A Proactive Threshold RSA Signature Scheme for Asynchronous Networks

Abstract:

To distribute the signing power and make the system more secure and robust, threshold signature is employed. To tolerate a more powerful, mobile adversary, proactive secret sharing should be adopted to enhance the security of threshold signature. Compared to synchronous networks, asynchronous networks better model practical distributed systems, such as Internet or mobile ad hoc networks. The purpose of this paper is to study proactive secret sharing in asynchronous networks.

So far, to our knowledge, two asynchronous proactive secret sharing schemes have been proposed. Despite their original and seminal works, there are still some drawbacks.

In this paper, we present a complete provably secure asynchronous proactive RSA scheme. Our paper has four contributions. Firstly, we present a provably secure asynchronous verifiable secret sharing for RSA schemes. Secondly, we propose an asynchronous threshold RSA signature scheme. Thirdly, we present a provably secure threshold coin-tossing scheme. Fourthly, we propose an asynchronous proactive secret sharing scheme. Finally, combining the proactive secret sharing scheme with the threshold RSA scheme, we achieve a complete provably secure asynchronous proactive RSA scheme.

Keywords: Asynchronous networks, Threshold RSA signature, Provably secure, Asynchronous verifiable secret sharing, Asynchronous proactive secret sharing scheme, Threshold coin-tossing scheme

1 Introduction

The idea of threshold signature is to distribute the power of the signing operation of a single party to a group of *n* parties. Each party has one share of the whole secret key; and no a single party holds the whole secret key. Only a part of parties together can carry out the signing operation successfully. For example, instead only one party, multiple parties jointly act as Certification Authority (CA) to issue certificates. Threshold signature eliminates the use of a single Trusted Third Party (TTP); an adversary who corrupts up to t parties in the whole system lifetime can not break the whole system. However, if a threshold cryptosystem operates over a longer time period, the assumption that an adversary can just corrupted up to t parties may not hold. In such cases, we need to tolerate a more powerful, *mobile adversary* [1]. To describe a mobile adversary, we divide the whole lifetime of the system into different phases. A mobile adversary can move from party to party and eventually corrupt every party in the system during the entire lifetime of the system, but in every phase it can only corrupt up to t parties. To tolerate a mobile adversary, a natural approach is to refresh the shares of parties at the beginning of the each phase. This method relies on the assumption that parties may *erase* data and on a special reboot procedure to remove the adversary from a corrupted party. In this way, the shares in the existing phase are independent of those in the next phase. Thus the shares obtained by the mobile adversary in this phase become useless in the next phase. Such a method is called proactive secret sharing. Proactive secret sharing schemes operate in phases, and can tolerate the corruption of up to t different parties during every phase [2].

Proactive secret sharing is not just a topic of theoretic research, but has great significance in practical applications. The most basic application is that multiple parties jointly act as the role of *online* CA. Since the lifetime of online CA is long, proactive secret sharing is required. As is well known, certificate issuing is the fundamental function and CA is the most basic component of Public Key Infrastructure (PKI). Such an application in PKI is general, not only in the wired Internet network, but in mobile ad hoc network and Peer to Peer (P2P) networks.

Despite its importance, most previous research in proactive secret sharing mainly focus on synchronous networks, meaning bounds on message delivery delays and processor execution speed are known. This assumption may lead to some vulnerability in practice. For example, when deployed in a distributed system over a wide-area network, denial of service attacks, in particular, might delay messages and/or consume processor cycles, hereby invalidating the defining assumption for a synchronous system.

Recently, researchers begin to consider constructing distributed cryptographic schemes in asynchronous networks, in which message delivery delays and processor execution speeds do not have fixed bounds. Canneti performed a lot of original works in secure asynchronous secure computation, asynchronous Byzantine agreement and asynchronous verifiable secret sharing [3]. A long term research project Malicious-and Accidental-Fault Tolerance for Internet Applications (MAFTIA) of the IBM Zurich research lab assumes an asynchronous network model, and researchers there proposed many seminal asynchronous schemes, such as non-interactive threshold signature [4], asynchronous Byzantine agreement [5][6], asynchronous secret sharing and proactive secret sharing scheme [7], asynchronous protocol for distributed computation of RSA inverses[8], etc. Meanwhile, other researchers also do a lot works in this field. In 2000, Castro proposed an asynchronous replication algorithm for the Internet network based on asynchronous Byzantine agreement [9]. In 2001, Zhou proposed and implemented the first asynchronous online certification authority scheme for the Internet, which is modeled as asynchronous networks [10][11]. In addition, Zhou regards mobile ad hoc networks as asynchronous networks too [12] and present some ideas for constructing proactive secret sharing for ad hoc networks.

The purpose of this paper is to study proactive threshold signature scheme in asynchronous networks. A proactive signature scheme should consist of two parts: a threshold signature scheme and a proactive secret sharing scheme for shares in the threshold signature scheme. In this paper, we present a complete provably secure asynchronous proactive threshold RSA signature scheme.

1.1 Related work

Let's have an overview of synchronous proactive threshold RSA signature schemes first. While many synchronous proactive signature schemes for discrete-log schemes were presented and well studied [13][14][15], the work on secure proactive RSA schemes progressed more slowly. The difficulty appeared because the parties need to be able to keep re-sharing the private key *d*, even if no single party is allowed to know the secret modulus $\phi(N)$ (recall that $\phi(N)$ enables computation of the private key *d* from the public key *e*). Firstly, Frankel *et al.* proposed two proactive RSA schemes [13][14]. Then, Rabin proposed a simplified protocol using similar ideas [15]. Recently, Jarecki presented a more efficient proactive RSA scheme [23] based on Rabin's scheme. The above schemes have similarities in two aspects. They all employ secret sharing "in two levels" to make the shares in the top level backed-up in the secondary level. For example, party p_i has a share d_i such that $\sum d_i = d$. Meanwhile, each d_i is a verifiable secret shared among the parties. In addition, all these schemes require some form of additive rather than polynomial secret-sharing. From these synchronous proactive RSA schemes, we know that the additive secret sharing and backing up shares are very important in constructing proactive RSA schemes. We adopt these ideas and use these two techniques in constructing our asynchronous proactive RSA scheme.

In addition to the above proactive RSA scheme, Luo et al also proposed a proactive RSA scheme for ad hoc networks, which built on polynomial secret sharing [16][17][18]. Unfortunately, due to lack of a formal proof, though it has been studied for several years, the scheme has been proved faulty in two recent papers [19][20]. It seems that provable security is necessary and important to proactive secret sharing schemes when proving its security. In addition, how to design a proactive RSA scheme using polynomial secret sharing is still a great challenge.

Now, let's see the related works in asynchronous networks. In 1999, Shoup proposed an efficient, non-interactive, asynchronous threshold RSA signature [4]. However, no asynchronous threshold discrete-log signature scheme is known now.

So far, to our knowledge, only two asynchronous proactive secret sharing schemes are presented. One is for RSA schemes, based on Rabin's RSA signature scheme, proposed by Zhou in 2001 [10]. The other scheme is for discrete-log based schemes, proposed by Cachin *et al.* in 2002 [7]. Despite their originality and seminal ideas, there are still several drawbacks in both schemes. In Zhou's scheme, the formal security proof of this scheme is missing (Zhou does gives a security proof, but he doesn't follow the approach of provable security.). In addition, Zhou avoids the use of Byzantine agreement in his scheme. However, due to lack of Byzantine agreement, Zhou's scheme needs the participation of the administrator in the presence of Byzantine errors (p71-72) [10]. In this case, the administrator acts as a trusted third party to implement the agreement to make sure the correctness of the scheme, which, to some extent, violates the property of the threshold signature and proactive secret sharing. In Cachin *et al.*'s proactive secret sharing schemes [7], one building block is Byzantine agreement, which builds on Shoup's non-interactive threshold RSA scheme and a random-access coin-tossing scheme. However, how to proactivize Shoup's scheme is missing. In addition, although this scheme is for discrete-log signature schemes, no corresponding asynchronous threshold discrete-log signature scheme is known so far. Considering the above both aspects, this scheme is not a complete proactive threshold signature scheme now.

