texttt{inverse}, y)$:

if $y^e \equiv x \pmod N$ **then**

send message $(ID, \texttt{inverse}, y)$ to all other parties

abort all protocols AINV1 with tags $ID|\texttt{sub}.j$

for $j \in [1, t+1]$

output $(ID, \texttt{out}, \texttt{inverse}, y)$

halt

Figure 3: Protocol AINV2 for asynchronous RSA inversion with a shared secret.

schemes can be implemented in a distributed system using threshold cryptography by sharing $\varphi(N)$ and carrying out a distributed RSA inversion protocol, as shown by Catalano et al. [9] for synchronous systems.

In both schemes, the public key contains a safe RSA modulus N, but the rest is slightly different:

- For the GHR scheme, the public key contains also a random $s \in \mathbb{Z}_N^*$. A signature on a message m is generated by computing $e = h(m, r)$, using a randomized hash function h with random input r, and then by computing the RSA inverse σ of s and e modulo N, resulting in the signature (σ, r). A signature (σ, r) on a message m is verified by computing $e = h(m, r)$ and checking that $\sigma^e \equiv s \pmod N$. The hash function must be division-intractable and it must be possible to efficiently compute a value r when given m and e such that $e = h(m, r)$; the latter property can be achieved by embedding a trap-door in the hash function h (for details, see [17]).

- For CS signatures, the public key contains also two random squares s and $x \in \mathbb{Z}_N^*$, and a sufficiently large random prime f. A signature on a message m is obtained by first selecting a random prime e of the same length as f and a random square $z \in \mathbb{Z}_N^*$. Then, using a hash function h, the value $x' = z^f s^{-h(m)} \mod N$ and the RSA inverse y of $xs^{h(x')}$ and e modulo N are computed; the signature is (e, y, z). To verify a signature, one checks that e is prime and that $y^e \equiv xs^{h(x')} \pmod N$ with $x' = z^f s^{-h(m)}$ (for details, see [12]).

Using the asynchronous RSA inversion protocol presented in this paper, we obtain the first implementations of RSA threshold signatures in asynchronous networks as follows.

Suppose there is a distinguished party P_s who serves as a gateway for signature requests from clients and starts the instance of the distributed signature protocol. Party P_s is assumed to be honest and not to crash. Recall that both signature schemes are deterministic apart from the initial choice of a random value (r and e, respectively), and that the only distributed computation is the RSA inversion.

Thus, the protocols for asynchronous RSA threshold signatures proceed as follows. First, the parties compute a random value using the standard approach: Every party shares a random secret using AVSS,

the distinguished party P_s announces a subset $\mathcal{S}$ of $t+1$ parties whose sharings have terminated successfully, all parties together reconstruct the secrets indicated by $\mathcal{S}$, and every party adds the reconstructed secrets. The result is the desired random value, which is needed in both signature schemes. Second, the parties carry out the RSA inversion protocol together. Finally, every party computes the signature and outputs it.

In absence of an honest P_s, we run the sketched protocol $t+1$ times in parallel; this may result in up to $t+1$ different signatures on the same message, which is unlikely to cause problems in most applications, however. Otherwise, a different protocol may be used that executes a multi-valued Byzantine agreement [5] to determine a set of parties who have successfully shared their secrets and the random value for the signature protocol.

6.2 Verifiable Random Functions and Byzantine Agreement

A *verifiable random function (VRF)* is a pseudo-random function that provides a non-interactively verifiable proof for the correctness of its output. A pseudo-random function f_s with a secret seed s maps κ-bit strings to λ-bit strings [19]; its output is computationally indistinguishable from a random function for any polynomial-time observer. Micali, Rabin, and Vadhan [21] formalized the notion of a verifiable random function: given a κ-bit input x, knowledge of the seed s allows to compute $v = f_s(x)$ together with a unique verification value or "proof" π_x. This proof convinces every verifier of the fact that $v = f_s(x)$ with respect to the given public key of the VRF. The difficulty is that π_x must not reveal anything about f_s on inputs different from x.

The VRF construction of [21] is based on the unpredictability of the RSA inversion operation (the construction is too complex to be recalled here, however). In order to obtain a *threshold verifiable random function* in asynchronous networks, only the RSA inversion step has to be distributed; all other operations are deterministic, given the public key and the shared initialization data of the scheme.

Our asynchronous RSA inversion protocol yields the first threshold VRF based on RSA (the strong RSA assumption, to be precise), which is not based on generic multi-party computation methods. The VRF construction executes a sequence of RSA inversions; our asynchronous distributed implementation succeeds without using a Byzantine agreement primitive since every inversion operation is self-verifiable, as is the final VRF output.

