Asynchronous Proactive RSA

Abstract:

Nowadays, to model practical systems better, such as the Internet network and ad hoc networks, researchers usually regard these systems as asynchronous networks. Meanwhile, proactive secret sharing schemes are often employed to tolerate a mobile adversary. Considering both aspects, an asynchronous proactive threshold signature scheme is needed to keep computer systems secure.

So far, two asynchronous proactive secret sharing schemes have been proposed. One is proposed by Zhou in 2001, which is for RSA schemes. The other scheme is proposed by Cachin in 2002, which is a proactive secret sharing scheme for discrete-log schemes. There exist several drawbacks in both schemes. In Zhou's scheme, the formal security proof of this scheme is missing. Furthermore, Zhou's scheme needs to resort to the system administrator as the trusted third party for further run when some Byzantine errors occur. In Cachin's scheme, the building block is based on the threshold RSA scheme proposed by Shoup. However, how to proactivize Shoup's scheme is omitted in Cachin's scheme, so this scheme is incomplete.

In this paper, we present a complete provably secure asynchronous proactive RSA scheme (APRS). Our paper has four contributions. Firstly, we present a provably secure asynchronous verifiable secret sharing for RSA schemes (asynchronous verifiable additive secret sharing, AVASS), which is based on a verifiable additive secret sharing over integers. Secondly, we propose an asynchronous threshold RSA signature scheme that is based on the AVASS scheme and the random oracle model, and is capable of being proactivized. 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, combining the proactive secret sharing scheme and 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, such that an adversary who corrupts up to *t* parties can not break the whole system. However, when a threshold cryptosystem operates over a longer time period, the threshold cryptosystem needs to tolerate a mobile adversary [1], which may move from party to party and eventually corrupt every party in the system. Proactive secret sharing schemes address this problem by operating in phases; they can tolerate the corruption of up to *t* different parties during every phase [2]. Technically, this is done by the parties randomly re-sharing the shared private key at the beginning of each update phase. Knowledge of shares from previous phases does not give the adversary an advantage.

Most distributed computing protocols, such as threshold signature schemes, assume a synchronous network with a broadcast channel connecting all parties. This assumption may lead to some vulnerability in practice. For example, when deployed in a distributed system over a wide-area network with only loosely synchronized clocks, such systems are vulnerable to timing attacks. Recently, researchers begin to model computer systems as asynchronous networks when constructing practical distributed computing schemes. A long term research project Malicious- and Accidental-Fault Tolerance for Internet Applications (MAFTIA) of the department of information security and cryptography of the IBM Zurich research lab assumes an asynchronous network model, and researchers in IBM Zurich research lab proposed many asynchronous schemes, such as non-interactive threshold signature [3], asynchronous Byzantine agreement [4][5], asynchronous secret sharing and proactive secret sharing scheme [6], asynchronous protocol for distributed computation of RSA inverses[7], 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 [8]. In 2001, Zhou proposed an online certification authority scheme for the Internet, which is also modeled as asynchronous networks [9][10]. Furthermore, Zhou considers ad hoc networks as asynchronous networks too [11] and present some ideas for constructing proactive secret sharing for ad hoc networks.

When considering proactive secret sharing in asynchronous networks, asynchronous proactive secret sharing schemes and asynchronous threshold signature schemes are needed. In this paper, we present a complete provably secure asynchronous proactive threshold RSA signature scheme to tolerate a mobile adversary in asynchronous networks.

1.1 Related work

Let's see synchronous proactive signature first. While many synchronous proactive signature schemes for discrete-log schemes were presented and well studied [12, 13, 14], 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 [12, 13]. Then, Rabin proposed a simplified protocol using similar ideas [14]. Recently, Jarecki presented a more efficient proactive RSA scheme [23] based on Rabin's scheme. The above schemes are similar in two aspects. They all use secret sharing "in two levels" to make the shares in the top level backed-up in the secondary level. For example, party p_i knows a number d_i such that $\sum d_i = d$. Meanwhile, each d_i is a verifiable secret shared among the parties. Furthermore, all these schemes require some form of additive rather than polynomial secret-sharing in the solutions that tolerate an optimal adversarial threshold. From synchronous proactive RSA schemes, we learn that the additive secret sharing and backing up shares are very important in constructing proactive RSA schemes. Our asynchronous proactive RSA scheme is built on these two techniques.