1.2 Our contributions

In this paper, we present a complete provably secure asynchronous proactive RSA scheme. Our paper has four contributions. Firstly, we present a provably secure asynchronous verifiable secret sharing for RSA schemes, which is based on a verifiable additive secret sharing over the integers. Secondly, we propose an asynchronous threshold RSA signature scheme that is based on the above asynchronous RSA scheme and the random oracle model; in this model, one treats a cryptographic hash function as if it were a black box containing a random function. Thirdly, we present a provably secure threshold coin-tossing scheme on the basis of the above threshold RSA scheme. Fourthly, we propose an asynchronous proactive secret sharing based on the threshold RSA scheme and the coin-tossing scheme. Finally, taken together, we achieve a complete provably secure asynchronous proactive threshold RSA scheme.

1.3 Outline

The paper is organized as follows. In section 2, we present the system model and the cryptographic assumptions used. Next we give an overview of our scheme in Section 3. Then an asynchronous verifiable secret sharing scheme, an asynchronous threshold RSA scheme, a threshold coin-tossing scheme and an asynchronous proactive secret sharing scheme are presented in Section 4,5,6,7, respectively. Then, in section 8, we give some discussions on our scheme. In Appendix A, B, C, the proof of security of the asynchronous verifiable secret sharing scheme, the asynchronous RSA scheme and the asynchronous refresh scheme are described, respectively.

2 System model and cryptographic assumptions

We adopt the basic system model from [7], which describes a proactive asynchronous network of parties with a computationally bounded adversary.

The computational model is parameterized by a security parameter *k*; a function $\varepsilon(k)$ is called *negligible* if for all $f > 0$ there exists a k_0 such that $\varepsilon(k) < \frac{1}{k^f}$ for all $k > k_0$.

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

Furthermore, The notation $a \leftarrow_R S$ denotes the uniformly random choice of an element a from a set S, and $[.,.]$ denotes an interval of Z.

2.1 Asynchronous proactive system model

We only give a brief overview here; for the details, refer to [7].

The network consists of n parties $p_1,..., p_n$, which are Probabilistic Interactive Turing Machine (PITM) that runs in polynomial time in *k*. There is an adversary, which is a PITM that runs in polynomial time in *k*. There is also a trusted dealer and an initialization algorithm, which is run by the trusted dealer before the system starts. The trusted dealer generates the initial state for all n parties. The adversary obtains the initial state of the corrupted parties, but obtains no information about the initial state given to the honest parties.

Every pair of parties is linked by a proactive secure asynchronous channel that provides privacy and authenticity. The delivery of messages is scheduled by the adversary. All communication is driven by the adversary. Furthermore, the adversary may modify or insert message as he wishes, herein the network is merely absorbed into the adversary in the formal model.

In proactive asynchronous networks, we divide the entire lifetime of the system into various phases. However, in asynchronous networks, there are no common clock, how to define a common phase. It turns out that a single time signal or *clock tick*, which defines the start of every phase locally, is enough. Every party operates in a sequence of local phases, which are defined with respect to a trivial protocol *timer*. Every honest party continuously runs one instance of this protocol, which starts when the party is initialized. Upon initialization, the protocol sends a timer message called a *clock tick* to itself. Whenever the party receives a clock tick, the party resends the message to itself over the network. The *local phase* of an uncorrupted party p_i is defined as the number of clock ticks that it has received so far. In our formal model, we leave the scheduling of this signal up to the network, i.e., the adversary.

A party in the proactive asynchronous networks has the ability to erase its internal state information; and there is a special reboot procedure to remove the adversary from a corrupted party. If the adversary corrupts a party during some phase τ, we define the corrupted party to remain in local phase τ until it is rebooted and the adversary is removed. We assume that after a reboot, a party is automatically activated on a clock tick and continues to operate in the subsequent phase. Hence every party is honest at the point in time when it enters the next local phase. However, the adversary can cause a party to appear corrupted during multiple subsequent phases (and across the phase changes) by corrupting it again immediately after the phase change. Given only locally defined phases and purely asynchronous scheduling, we assume that secure channels in the proactive model guarantee that messages are delivered in the same local phase in which they are sent, or else, they are invariably lost. Notice in asynchronous networks, things are a little different. In an asynchronous network model, messages sent on a channel can be arbitrarily delayed and each message ever sent is eventually received.

The adversary can control or corrupt parties. Those parties are controlled by the adversary and called corrupted; the remaining parties are called honest. We assume that the adversary corrupt at most *t* parties who are in the same local phase, which is called a t-limited adversary. We assume the adversary is a static adversary; the adversary's choice of who to corrupt is independent of the network traffic and is decided at the beginning of the phase. The adversary gains complete control over the corrupted parties, and obtains the entire view of the corrupted parties. The view of a party consists of its internal state information and publicly known information. Simply speaking, the view of the party is what she "see". Note, since a party has the ability to erase data, the internal state information in the party is only related to the current phase; the state information related to previous phases is erased. The view of the adversary consists of the views of all corrupted parties and publicly known information.

In computational setting, the notion of the termination of an asynchronous scheme or protocol is a little different. Traditionally, we say the termination of a scheme or a protocol to be that all honest parties "eventually" decide (with probability 1), which means infinite runs of a scheme or protocol. However, in the computationally bounded setting, this simply does not work. So in computational setting, the notion of termination is to be defined to the undecided probability of honest parties is negligible in *k*. Note, although the presentation of ours here is a little different that in [5], the basic idea is similar. For more discussion, see [5].

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 *k*.

Since the adversary runs in time polynomial in *k*, the parameter n should be bounded by a fixed polynomial in *k*, and that the same should hold for all messages in the protocol. (In our model, to facilitate our analysis, we assume $n \le k$).

For a particular scheme or protocol, a *scheme or protocol statistic X* is a family of real-valued, non-negative random variables $\{X_A(k)\}$, parameterized by adversary *A* and security parameter *k*, where each $\{X_A(k)\}$ is a random variable induced by running the system with *A*. (Message complexity is an example of such a statistic.) For our discussion, a statistics should be bounded by a polynomial in *k*.

A statistic *X* is called *uniformly bounded* if there exists a fixed polynomial $p(k)$ such that for all adversaries *A*, there is a negligible function ε_A, such that for all $k \ge 0$,

$$\Pr[X_A(k) > p(k)] \le \varepsilon_A(k)$$

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

$$\Pr[X_A(k) > lp(k)] \le \delta(l) + \varepsilon_A(k)$$

2.2 Cryptographic assumptions

The RSA modulus is $N = pq$, where *p* and *q* are two random large primes of equal length (512 bit, say), and $p = 2p'+1$, $q = 2q'+1$, with p', q' themselves prime. Let $M = p'q'$. The public key is $PK = (N, e)$, and the private key is $d \in Z_{\Phi(N)}$. Denote by Q_N the subgroup of squares in Z_N^*. Clearly, Q_N is cyclic of order *M*. Choose $v \in Q_N$ at random, which generates Q_N since this happens with all but negligible probability.

We also assume a trusted dealer who initializes the distributed scheme (picks the RSA key and shares the private key among the parties) before the adversary can corrupt any of the parties.