An interesting application of this threshold VRF is to implement asynchronous Byzantine agreement by using the VRF as a common coin sub-protocol (cf. [8, 6]). Thus, our inversion protocol yields also an efficient cryptographic asynchronous Byzantine agreement protocol under the strong RSA assumption, and without random oracles. Apart from this, verifiable random functions have many other interesting applications.

References

[1] N. Barić and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees," in *Proc. EUROCRYPT '97*, pp. 480–494, 1997.

[2] M. Ben-Or, R. Canetti, and O. Goldreich, "Asynchronous secure computation," in *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 52–61, 1993.

[3] M. Ben-Or, B. Kelmer, and T. Rabin, "Asynchronous secure computation with optimal resilience," in *Proc. 13th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 183–192, 1994.

[4] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl, "Asynchronous verifiable secret sharing and proactive cryptosystems," in *Proc. 9th ACM Conference on Computer and Communications Security (CCS)*, pp. 88–97, 2002.

[5] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols (extended abstract)," in *Proc. CRYPTO 2001*, pp. 524–541, Springer, 2001.

[6] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography," in *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 123–132, 2000.

[7] R. Canetti, R. Gennaro, A. Herzberg, and D. Naor, "Proactive security: Long-term protection against break-ins," *RSA Laboratories' CryptoBytes*, vol. 3, no. 1, 1997.

[8] R. Canetti and T. Rabin, "Fast asynchronous Byzantine agreement with optimal resilience," in *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 42–51, 1993.

[9] D. Catalano, R. Gennaro, and S. Halevi, "Computing inverses over a shared secret modulus," in *Proc. EUROCRYPT 2000*, pp. 190–206, Springer, 2000.

[10] B. Chor and C. Dwork, "Randomization in Byzantine agreement," in *Randomness and Computation* (S. Micali, ed.), vol. 5 of *Advances in Computing Research*, pp. 443–497, JAI Press, 1989.

[11] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults," in *Proc. 26th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 383–395, 1985.

[12] R. Cramer and V. Shoup, "Signature schemes based on the strong RSA problem," *ACM Transactions on Information and System Security*, vol. 3, no. 3, pp. 161–185, 2000.

[13] Y. Desmedt, "Threshold cryptography," *European Transactions on Telecommunications*, vol. 5, no. 4, pp. 449–457, 1994.

[14] Y. Desmedt and Y. Frankel, "Shared generation of authenticators and signatures," in *Proc. CRYPTO '91*, pp. 457–469, Springer, 1992.

[15] Y. Frankel, P. Gemmell, P. MacKenzie, and M. Yung, "Optimal-resilience proactive public-key cryptosystems," in *Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 384–393, 1997.

[16] E. Fujiksaki and T. Okamoto, "Statistical zero knowledge protocols to prove modular polynomial relations," in *Proc. CRYPTO '97*, pp. 16–30, Springer, 1997.

[17] R. Gennaro, S. Halevi, and T. Rabin, "Secure hash-and-sign signatures without the random oracle," in *Proc. EUROCRYPT '99*, pp. 123–139, Springer, 1999.

[18] R. Gennaro, T. Rabin, S. Jarecki, and H. Krawczyk, "Robust and efficient sharing of RSA functions," *Journal of Cryptology*, vol. 13, pp. 273–300, 2000.

[19] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM*, vol. 33, pp. 792–807, Oct. 1986.

[20] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, pp. 186–208, Feb. 1989.

[21] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 120–130, 1999.

[22] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," in *Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 458–467, 1997.

[23] J. B. Nielsen, "A threshold pseudorandom function construction and its applications," in *Proc. CRYPTO 2002*, pp. 401–416, Springer, 2002.

[24] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Proc. CRYPTO '91*, pp. 129–140, Springer, 1992.

[25] M. O. Rabin, "Randomized Byzantine generals," in *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 403–409, 1983.

[26] T. Rabin, "A simplified approach to threshold and proactive RSA," in *Proc. CRYPTO '98*, pp. 89–104, Springer, 1998.

[27] M. Reiter, "Secure agreement protocols: Reliable and atomic group multicast in Rampart," in *Proc. 2nd ACM Conference on Computer and Communications Security*, pp. 68–80, 1994.

[28] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, Feb. 1978.

[29] V. Shoup, "Practical threshold signatures," in *Proc. EUROCRYPT 2000*, pp. 207–220, Springer, 2000.