In addition the above proactive RSA scheme, Luo et al also proposed a proactive RSA scheme for ad hoc networks, which built on polynomial secret sharing[15,16,17]. Unfortunately, due to lack of a formal proof, though this scheme has been studied for several years, this scheme has been proved faulty in two recent papers [18][19]. It seems that how to proactivize RSA schemes using polynomial secret sharing is still a great challenge. In addition, a formal proof of security of proactive RSA scheme seems to be necessary.

In 1999, Shoup proposed a non-interactive threshold RSA signature which could be used in asynchronous networks [3]. However, there are still no asynchronous discrete-log threshold signature schemes now. So far, two asynchronous proactive secret sharing schemes are presented. One is proposed by Zhou in 2001, which is a proactive secret sharing scheme for RSA schemes and based on Rabin's RSA signature scheme. The other scheme is proposed by Cachin in 2002, which is a proactive secret sharing scheme for discrete-log schemes. There exist several drawbacks in both schemes. In Zhou's scheme, the formal security proof of this scheme is missing. Furthermore, Zhou's scheme needs to resort to the system administrator to act as a trusted third party for further run when some Byzantine errors occur. In Cachin's scheme, though Cachin's asynchronous proactive scheme is for discrete-log schemes, the building block is based on Shoup's threshold RSA scheme. However, how to proactivize Shoup's scheme is omitted in Cachin's scheme. Furthermore, there are still no corresponding asynchronous discrete-log signature schemes for asynchronous proactive schemes. Considering both aspects, Cachin's scheme is incomplete now.

1.2 Our contributions

In this paper, we present a complete provably secure asynchronous proactive RSA (APR) scheme. Our paper has four contributions.

Firstly, we present a provably secure asynchronous verifiable secret sharing for RSA schemes (asynchronous verifiable additive secret sharing, AVASS), which is based on a verifiable additive secret sharing over integers. Secondly, we propose an asynchronous threshold RSA signature scheme that is based on the AVASS scheme and the random oracle model, and is capable of being proactivized. 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, combining the proactive secret sharing scheme and the threshold RSA scheme, we achieve a complete provably secure asynchronous proactive RSA scheme.

1.3 Discussion of techniques used

To describe the techniques used clearly, we first see the whole layered architecture of our asynchronous proactive RSA scheme in Figure 1.

The lowest layer is the broadcast primitive, and Bracha's broadcast scheme [20] is used in this layer. The second layer is the asynchronous verifiable secret sharing scheme, and our AVASS scheme is used in this layer. Similar to Rabin's scheme, the AVASS scheme has two-levels, and is based on the additive secret sharing over integers. However, unlike Rabin's scheme, which back up an additive secret share by a polynomial secret sharing, the AVASS back up an additive secret shares at several parties in a redundant way. The third layer is the asynchronous RSA signature, and our proposed asynchronous RSA scheme is used in this layer. Like Rabin's scheme, our RSA scheme is based on the additive secret sharing. However, our scheme employs a technique of Shoup's scheme to prove the robustness of the scheme, while Rabin's scheme employs a different technique. The fourth layer is the threshold coin-tossing scheme, and we implement such a scheme on the basis of our RSA scheme. The fifth layer is the Byzantine agreement, and we use Toueg's Byzantine agreement and replace Rabin dealer in this scheme with our coin-tossing scheme. The sixth layer is the validated Byzantine agreement, and we use Cachin's validated Byzantine agreement and replace the Byzantine agreement in this scheme with the modified Touge's scheme. Finally, the top level is the asynchronous refresh scheme, and we implement such scheme based on the AVASS scheme and validated Byzantine agreement. Our refresh scheme adopts the similar idea with Rabin's scheme except that our scheme uses validated Byzantine agreement to replace broadcast channel in synchrotrons networks.

Combining all these, 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 AVASS scheme and the asynchronous refresh scheme). So does our proof of the security of the whole scheme.