3 Overview of the scheme

Our scheme mainly builds on the basic of Cachin et al's scheme and Zhou's scheme. We adopt the framework of Cachin's proactive secret sharing and some similar ideas with Zhou's threshold signature method. In Cachin et al's scheme, discrete-log based primitives are often employed. For example, discrete-log based primitives are used to construct threshold coin-tossing scheme and proactive secret sharing. In our scheme, we must replace these discrete-log based primitives with RSA-based primitives. In addition, although we adopt some ideas form Zhou's threshold RSA signature, our asynchronous threshold signature scheme and asynchronous verifiable secret sharing scheme are still different and new.

Our scheme is complex and involves many layers. For better clarity and convenience, we give an overview of the whole scheme here. The whole layered architecture of the whole scheme is depicted in Figure 1. A complete proactive threshold signature scheme should contain two parts: a threshold signature scheme and a proactive secret sharing scheme for the threshold signature scheme.

An asynchronous threshold RSA signature scheme builds on the Bracha's asynchronous broadcast primitive [21] and an asynchronous verifiable secret sharing scheme. So an asynchronous verifiable secret sharing and an asynchronous threshold RSA signature scheme are basic building blocks for our whole scheme. The asynchronous verifiable secret sharing scheme has two-levels, and is based on the additive secret sharing over the integers. The asynchronous verifiable secret sharing scheme back up an additive secret shares at several parties in a redundant way.

The basic idea of the proactive secret sharing scheme is simple. On a high level, it consists of three steps. In the beginning, multiple new asynchronous verifiable secret sharings are generated; then an agreement needed to be reached on a set of sharings chosen to update the current shares. Finally, the chosen sharings are used to refresh the existing shares.

Notice that the asynchronous proactive secret sharing scheme and the asynchronous threshold signature scheme are related. Our proactive secret scheme is just for our asynchronous threshold signature scheme. In addition, not all asynchronous threshold signature schemes can be proactivized.

In our scheme, the validated Byzantine agreement, a variant of standard Byzantine agreement is utilized to reach the agreement mentioned above. The standard notion of a Byzantine agreement implements only a 0 or 1 binary decision in asynchronous networks. A validated Byzantine agreement [6] scheme extends this to arbitrary domains by means of a so-called external validity condition. It is based on a global, polynomial-time computable predicate Q_{ID} known to all parties, which is determined by an external application. Each party may propose a value that perhaps contains validation information. The agreement ensures that the decision value satisfies Q_{ID}, and that it has been proposed by at least one party. In [6], Cachin et al implemented a validated Byzantine agreement scheme, on the basis of the standard Byzantine agreement. So in our scheme, we just adopt their validated Byzantine agreement scheme. However, we replace the Byzantine agreement scheme in Cachin et al's scheme with ours. The reason is as follows.

A validated Byzantine agreement builds on standard Byzantine agreement. In [5], Cachin et al implemented a standard Byzantine agreement scheme, which is used as a building block to construct the validated Byzantine agreement scheme in [6].The Byzantine agreement scheme in [5] are based on the threshold coin scheme and Shoup's non-interactive threshold RSA signature scheme; and threshold coin scheme is implemented based on discrete-log based primitives. However, how to proactivize Shoup's scheme is unknown. Furthermore, we need to proactivize discrete-log based threshold signature, if discrete-log based primitives used. Considering the above two aspects, we use Toueg's Byzantine agreement [22] to replace Cachin et al' scheme. In addition, in Toueg's scheme, common coins are generated by a Rabin dealer, which is ideal and not practical in practice. So we replace Rabin dealer in his scheme with our threshold coin scheme, which is implemented on the basis of our threshold RSA signature scheme.

Taken together, we obtain a complete asynchronous proactive secret sharing scheme and asynchronous proactive RSA scheme. Our whole asynchronous proactive RSA scheme either employ other schemes directly (e.g. validated Byzantine agreement), or construct new similar schemes based on other schemes (e.g. the asynchronous verifiable secret sharing scheme and the asynchronous proactive scheme).

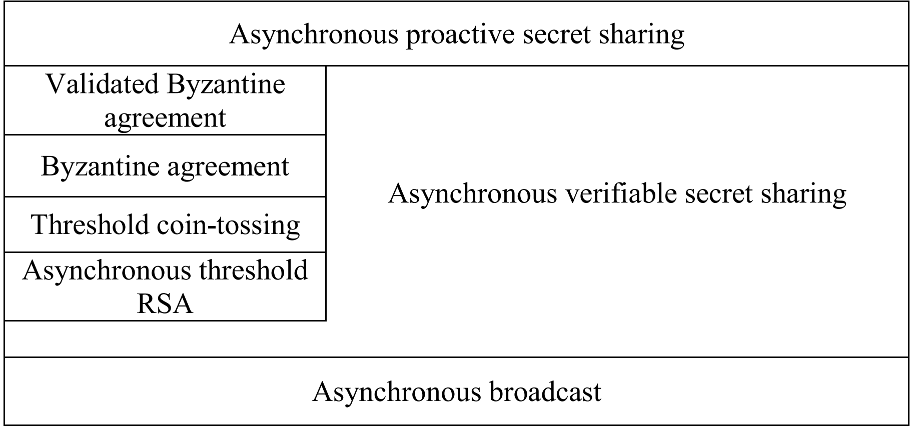
Asynchronous proactive secret sharing
Validated Byzantine agreement
Byzantine agreement
Threshold coin-tossing
Asynchronous threshold RSA
Asynchronous verifiable secret sharing
Asynchronous broadcast

Figure 1 The whole layered architecture of the asynchronous proactive RSA scheme

4 An asynchronous verifiable secret sharing scheme

In this section, we propose an asynchronous verifiable secret sharing (AVSS) scheme, which is employed to build the asynchronous RSA scheme in Section 5.

4.1 Definition of the AVSS scheme

A scheme to share a secret d consists of a *sharing* stage and a *reconstruction* stage, in which the secret is shared or reconstructed, respectively.

The definition of our AVSS is mainly adopted from [7], with some modifications.

Definition 1. A scheme for asynchronous verifiable secret sharing satisfies the following conditions for any *t-limited* adversary:

Liveness: If the adversary initializes all honest parties on a sharing, delivers all associated messages, and the dealer p_d is honest throughout the sharing stage, then all honest parties complete the sharing, except with negligible probability.

Agreement: Provided the adversary initializes all honest parties on a sharing and delivers all associated messages, the following holds: If some honest party completes the sharing, then all honest parties complete the sharing and if all honest parties subsequently start the reconstruction, then every honest party p_i reconstructs some z_i, except with negligible probability.

Correctness: Once $t+1$ honest parties have completed the sharing, there exists a fixed value z such that the following holds except with negligible probability:

1. If the dealer has shared d and is honest throughout the sharing stage, then $d = z$.
2. If an honest party p_i reconstructs z_i, then $z_i = z$.

Privacy: We define privacy using the usual simulation approach. That is, we say that the scheme is private if for the adversary there exists a simulator that runs an execution of the scheme together with the adversary and produces for it a view that is indistinguishable from the real one.
Efficiency: the communication complexity of the scheme is uniformly bounded. Since our scheme runs in computational setting and the adversary is a PTIM, the communication complexity of the scheme should be restricted to uniformly bounded or probabilistically uniformly bounded. Here we use "uniformly bounded"; and in proactive secret sharing, we use "probabilistically uniformly bounded".

4.2 Implementation of the AVASS scheme

Now we describe our asynchronous verifiable secret sharing scheme (asynchronous verifiable additive secret sharing, AVASS) with computational security. The AVASS scheme adopts the similar ideas of Rabin's scheme and Zhou's scheme. There are two levels in the AVASS scheme. The lower level is an additive secret sharing (ASS) over the integers, and the top level is the AVASS scheme. We sketch the whole scheme first.

Let's see the ASS scheme, which is a (l,l) scheme, where $l = \binom{n}{t}$. The shared secret is *d*. The dealer chooses and hands p_i value $d_i \in_R [-lN^2, lN^2]$ for $1 \le i \le l$. Then the dealer sets $d_{public} = d - \sum_{i=1}^{l} d_i$ and computes a commitment array *C* with $C_0 = d_{public}$, $C_i = v^{d_i} \bmod N$ for $1 \le i \le l$ and $C_{l+1} = v^d$.

Next, we construct the AVASS scheme on the basis of the ASS scheme. In order that up to *t* corrupted parties cannot reconstruct the secret, we need to consider all kinds of combination of *t* parties of *n*, which constitutes a set of $\{P_1, ..., P_l\}$, such that each element P_i contains exactly *t* parties. Include secret share d_i in S_p, the share set for a party *p*, if and only if *p* is not in corresponding P_i. That is, for any party *p*, share set S_p equals $\{d_i \mid 1 \le i \le l \wedge p \notin P_i\}$. Notice that, by not assigning d_i to any party in P_i, we ensure that parties in P_i do not together have all *l* shares to reconstruct the secret. Also, for any party *p*, construct an index set $I_p = \{i \mid 1 \le i \le l \wedge p \notin P_i\}$. Clearly, we have $I_p = \{i \mid d_i \in S_p\}$ and $S_p = \{d_i \mid i \in I_p\}$. The index sets provide a sharing-independent description of the share-set construction. Figure 2 illustrates a $(4,2)$ (i.e., *n* = 4 and *t* = 1) AVASS example based on a $\left(\binom{4}{1},\binom{4}{1}\right) = (4,4)$ ASS $\{d_1, d_2, d_3, d_4\}$. The share set for each party p_i consists of all shares except d_i. The index sets are also shown.

party(p)	*Share set* (S_p)	*Index set* (I_p)
p_1	$\{d_2, d_3, d_4\}$	{2,3,4}
p_2	$\{d_1, d_3, d_4\}$	{1,3,4}
p_3	$\{d_1, d_2, d_4\}$	{1,2,4}
p_4	$\{d_1, d_2, d_3\}$	{1,2,3}

Figure 2 An Example of the AVASS scheme.

In the ASS scheme, the secret *d* is shared among *l* shares, and shares are single values. However, in the AVASS scheme, shares are sets of values, called shares sets, which are kept by parties. Shares in $t+1$ share sets implement an additive secret sharing. In the AVASS scheme, there is *l* shares and a size $|S_p|=(l-t)$ of shares for party *p*.

The AVASS scheme uses exactly the same communication pattern as the asynchronous broadcast primitive proposed by Bracha [].There are four steps in the AVASS scheme, and the details of these steps are describes as follows.

1. The dealer computes an (l,l) ASS by choosing a sharing $\{d_1,d_{2,}...,d_l\}$ with $d_{public}=d-\sum_{i=1}^{l}d_i$ for $1\le i\le l$ and $d_i\in_R[-lN^2,lN^2]$. The corresponding witness is *C* such that $C_i=\mathrm{v}^{d_i}\bmod N$. Then the dealer computes the commitment array *C* with $C_0=d_{public}$, $C_i=v^{d_i}\bmod N$ for $1\le i\le l$ and $C_{l+1}=v^d\bmod N$.

Then the dealer sends to every party *p* share set S_p and the commitment array *C*, respectively, in the *send* messages.

2. When they receive the `send` message from the dealer, the parties use verify-share (d_i, *C*) to check if the share d_i is valid, where $d_i\in S_p$. Notice here *C* means the received commitment array. If all shares of S_p are valid, then the parties send the share set in which their share set overlap to each other in an *echo* message. For example, p_i sends an *echo message* containing *C*, share set $S_{p_{i,j}}\in S_{p_i}\bigcap S_{p_j}$ to every party p_j.

3. Upon receiving $\left\lceil\frac{n+t+1}{2}\right\rceil$ `echo` messages that agree on *C* and contain valid shares checked by using verify-share (d_i, *C*), every party computes its share set from the received share sets. (Notice that "agree on *C*" means received *Cs* are the same.) For example, p_j computes its shares set $S_{p_j}=\bigcup_i^k S_{p_{i,j}}$ where $k=\left\lceil\frac{n+t+1}{2}\right\rceil$ and $S_{p_{i,j}}$ is share set sent by p_i. (In case the dealer is honest, the resulting share set is the same as that in the `send` message.) Then p_j sends a *ready* message containing *C*, share set $S_{p_{j,m}}\in S_{p_j}\bigcap S_{p_m}$ to every party p_j. It is also possible that a party receives $\left\lceil\frac{n+t+1}{2}\right\rceil$ valid `ready` messages that agree on *C* and contain valid shares, but has not yet received $\left\lceil\frac{n+t+1}{2}\right\rceil$ valid `echo` messages. In this case, the party computes its share set from the `ready` messages and sends its own `ready` message to all parties as above.

4. Once a party receives a total of $2t+1$ `ready` messages that agree on *C* and contain valid shares, it *completes* the sharing.

The reconstruction stage is straightforward. Every party p_i reveals its share set S_{p_i} to every other party, and waits for $t+1$ such share sets from parties such that for shares contained in these share sets verify-share should hold. Then it computes the secret *d* from these share sets.

In the scheme description, the following predicate is used:

verify-share (d_i, *C*), where d_i is a share and C is the commitment array, verifies that d_i is consistent with *C*; it is true if and only if following three conditions hold: $C_{l+1} = v^{d_{public}} \prod_{j=1}^{l} C_j$ ($C_{l+1} = v^d$), $d_i \in_R [-lN^2, lN^2]$ and $C_i = v^{d_i} \bmod N$.

Theorem 1. In the random oracle model, the AVASS scheme is secure assuming the standard RSA signature scheme is secure for $n > 3t$.

Due to lack of space, the proof of the security of the AVASS scheme appears at appendix A.

5 An asynchronous threshold RSA signature scheme

In this section, we propose an asynchronous threshold RSA scheme and give a formal proof of security. Our asynchronous RSA scheme builds on the AVASS scheme. After the dealer shared the secret key *d* using the AVASS scheme, parties can begin to generate their signature shares.

5.1 Implementation of the asynchronous RSA scheme

Given a message *m*, its signature under the public key (*N*, *e*) is $H(m)^d \bmod N$, where *H* is a hash function. In our setting this signature needs to be generated by the parties in a distributed manner where each individual party uses shares of its share set. As the secret key d is shared using a sum, i.e, $d = d_{public} + \sum_{i=1}^{l} d_i \in Z$, we have that

$$H(m)^{2\times d} = H(m)^{2\times d_{public} + 2\times \sum_{i=1}^{l} d_i} = H(m)^{2\times d_{public}} \prod_{i=1}^{l} H(m)^{2\times d_i} \bmod N .$$

We now describe how a signature share on a message *m* is generated. Let $x = H(m)$. Every p_i has $| S_{p_i} |$ signature shares, and every signature share consists of $x_i = x^{2\times d_i} \in Q_N$, along with a "proof of correctness," where $d_i \in S_{p_i}$. The proof of correctness is basically just a proof that the discrete logarithm of x_i to the base of x^2 is the same as the discrete logarithm of v_i to the base *v*.

Now let's see the details. Let $L(lN^2)$ be the bit-length of lN^2 and H' be a hash function, whose output is an L_1 -bit integer, where L_1 is another security parameter (L_1 =128, say). To construct the proof of correctness, party p_i chooses a random number $r \in_R [0, 2^{L(lN^2)+2L_1} - 1]$, and computes

$v' = v^r, \;\; x' = x^{2\times r}, \;\; H'(v, x^2, v_i, x_i, v', x'), \;\; z = d_i c + r$ The proof of correctness is (*z*, *c*).