1.4 Outline

The paper is organized as follows. Section 2 presents the system model and the cryptographic assumptions used. An asynchronous verifiable secret sharing scheme is presented in Section 3, and an asynchronous threshold RSA scheme is presented in Section 4. Then a threshold coin-tossing scheme is proposed in Section 5. In Section 6, an asynchronous proactive secret sharing scheme is presented. In section 7, we analyze the efficiency of the whole scheme and give some discussions. In Appendix A, B, C, the proof of the AVASS scheme, the asynchronous RSA scheme and the asynchronous refresh scheme are described, respectively.

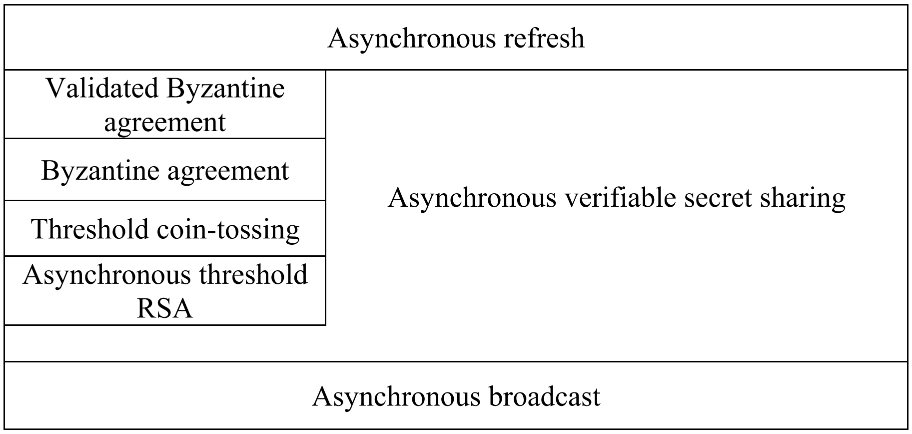

Figure 1 The whole layered architecture of the asynchronous proactive RSA scheme

2 System model and cryptographic assumptions

We adopt the basic system model from [3][4][5], which describes an 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 system model

We work in one of the several standard models of threshold cryptography and distributed systems, known in short as asynchronous, secure links, trusted dealer, static but proactive adversary model. This system model is as same as that of [4, 5]. We can only give a brief overview here; for the details, see [4,5] .

The network consists of *n* parties $p_1,..., p_n$, which are linked by a secure asynchronous channel that provides privacy and authenticity with scheduling determined by the adversary. There is an adversary, which is a probabilistic interactive Turing machine (PITM) that runs in polynomial time (in *k*). Some parties are controlled by the adversary and called *corrupted*; the remaining parties are called *honest*. An adversary that corrupts at most t servers is called *t-limited*. The scheme these parties run aims at tolerating a presence of the so-called "mobile", which is strictly stronger than "threshold", adversary who is a probabilistic polynomial time algorithm, and who can *statically* (i.e., at the beginning of the life time of the scheme) schedule up to *t* arbitrarily malicious faults among any *n* of the parties, for any $t < n/3$, independently in every *update phase*. A single time signal or *clock tick* is used to activate parties to inform the start of every phase locally. The adversary may corrupt up to *t* servers who are in the same local phase. Furthermore, for simplicity, we assume that secure channels in the proactive model guarantee that messages are delivered in the same local phase in which they are sent. More precisely, a message sent in some local phase of the sender arrives when the receiver is in the same local phase or it is invariably lost. Under these restrictions, we leave all scheduling up to the adversary (See [6] for more discussion about this topic). In practice, such proactive secure channels might be implemented by re-keying every point-to-point link when a phase change occurs, as discussed our asynchronous refresh scheme in Section 6.

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.

In our setting, the *message complexity* of a scheme is defined as the number of associated messages (generated by honest parties). Similarly, the *communication complexity* of a scheme is defined as the bit length of all associated messages generated by honest parties.

2.2 Cryptographic assumptions

The RSA modulus is $N = pq$, where p and q *are* random two 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 $PK = (N, e)$. The private key $d \in Z_{\Phi(N)}$ should be an even number. 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.