To verify this proof of correctness, one checks that $c = H^{'}(v, x, v_i, x_i, v^z v_i^{-c}, x^{2\times z} x_i^{-c})$.

We next describe how signature shares are combined. Suppose we have valid shares from a set *S* of parties, where $S = \{p_1, ..., p_{t+1}\}$, and all *l* shares of the secret are contained in $t+1$ share sets of *S*.

Assume that $x_i = x^{2\times d_i}$ for $(1 \le i \le l)$. Then to combine shares, we compute $y' = x^{2\times d_{public}} \prod_{i=1}^{l} x_i$ such that $y'^{e} = x^{2} \bmod N$. Since *e* is an odd number, $\gcd(e,2) = 1$. Apply the extended Euclidean algorithm on *e* and 2 to compute *a* and *b*, such that $2\times a + e\times b = 1$ .Thus we achieve the signature $y = y'^{a}\, x^{b}$ of the message *m* such that $y^{e} = x \bmod N$.

5.2 Security analysis of the asynchronous RSA scheme

Theorem 2. In the random oracle model for H' , the asynchronous RSA scheme is a secure threshold signature scheme (robust and non-forgeable) assuming the standard RSA signature scheme is secure.

Due to lack of space, the proof is given at appendix B.

6 A threshold coin-tossing scheme

The purpose of the threshold coin-tossing scheme is to replace Rabin dealer in Toueg's Byzantine agreement scheme. Next, modified Toueg's scheme is used to replace the Byzantine agreement scheme in Cachin et al's validated Byzantine agreement. In Section 3, an overview of the relationship between the validated Byzantine agreement and the threshold coin-tossing scheme is given. In this section, we then propose a threshold coin-tossing scheme.

6.1 Definition of threshold coin-tossing scheme

First, we define the notion of a $(n, t+1)$ threshold coin-tossing scheme. The basic idea is that there are *n* parties, up to *t* of which may be corrupted. The parties hold shares of an unpredictable function F mapping the name C (which is an arbitrary bit string) of a coin to its value $F(C) \in \{0,1\}$. The parties may generate shares of a coin—*l* coin shares are both necessary and sufficient to construct the value of the particular coin.

Definition 2. Our definition of a threshold coin-tossing scheme is adopted from [5]. A threshold coin-tossing scheme satisfies the following conditions for any *t-limited* adversary:

Robustness. There is only negligible probability for an adversary to produce a name C and *l* valid shares of C such that the output of the share combining algorithm is not *F(*C*)*.

Unpredictability. An adversary's advantage in the following game is negligible. The adversary interacts with the honest parties as above, and at the end of this interaction, he outputs a name C that has not been submitted as a reveal request, and a bit $b \in \{0,1\}$. The adversary's advantage in this game is defined to be the distance from 1/2 of the probability that $F(C) = b$.

6.2 Implementation of the threshold coin-tossing scheme

For a given coin C, to obtain the value of the coin C, first, compute the threshold RSA signature of name of coin C, suppose the result is g_0 ; then computes $H''(g_0)$ to obtain the value of coin C. Here H'' is a hash function, which could actually be implemented in the standard model by the inner product of the bit representation of the input with a random bit string, chosen once and for all by the dealer in the initial phase.

6.3 Proof of the security of the threshold coin-tossing scheme

Theorem 3. In the random oracle model, the threshold coin-tossing scheme is secure assuming the standard RSA signature scheme is secure.

Clearly, the robustness of the scheme follows from the robustness of the asynchronous RSA scheme.

To prove unpredictability, we assume we have an adversary that can predict a coin with non-negligible probability, and show how to use this adversary to efficiently generate RSA signature. Observe that because the adversary has a non-negligible advantage in predicting the value of the coin C, he must evaluate H'' at the corresponding point g_0 with non-negligible probability, which violates the non-forgeablity of the asynchronous RSA scheme.

7 An asynchronous proactive secret sharing scheme

An asynchronous proactive secret sharing consists of two parts: an AVASS scheme and a refresh scheme to update shares. In Section 4, we described an AVASS scheme. In this Section, we present a proactive secret sharing scheme to update shares in the AVASS scheme. Combining the AVASS scheme and the refresh scheme shown in this section, we achieve an asynchronous proactive secret sharing scheme.

7.1 Definition of asynchronous refresh scheme.

Our definition of an asynchronous secure refresh is mainly adopted from [7], with some modifications.

Definition 3. Suppose the shared secret key is *d*. An *asynchronous refresh* scheme satisfies the following conditions for any *t-limited* adversary:

Liveness: If the adversary activates all honest parties on a clock tick for the beginning of a phase and delivers all associated messages within phases, then all honest parties complete the refresh, except with negligible probability.

Correctness: If at least $t+1$ honest parties have completed the refresh of sharing and have not *detected* a subsequent clock tick for a new phase, these parties can reconstruct the secret key and the reconstructed value equals *d*, except with negligible probability.

Privacy: We define privacy using the usual simulation approach. That is, we say that the protocol is private if for the adversary there exists a simulator that runs any polynomial number of consecutive executions the scheme together with the adversary and produces for it a view that is indistinguishable from the real one.

Efficiency: the communication complexity of the scheme is probabilistically uniformly bounded.

Notice that this definition guarantees that the parties complete the refresh only when the adversary delivers messages within phases. Otherwise, the model allows the adversary to cause the secret to be lost, in order to preserve privacy. Such a trade-off between privacy and correctness seems unavoidable in asynchronous networks (See [7] for more discussion about this topic).

7.2 Implementation of the asynchronous refresh scheme

A party starts to perform the refresh scheme as soon as it detects or receive the next clock tick. From a high-level point of view, the scheme works in three stages. First, every party p_i shares every share $d_i \in S_{p_i}$ using an AVASS scheme. Since every share set has $(l-t)$ shares and there are all *n* parties, there are $(l-t)\times n$ sharings. In order to distinguish these sharings, every sharing is identified by a symbol ID|*j*, $(1 \le j \le (l-t)\times n)$. In these sharings, we call a set of sharings a candidate set if that set consists of exactly one sharing generated from each share of all *l* shares of the secret *d*. Second, for the above sharings, the parties propose a candidate set that have successfully terminated as their input to a validated Byzantine agreement scheme; then use the validated Byzantine agreement to select a candidate set as the output. Third, they compute fresh shares and share sets from the set of sharings which they agreed on.

Notice that, to ensure the correctness of a sharing, parties need to check if the commitment v^s of the shared s is correct. For this purpose, parties have to store the commitments of $v^{d_i} \bmod N$ for $1 \le i \le l$ in an array V, every element of which is the commitment of the corresponding d_i. At the end of every phase, d_i and V are updated. Then in next phase, parties can check if the commitment of $v^{d_i} \bmod N$ in a new phase is correct by comparing it with that stored in V.

Every party executes the following three steps to refresh secret shares in phase τ.

1. Party p_i participate in initializing $(l-t)\times n$ AVASS (n, t + 1)-sharings ID|*j* for $j \in [1,(l-t)n]$ using an extended version of the AVASS scheme. Thus the shares d_i $(1 \le i \le l)$ of the secret key d are re-shared. The AVASS scheme here is a little different from that one in Section 3. Firstly, the range of shares' value is different. For example, shares $d_{i,j}$ and $d_{i,public}$ of d_i $(1 \le i \le l)$ are chosen or computed as follows. $d_{i,public} = d_i - \sum_{k=1}^{l} d_{i,k}$, $d_{i,k} \in_R [-N^2, N^2]$ (not $d_{i,k} \in_R [-lN^2, lN^2]$). Secondly, in the extended AVSS scheme, each party adds a digital signature to every *ready* message. In extended AVASS instance $ID \mid j$, the signature is computed on ($ID \mid j, \tau, \text{ready}$). A list Π_j of 2*t* + 1 such signature is output when the sharing is completed and may serve as a proof for this fact. Thirdly, to keep a sharing correct, parties have to check the validness of the commitment of the shared secret. Finally, party p_i should immediately *erase* the current shared secret in sharing ID|*j*, in which p_i as the dealer. (This is used to preserve privacy. See [6] for more discussion about this topic.)
2. p_i waits for completing a candidate set. Recall that the extended AVASS scheme also returns a proof Π_j for the completion of the sharing. Next, P_i proposes the candidate set for the validated Byzantine agreement. Its proposal is a set $\mathcal{L}_i = \{(j, \Pi_j)\}$ of *l* tuples, indicating the sharing ID|*j* is completed and containing the list Π_j of signatures on *ready* messages from the extended sharing. The predicate of the VBA scheme is set to **verify-termination**$(\tau, \mathcal{L}_i)$, which verifies that $\mathcal{L}_i$ contains *l* sharing with the proofs that these sharings will actually terminate. It is true if and only if $|\mathcal{L}_i| = l$ and for every $(j, \Pi_j) \in \mathcal{L}_i$, the list Π_j contains at least 2*t* + 1 valid signatures on *ready* messages from distinct parties.
3. After p_i decides in the VBA scheme for a set $\mathcal{L}$ that indicates *l* AVASS instances, it waits for these sharings to complete. Then compute its new shares, share set and the new commitments *V*. The new shares for d_i is computed as (1); the new shares for d_{public} is computed as (2).

$$d_i^{new} = \sum_{j=1}^{l} d_{j,i} \tag{1}$$

$$d_{pubic}^{new} = d_{public}^{old} + \sum_{j=1}^{l} d_{j,public} \tag{2}$$

Then, the new commitment for d_i $(1 \le i \le l)$ is computed as (3)

$$v^{d_i^{new}} = \prod_{j=1}^{l} v^{d_{j,i}} \tag{3}$$

Finally, the party *aborts* all sharing ID|j $(1 \le j \le (l-t) \times n)$, which automatically *erases* all information of these sharings.

In practice, there is still another step needed to be considered. After the refresh scheme updates the shares, parties need to update their certificates. That is, honest parties request the certificates for the new phase, and the private keys for the old certificates are erased. Since then, the new certificates are used to sign messages; and the messages signed by old private keys are not accepted. Note that the certificates issued should contain some information, such as the phase information, to distinguish the certificates in different phases.

Theorem 4. In the random oracle model, the asynchronous refresh scheme is secure assuming the standard RSA signature scheme is secure for $n > 3t$.

Due to lack of space, the proof of security of the asynchronous refresh scheme appears at appendix C.

8 Discussion

Combined all the above schemes into a single one, we obtain a complete provably secure proactive threshold RSA signature scheme. Now we consider the performance of our scheme.

Clearly, the message complexity and the communication complexity of our proactive secret sharing scheme are both $O(\binom{n}{t})$. In normal cases, the message complexity and the communication complexity of the asynchronous RSA scheme are also both $O(\binom{n}{t})$ (Since every party have to send $(l-t)$ signature shares for every signature.). However, when parties are honest, we can optimize the asynchronous RSA signature scheme as follows. For a set of $t+1$ honest parties, suppose $p_1, p_2, ..., p_{t+1}$, p_1 generate its signature share for message *m* as

$Sig_{p_1} = m^{\sum d_i}, d_i \in S_{p_1}$, and p_2 generate its signature share for message *m* as

$Sig_{p_2} = m^{\sum d_i}, d_i \in (\overline{S}_{p_1} \bigcap S_{p_2})$, and so on. The final signature is

$Sig(m) = m^{public} \prod_{i=1}^{t+1} Sig_{p_i}$ .Consequently, the message complexity is just $O(t)$.

Although our scheme is the first complete provably secure asynchronous proactive RSA scheme and asynchronous proactive secret sharing, the drawback that the message complexity and the communication complexity of the refresh scheme are both $O(\binom{n}{t})$, renders the whole scheme a little inefficient. Such a drawback makes our scheme only suitable to cases when *t* is small (1 or 2). Although it is not very satisfactory, for most applications, such as online CA, a small *t* is sufficient.

In addition, the message complexity and the communication complexity of our scheme are close to those of Zhou's scheme [10][11]. In his paper, Zhou analyze that such scheme is acceptable *t* is small (1 or 2), and the experiments on the prototype system of their online CA also demonstrated that such scheme is feasible in practical applications.

Acknowledgements

Many thanks go to Stanislaw Jarecki and Christian Cachin for their valuable suggestions.

REFERENCES

[1] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proc. 10th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 51–59, 1991.

[2] 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 (D. Coppersmith, ed. Springer.).* 963:339-352, 1995.

[3] R. Canetti, *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute, 1995.

[4] V. Shoup. Practical threshold signatures. In *Advances in Cryptology: EUROCRYPT 2000* (B. Preneel,ed.), 1087(207-220), 2000.

[5] 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),* pages 123–132, 2000.

[6] Cachin C., Kursawe, K. Petzold F., and Shoup V. Secure and efficient asynchronous br2adcast protocols (extended abstract). In *Advances in Cryptology: CRYPTO 2001*(LNCS). 2139:524-541, August 2001.

[7] Cachin C., Kursawe K., Lysyanskaya A., and Strobl R. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proc. 9th ACM Conference on Computer and Communications Security (CCS).* Washington, DC, USA, pages 88–97, November 2002.

[8] Cachin C. An asynchronous protocol for distributed computation of RSA inverses and its applications. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing. Boston, USA*, , pages 153 – 162, April 2003.

[9] CASTRO, M. 2000. Practical Byzantine fault tolerance. PhD. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Mass.

[10] Zhou L. Towards Fault-tolerant and Secure On-line Services. PhD thesis, Department of Computer Science, Cornell University, Ithaca, NY USA. April 2001.

[11] Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. COCA: A Secure Distributed On-line Certification Authority. ACM Transactions on Computer Systems 20, 4 (November 2002), 329--368. Earlier version: Technical Report TR 2000-1828, December 7, 2000.

[12] L. Zhou and Z. J. Haas. Securing Ad Hoc Networks. IEEE Network Magazine, 13(6):24–30, 1999.

[13] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Optimal-Resilience Proactive Public-Key Cryptosystems. In *Foundations of Computer Science FOCS'97*, pages 384–393, 1997.

[14] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Proactive RSA. In *Proc. of Crypto'97*, pages 440–454, 1997.

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

[16] Jiejun Kong, Petros Zerfos, Haiyun Luo, Songwu Lu, and Lixia Zhang. Providing Robust and Ubiquitous Security Support for MANET. In *IEEE 9th International Conference on Network Protocols (ICNP)*, 2001.

[17] Haiyun Luo, Jiejun Kong, Petros Zerfos, Songwu Lu, and Lixia Zhang. URSA: Ubiquitous and Robust Access Control for Mobile Ad Hoc Networks, available on-line at

http://www.cs.ucla.edu/wing/publication/publication.html. In *IEEE/ACM Transactions on Networking (ToN)*, to appear, Oct 2004.
[18] H. Luo and S. Lu. Ubiquitous and Robust Authentication Services for Ad Hoc Wireless Networks, available on-line at http://citeseer.ist.psu.edu/luo00ubiquitous.html. Technical Report TR-200030, Dept. of Computer Science, UCLA, 2000.
[19] M. Narasimha, G. Tsudik, and J. H. Yi. On the Utility of Distributed Cryptography in P2P and MANETs: The Case of Membership Control. In *IEEE 11th International Conference on Network Protocol (ICNP)*, pages 336–345, November 2003.
[20] Stanislaw Jarecki, Nitesh Saxena, and Jeong Hyun Yi. Cryptanalyzing the Proactive RSA Signature Scheme in the URSA Ad Hoc Network Access Control Protocol. In ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN), to appear, October 2004.
[21] G. Bracha. An asynchronous [(n- 1)/3]-resilient consensus protocol. In *Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 154-162, 1984.
[21]S. Toueg, Randomized Byzantine agreements, In *Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 163–178, 1984.
[23] Stanislaw Jarecki and Nitesh Saxena, Further Simplifications in Proactive RSA Signature Schemes. A draft communicated to the authors by email by Stanislaw Jarecki.