3 An asynchronous verifiable secret sharing scheme

In this section, we propose an asynchronous verifiable secret sharing (AVSS) scheme for RSA schemes, which is employed to build the asynchronous RSA scheme in Section 4.

3.1 Definition of the AVSS scheme

A scheme to share a secret s consists of a *sharing* stage and a *reconstruction* stage as follows.

Sharing stage. The sharing stage starts when a party initializes the scheme. In this case, we say the party *initializes a sharing*. There is a special party p_d, called the *dealer*, which is activated additionally on an input message of the form (in, share, s). If this occurs, we say p_d *shares s* among the group. A party is said to *complete the sharing* when it generates an output of the form (out, shared).

Reconstruction stage. After a party has completed the sharing, it may be activated on a message (in, reconstruct). In this case, we say the party *starts the reconstruction*. At the end of the reconstruction stage, every party should output the shared secret. A party p_i terminates the reconstruction stage by generating an output of the form (out, reconstructed, z_i). In this case, we say p_i reconstructs z_i.

The definition of our AVSS is adopted from [6].

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 $d = s$.
2. If an honest party p_i reconstructs z_i, then $z_i = z$.

Privacy: If an honest dealer has shared d and less than $t + 1$ honest parties have started the reconstruction, the adversary's view is statistically independent of d.

3.2 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 integers, and the top level is the AVASS scheme.

We outline the whole scheme informally first.

Let's see the ASS scheme, which is a (l,l) scheme, where $l = \binom{n}{t}$. The shared secret $d \in Z_{\Phi(N)}$ and d is an even number. The dealer chooses and hands p_i value $d_i \in_R [-lN^2, lN^2]$ for $1 \le i \le l$ such that d_i is an even number. 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 $C_{l+1} = v^d$.

Now, we consider how to 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 equal $\{d_i \mid 1 \le i \le l \wedge p \notin P_i\}$. Note 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 $I_p = \{i \mid 1 \le i \le l \wedge p \notin P_i\}$. Obviously, 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 [20].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 $1 \le i \le l$ such that $d_i \in_R [-lN^2 ... lN^2]$ and d_i is an even number. The corresponding witness is *C* 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 in the *send* messages, the dealer sends to every party *p* share set S_p and the commitment array *C*, respectively.

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 $d_i \in S_p$. 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 $S_{p_{i,j}} \in S_{p_i} \cap S_{p_j}$ 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( ) , every party computes its share set from the received share sets (agree on C means Cs in the received *echo* messages 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 $S_{p_{j,m}} \in S_{p_j} \cap S_{p_m}$ 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 it holds $C_{l+1} = v^{d_{public}} \prod_{j=1}^{l} C_j$ ($C_{l+1} = v^d$),

$d_i \in [-lN^2..lN^2]$, and d_i is an even number, 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.

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

4 An asynchronous RSA 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.

4.1 Implementation of the asynchronous RSA scheme

Given a message *m*, its signature under the public key (*N*, *e*) is $m^d \bmod N$. 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 $m^d = m^{d_{public} + \sum_{i=1}^{l} d_i} = m^{d_{public}} \prod_{i=1}^{l} m^{d_i} \bmod N$.

We now describe how a signature share on a message *m* is generated. Let $x = H(m)$. p_i has $|S_{p_i}|$ signature shares, and every signature share consists of $x_i = x^{d_i} \in Q_n$ (d_i is an even number), 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* 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 a security parameter (L_1=128, say). To construct the proof of correctness, party p_i chooses a random number $r \in \{0,...,2^{L(lN^2)+2L_1} - 1\}$ and *r* is an even number, computes

$$v' = v^r,\ x' = x^r,\ c = H'(v, x, v_i, x_i, v', x'), z = d_i c + r$$

The proof of correctness is (*z*, *c*).

To verify this proof of correctness, one checks that

$$c = H'(v, x, v_i, x_i, v^z v_i^{-c}, x^z x_i^{-c})$$ and z is an even number.

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.

Let $x = H(m) \in Z_N^*$, and assume that $x_i = x^{d_i}$ for $(1 \le i \le l)$. Then to combine shares, we compute $y = x^{d_{public}} \prod_{i=1}^{l} x_i$. Thus we achieve the signature *y* of the message *m* such that $y^e = x \bmod N$.

4.2 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.

5 A threshold coin-tossing scheme

In this section, we first introduce the concept of the validated Byzantine agreement (VBA) scheme, and its relations to Byzantine agreement scheme and the threshold coin-tossing scheme, then propose a threshold coin-tossing scheme based on the RSA scheme in Section 4.

5.1 The VBA scheme

In constructing the asynchronous refresh scheme in Section 6, the building blocks are the AVASS scheme and the VBA scheme. The standard notion of a Byzantine agreement implements only a binary decision in asynchronous networks. A VBA [5] 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.

Cachin presented a complete VBA scheme in [5], which was built on the basis of a Byzantine agreement in [4]. However, we cannot use this VBA scheme completely, since the Byzantine agreement in this scheme is not appropriate for our use. In constructing Cachin's Byzantine agreement, two threshold signature schemes are used. One is a $(n, n-t, t)$ threshold scheme, and the other is a $(n, t+1, t)$ threshold scheme. If we adopt this Byzantine agreement, we have to refresh shares of both threshold signature schemes, which is very complex and hard. For simplicity, we use the Byzantine agreement proposed by Toueg in [21]. In Toueg's scheme, a Rabin dealer is used to generate random common coins for Byzantine agreement schemes. A Rabin dealer has some drawbacks and is not appropriate for our use, so we select to use the threshold coin-tossing scheme to generate random common coins. In [4][5], Cachin propose a threshold coin-tossing scheme based on computational Diffile-Hellman (CDH) assumption and a Byzantine agreement based on this coin-tossing scheme. However, we cannot use Cachin's coin-tossing scheme, since we have to build all schemes based on our asynchronous RSA scheme. (S. Micali et al proposed such ideas and a complete scheme in [22]. In [7], Cachin also proposed to use RSA inverse to implement verifiable random functions (VRFs), but no detailed description.)

5.2 Definition of threshold coin-tossing scheme

In this section, 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. A threshold coin-tossing scheme satisfies the following conditions for any *t-limited* adversary:

Robustness. It is computationally infeasible 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 $F(C) = b$.

5.3 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.

5.4 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.

Replacing Rabin dealer with our coin-tossing scheme in Touge's scheme and the Byzantine agreement with the modified Touge's scheme in Cachin's VBA schemes, we achieve a VBA scheme appropriate for our use.

6 An asynchronous proactive secret sharing scheme

In Section 2, we described an AVASS scheme. In this Section, we present a refresh 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.

6.1 Definition of asynchronous refresh scheme.

Our definition of an asynchronous secure refresh is similar to that of [6].

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: In any polynomial number of consecutive executions of the scheme, the adversary's view is statistically independent of *d*.

Note 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 [6] for more discussion about this topic).

6.2 Implementation of the asynchronous refresh scheme

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 VBA scheme, then use the VBA scheme 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.

Note 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 $ID\,|\,j$, the signature is computed on ($ID\,|\,j, \tau,$ ready). A list Π_j of $2t + 1$ such signatures 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 $L_i = \{(j, \Pi_j)\}$ 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**(τ, L_i), which verifies that L_i contains *l* sharing with the proofs that these sharings will actually terminate. It is true if and only if $|L_i| = l$ and for every $(j, \Pi_j) \in L_i$, the list Π_j contains at least $2t + 1$ valid signatures on *ready* messages from distinct parties.

3. After p_i decides in the VBA scheme for a set 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.

The above scheme still has a minor flaw. After many rounds of share updates, the absolute value of d_{public} might be very large. To avoid this case, we give some improvements here. We specify one threshold value $D_{\max}$ for the absolute value of d_{public}. Once $d_{public} > D_{\max}$, $d_{i,j} (1 \le i \le l, 1 \le j \le l)$ are selected randomly from the range of $[-N^2, 0]$, and once $-d_{public} > D_{\max}$, $d_{i,j} (1 \le i \le l, 1 \le j \le l)$ are selected randomly from the range of $[0, N^2]$. Otherwise, $d_{i,j} (1 \le i \le l, 1 \le j \le l)$ are selected randomly from the range of $[-N^2, N^2]$.