Appendix

A Proof of the security of the AVASS scheme

Lemma 1. Suppose an honest party p_i sends a `ready` message containing C_i and a distinct honest party p_j sends a `ready` message containing C_j. Then $C_i = C_j$.

Proof. We prove the lemma by contradiction. Suppose $C_i \neq C_j$. p_i generates the `ready` message for C_i only if it has received at least $\left\lceil \frac{n+t+1}{2} \right\rceil$ `echo` messages containing C_i or $t+1$ `ready` messages containing C_i. In the second case, at least one honest party has sent a `ready` message containing C_i upon receiving at least $\left\lceil \frac{n+t+1}{2} \right\rceil$ `echo` messages; we may as well assume that this is p_i to simplify the rest of the argument. Thus, p_i has received $\left\lceil \frac{n+t+1}{2} \right\rceil$ `echo` messages containing C_i, of which up to t are from corrupted parties. Using the same argumentation, p_j must have received at least $\left\lceil \frac{n+t+1}{2} \right\rceil$ `echo` messages containing C_j .Then there are at least $2\left\lceil \frac{n+t+1}{2} \right\rceil = n+t+1$ `echo` messages received by p_i and p_j together, among them at least $n+t+1$ from honest parties. But no honest party generates more than one such message by the scheme.

Liveness. If the dealer p_d is honest, it follows directly by inspection of the scheme that all honest parties complete the sharing, provided all parties initialize the sharing and the adversary delivers all associated messages.

Agreement. We first show that if some honest party completes the sharing, then all honest parties complete the sharing, provided all parties initialize the sharing and the adversary delivers all associated messages.

Suppose an honest party has completed the sharing. Then it has received $2t+1$ valid `ready` messages that agree on some $\overline{C}$. Of these messages, at least $t+1$ have been sent by honest parties. A *valid* `echo` or `ready` message is one that satisfies verify-share, and it is easy to see from the definition of verify-share that honest parties send only valid `ready` messages. Since an honest party sends its `ready` message to all parties, every honest party receives at least $t+1$ valid `ready` messages with the same $\overline{C}$ by Lemma 1 and sends a `ready` message containing $\overline{C}$. Hence, by the assumption, any honest party receives $n-t > 2t+1$ valid `ready` messages containing $\overline{C}$ and completes the sharing.

As for the reconstruction part, it follows from Lemma 1 that every honest party p_i computes the same $\overline{C}$. Moreover, p_i has received enough valid `echo` or `ready` messages with respect to $\overline{C}$ so that it computes valid `ready` messages and a *valid* share d_i with respect to $\overline{C}$ such that verify-share($d_i, \overline{C}$) holds. Thus, if all honest parties subsequently start the reconstruction stage, then every party receives enough valid shares to reconstruct some value, provided the adversary delivers all associated messages.

Correctness. Let *J* be the index set of the $t+1$ honest parties p_j that have completed the sharing.

To prove the first part, suppose the dealer has shared d and is honest throughout the sharing stage. Towards a contradiction assume $z \neq d$. Because the dealer is honest, it is easy to see that every `echo` message sent from an honest p_i to p_j contains *C*, $S_{p_{i,j}} \in S_{p_i} \bigcap S_{p_j}$ as the same as sent by the dealer. Furthermore, if the party *p* in *J* computed their share sets only from these `echo` messages, then the resulting S'_p should be the same as S_p sent by the dealer. But since $z \neq d$, at least one honest party p_i computed $S'_{p_i} \neq S_{p_i}$; this must be because p_i accepted an `echo` or `ready` message from some corrupted p_m containing $\overline{C}$ and $S'_{p_m} \neq S_{p_m}$. It is easy to see from Lemma 1 and from the fact that the dealer is honest that *C* used by the dealer and $\overline{C}$ sent by p_m are equal. Since p_i has evaluated verify-share to true for all shares of S'_{p_m}, we have $C_i = v^{d'_i} \bmod N$ for all shares $d'_i \in S'_{p_m}$, where $i \in I_{p_m}$. Thus, $v^{d'_i} \bmod N = v^{d_i} \bmod N$, implies $v^{d'_i - d_i} \bmod N = 1$. Recall that the order of Q_N is *M*, which means $d_i - d'_i = 4k'M, k' \in Z \wedge k' \neq 0$. Consequently, since $d_i \in [-lN^2, lN^2]$, $d'_i \in [-lN^2, lN^2]$ and *N* are known, one can easily compute *M* in polynomial time in *k*, which means the standard RSA scheme is not secure.

To prove the second part, assume that two distinct honest parties p_i and p_j reconstruct values z_i and z_j. This means that they have received two distinct share sets S_i and S_j of $t+1$

shares each, which are valid with respect to the unique commitment array $\overline{C}$ used by P_i and P_j (the uniqueness of $\overline{C}$ follows from Lemma 1). According to the scheme, z_i and z_j are computed from the shares in the share sets obtained from S_i and S_j , respectively. Since the shares in S_i and S_j are valid, it is easy to see that $v^{d_i^{'}} \bmod N = v^{d_i^{''}} \bmod N$ for all shares $d_i^{'} \in S_i$ and $d_i^{''} \in S_j$ ($1 \le i \le l$). The remaining proof is similar to the first part.

Privacy. We show how to simulate the adversary's view. Let $p_1,..., p_t$ be the set of corrupted parties, and assume $\{d_1,..., d_{l-1}\}$ are shares contained in $\{S_{p_1}, S_{p_2},..., S_{p_t}\}$.

Choose secret value $\hat{d} \in_R [0, N-1]$, shares $\hat{d}_1,..., \hat{d}_l \in_R [-lN^2, lN^2]$ and $\hat{d}_{public} = \hat{d} - \sum_{i=1}^{l} \hat{d}_i$ .Then perform the following steps.

(a) compute $\hat{w}_l = v^{\hat{d}_l},..., \hat{w}_{l-1} = v^{d_{l-1}} \bmod N$

(b) set $\hat{w}_l = v^{\hat{d}_l} = v^d / v^{d_{public}} \prod_{i=1}^{l-1} \hat{w}_1 \bmod N$

The statistical distance between the probabilistic distribution of $d_i (1 \le i \le l)$ and that of $\hat{d}_i (1 \le i \le l)$ is 0. Let's see the statistical distance between the probabilistic distribution of d_{pub} and that of $\hat{d}_{public}$. Recall that the adversary corrupts *t* parties, views the internal state of *t* parties and $d_i (1 \le i \le l-1)$, then the probability distribution of $d_{public} = d - \sum_{i=1}^{l-1} d_i - d_l$ is related to d_l and the probability distribution of $\hat{d}_{public} = \hat{d} - \sum_{i=1}^{l-1} \hat{d}_i - \hat{d}_l$ is related to $\hat{d}_l$. Since $d_l, \hat{d}_l$ are randomly selected from the range of $[-lN^2, lN^2]$ and $d - \hat{d} < N$, the statistical distance between probability distribution of $d_{public} = d - \sum_{i=1}^{l-1} d_i - d_l$ and that of $\hat{d}_{public} = \hat{d} - \sum_{i=1}^{l-1} \hat{d}_i - \hat{d}_l$ is $O((lN)^{-1})$. Thus, the above simulation is statistically indistinguishable to the adversary.