Theorem 4. In the random oracle model, the asynchronous refresh scheme is secure assuming the standard RSA signature scheme is secure.

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

7 Discussion

Clearly, the message complexity and the communication complexity of our refresh 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, we can optimize the asynchronous RSA signature scheme as follows when parties are honest. 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

$Sig_{p_2} = m^{\sum d_i}, d_i \in (\overline{S}_{p_1} \cap S_{p_2})$, and so on. The final signature $Sig(m) = m^{public} \prod_{i=1}^{t+1} Sig_{p_i}$ .Thus 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. However, the drawback that the message complexity and the communication complexity of the refresh scheme are both $O(\binom{n}{t})$, renders the whole scheme very inefficient. Such a drawback makes our scheme only applicable to cases when *t* is very small.

So far, all proactive RSA schemes have to employ additive secret sharing [12][13][14][23]. In asynchronous networks, since no broadcast channel exists, additive secret shares have to been backed up in many parties in a redundant way. Thus the whole scheme becomes very inefficient.

Furthermore, we have also considered proactivizing the polynomial secret sharing. Our initial idea is to proactivize Shoup's scheme and presents a preliminary scheme. However, we find out that it's hard to constructor a simulator. Though in [12][14], polynomial secret sharing is used , however, the proactivization of both scheme is implemented by additive secret sharing.

In [6], Cachin proposed an efficient proactive secret sharing for discrete-log schemes. However, no corresponding asynchronous threshold signature scheme was given.

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

[4] 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.

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

[6] 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.

[7] 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.

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

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

[10] 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.
[11] L. Zhou and Z. J. Haas. Securing Ad Hoc Networks. IEEE Network Magazine, 13(6):24–30, 1999.
[12] 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.
[13] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Proactive RSA. In *Proc. of Crypto'97*, pages 440–454, 1997.
[14] T. Rabin. A simplified approach to threshold and proactive RSA. in *Proc. CRYPTO '98*, pp. 89–104, Springer, 1998.
[15] 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.
[16] 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.
[17] 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.
[18] 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.
[19] 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.
[20] 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.
[22] 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.
[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, 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,

$v^{d'_i} \bmod N = v^{d_i} \bmod N$. This implies also $v^{d'_i - d_i} \bmod N = 1$. Since the order of Q_N is M, $d_i - d'_i = kM, k \in Z \wedge k \neq 0$. Thus, since d_i, d'_i and N are known, one can easily compute M in polynomial time, 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 $d''_i \in S_j$, where $1 \leq i \leq l$. The remaining proof is similar to the first part.

Privacy. We use the assumption of the asynchronous RSA scheme is secure to prove the privacy. The security of the asynchronous RSA scheme will be given in Section 4. Suppose the asynchronous RSA scheme is secure, the privacy holds. Otherwise, the adversary can forge RSA signatures successfully with non-negligible probability.

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 $p_1,\ldots,p_t$ be the set of corrupted parties, and assume $\{d_1,\ldots,d_{l-1}\}$ are shares contained in $\{S_{p_1}, S_{p_2}, \ldots, S_{p_t}\}$.

A natural idea is to construct a simulator similar with Rabin's simulator. However, Jarecki pointed out a flaw of the simulator in Rabin's scheme and give an improved simulation [23]. So we adopt Jarecki's method to construct the simulator of the asynchronous RSA scheme. We first describe the flaw of Rabin's simulation. Then we give a simulator based on Jarecki's method.

If we adopt Rabin's simulator, then we should choose shares $\hat{d}_1,\ldots,\hat{d}_{l-1} \in_R [-lN^2..lN^2]$, and $\hat{d}_{public} \in_R [-l^2N^2...l^2N^2+N]$. However, such selection is wrong. Since $d_{public} = d - \sum_{i=1}^{l} d_i$ and $d_i (i \leq i \leq l)$ have uniform probability distributions. Therefore, d_{public} has a normal probability distribution.