Efficiency. Clearly, the communication complexity of the scheme is $O(\binom{n}{t})$, and from the discussion in Section 8, we know *t* is chosen as a small number, so the communication complexity is uniformly bounded.

B Proof of the security of the asynchronous RSA scheme

We show how to simulate the adversary's view, given access to an RSA signing oracle which we use only when the adversary asks for a signature share from an uncorrupted party. Let $p_1,..., p_t$ be the set of corrupted parties, and assume $\{d_1,..., d_{l-1}\}$ are shares contained in $\{S_{p_1}, S_{p_2}, ..., S_{p_t}\}$.

First, simulate the adversary' view of asynchronous verifiable secret sharing as Appendix A, then perform the following steps.

(1) compute $\hat{x}_i = m^{2\times\hat{d}_i} \bmod N$ for $1 \le i \le l-1$

(2) set $\hat{x}_l = m^{2\times d} /(\hat{x}_{public}\prod_{i=1}^{l-1}\hat{x}_i) \bmod N$, where $\hat{x}_{public} = m^{2\times\hat{d}_{public}} \bmod N$

Our proof is similar to that of Shoup's scheme [4]. With regard to the "proofs of correctness", one can invoke the random oracle mode for the hash function H' to get soundness and statistical zero knowledge.

First, consider soundness. We want to show that the adversary cannot construct except with negligible probability, a proof of correctness for an incorrect share. Let *x* and x_i be given, along with a valid proof of correctness (*z, c*). We have $c = H'(v, x, v_i, x_i, v', x')$, where

$$v' = v^z v_i^{-c}, x' = x^{2\times z} x_i^{-c}.$$

Now, v_i, v', x', x_i, x^2 are all easily seen to lie in Q_N, and we are assuming that *v* generates Q_N. So we have

$$v_i = v^{d_i}, x_i = v^{\beta}, v' = v^{\gamma}, x^2 = v^{\alpha}, x' = v^{\delta},$$

for some integers α, β, γ, δ. Moreover,

$$z - cd_i \equiv \gamma \bmod M \text{ and } \alpha z - c\beta \equiv \delta \bmod M.$$

Multiplying the first equation by α and subtracting the second, we have

$$c(\beta - d_i\alpha) \equiv \alpha\gamma - \delta \bmod M \qquad (1)$$

Now, a share is correct if and only if

$$\beta \equiv d_i\alpha \bmod M \qquad (2)$$

If (2) fails to hold, then it must fail to hold mod *p'* or mod *q'*, and so (1) uniquely determines *c* modulo one of these primes. But in the random oracle model, the distribution of *c* is uniform and independent of the inputs to the hash function, and so this even happens with negligible probability.

Second, consider zero-knowledge simulatability. We can construct a simulator that simulates the adversary's view without knowing the value d. This view includes the values of the random oracle at those points where the adversary has queried the oracle, so the simulator is in complete change of the random oracle. Whenever the adversary makes a query to the random oracle, if the oracle has not been previously defined at the given point, the simulator defines it to be a random value, and in any case returns the value to the adversary. When an uncorrupted party is supposed to generate a proof of correctness for a given *x*, x_i, the simulator chooses $c \in_R [0, 2^{L_1} - 1]$ and $z \in_R [0, ..., 2^{L(lN^2)+2L_1} - 1]$ at random, and for given values *x* and x_i, defines the value of the random oracle at $(v, x, v_i, x_i, v^z v_i^{-c}, x^{2\times z} x_i^{-c})$ to be *c*. With all but negligible probability, the simulator has not defined the random oracle at this point before, and so it is free to do so now. The proof is just (*z, c*). It is straight forward to verify that the distribution produced by this simulator is statistically close to perfect.

From soundness, we get the robustness of the threshold signature scheme. From zero-knowledge, and the above arguments, we get the non-forgeability of the threshold signature scheme, assuming that the standard RSA signature scheme is secure, i.e., existentially non-forgeable against adaptive chosen message attack.

Notice in the above analysis, we omit the discussion of the case of the outputs of random oracle equal to the generator v. The probability of such a case is negligible; otherwise, the standard RSA signature scheme is insecure.

C Security analysis of the asynchronous refresh scheme

We have to show the proposed scheme satisfies the *liveness*, *correctness*, *privacy* properties.

Liveness. Since there are at least $t+1$ honest parties, and there is at least a candidate of set. Then the validated Byzantine agreement scheme will terminate with a candidate of set as the output within a phases provided the adversary delivers all associated messages within phases.

Correctness. Fix a point in time where a set $\mathcal{H}$ of at least t + 1 honest parties has completed the refresh scheme and not yet detected the next clock tick for the beginning of the next phase. Due to the correctness of the AVSS scheme, the secret key *d* is shared among new *l* shares. And since the next phase hasn't started yet, then, for any $t+1$ honest parties, all *l* shares are contained in their $t+1$ share sets. So any $t+1$ honest parties can reconstruct the secret key and the reconstructed value equals *d,* except with negligible probability.

Privacy. Assume that we have constructed a simulator to simulate previous phases $\{0,1,...,\tau-1\}$ (In appendix B, we construct a simulator for the initial phase 0, so this holds,) we now consider constructing a simulator for phase τ.

We only consider the initial scheme here (To the modified scheme, proof is similar.).

For simplification, let $p_1,...,p_t$ be the set of corrupted parties in the *current* phase, and $\{d_1,d_2,...,d_{l-1}\}$ are the shares that could be observed by the adversary in the *next* phase. Now, we simulate the adversary's view on the asynchronous refresh scheme. First, for those $(l-1)$ shares contained in share sets of $p_1,...,p_t$, perform as the same steps as that in the asynchronous refresh scheme; for that remaining one, suppose it to be d_i, perform the following simulation.

Assume that $\hat{d}_{i,1},...,\hat{d}_{i,l-1}$ can be observed by the adversary. Choose shares $\hat{d}_{i,1},...,\hat{d}_{i,l} \in_R [-N^2,N^2]$, $\hat{d}_{i,public} = \hat{d}_i - \sum_{j=1}^{l} \hat{d}_{i,j}$. Compute $\hat{w}_{i,1} = v^{\hat{d}_{i,1}},...,\hat{w}_{i,l-1} = v^{d_{i,l-1}} \bmod N$ and set $\hat{w}_{i,l} = v^{\hat{d}_{i,l}} = \hat{V}_i / v^{\hat{d}_{i,public}} \prod_{j=1}^{l-1} \hat{w}_{i,j} \bmod N$ ($\hat{V}_i$ is the "correct" commitment for the current $\hat{d}_i$, note that $\hat{V}_i = v^d / v^{\hat{d}_{public}} \prod_{j=1,j=i}^{l} v^{\hat{d}_j}$).

Then, new $\hat{d}_i$, $\hat{d}_{public}$ and $\hat{V}_i$ for $(1 \le i \le l)$ are computed.

With same method of Appendix A, we can prove that the above simulation is statistically indistinguishable to the adversary's view, and the privacy still holds in other phases rather than 0.

Efficiency. The parties execute $(l-t) \times n$ AVSS schemes and one validated Byzantine agreement scheme. Because the communication complexity of scheme AVSS is uniformly bounded and the communication complexity of the validated Byzantine agreement scheme is probabilistically uniformly bounded (Cachin et al prove this in [6]; and here we use the result directly), the communication complexity of the asynchronous refresh scheme is probabilistically uniformly bounded.

Taken together, the security of the asynchronous refresh scheme is proved.