To avoid the flaw in Rabin's simulator, we use a method similar with Jarecki's to construct a simulator. Now, we simulate the adversary's view on the AVASS scheme first.

Choose secret value $\hat{d} \in [0, N-1]$ and $\hat{d}$ is an even number, share $\hat{d}_1,..., \hat{d}_l \in_R [-lN^2..lN^2]$ and $\hat{d}_1,..., \hat{d}_l$ are even number, and $\hat{d}_{public} = \hat{d} - \sum_{i=1}^{l} \hat{d}_i$

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

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

3. (a) Perform as the same operations as the AVASS scheme for $p_1,..., p_t$

   (b) Perform as the same operations as the AVASS scheme when interacting with $p_1,..., p_t$. Send random messages when interacting with other parties.

We have $N - \phi(N) = pq - (p-1)(q-1) = p + q - 1 = O(N^{1/2})$, and from this a simple calculation shows that the statistical distance between the uniform distribution on $[0, N-1]$ and the uniform distribution on $[0, \phi(N)]$ is $O(N^{-1/2})$. Thus, the above simulation is statistically indistinguishable to the adversary.

Then, we simulate the adversary's view on the asynchronous RSA scheme.

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

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

Our proof is similar to that of Shoup's scheme [3].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

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

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,

$$z - cd_i \equiv \gamma \bmod M \text{ and } \frac{\alpha z}{2} - c\beta \equiv \delta \bmod M .$$

Multiplying the first equation by α and subtracting the second multiplying by 2, we have

$$c(2\beta - d_i\alpha) \equiv \alpha\gamma - 2\delta \bmod M \qquad (1)$$

Now, a share is correct if and only if

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

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_i. 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 $c \in \{0,...2^{L_1}-1\}$ and $z \in 0,...,2^{L(lN^2)+2L_1}-1$ at random, and *z* is an even number , and for given values *x* and x_i, defines the value of the random oracle at $(v, x, v_i, x_i, v^z v_i^{-c}, x^z x_i^{-c})$ 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.

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 VBA 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 *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. Since the correctness of the AVASS 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. We show that the adversary's view in an execution of the scheme is statistically independent of *d*. Similar to the proof of privacy of the AVASS scheme; we prove privacy of the asynchronous refresh scheme by proving the security of the asynchronous RSA scheme in the current phase. Note that our proof the security of asynchronous RSA scheme in appendix B is only applicable to the asynchronous RSA scheme for the initial phase (Just after the dealer of the system completes the initialization of the system). For the other phases (When some rounds of refreshing have been completed), the proof should be a little different.

Assume that we have constructed a simulator to simulate previous phases $\{1,2,...,\tau-1\}$ (In appendix B, we construct a simulator for the initial phase 1, so this holds,) we now consider constructing a simulator for phase τ .

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

For simplifying, 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 the asynchronous refresh scheme; for that remaining one, suppose it to be d_i, perform the following simulation.

Assume that $\hat{d}_{i,1},\ldots,\hat{d}_{i,l-1}$ can be observed by the adversary. Choose shares $\hat{d}_{i,1},\ldots,\hat{d}_{i,l} \in_R [-N^2 \ldots N^2]$ and $\hat{d}_{i,1},\ldots,\hat{d}_{i,l}$ are even numbers, $\hat{d}_{i,public} = \hat{d}_i - \sum_{j=1}^{l} \hat{d}_{i,j}$ and $\hat{d}_{i,public}$ is an even number. Compute $\hat{w}_{i,1} = g^{\hat{d}_{i,1}},\ldots,\hat{w}_{i,l-1} = g^{d_{i,l-1}} \bmod N$ and set $\hat{w}_{i,l} = g^{\hat{d}_{i,l}} = g^{\hat{d}_i} / g^{\hat{d}_{i,public}} \prod_{j=1}^{l-1} \hat{w}_{i,j} \bmod N$.

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

Clearly, the above simulation is statistically indistinguishable to the adversary. We then simulate the adversary's view on the signing scheme with the similar method used in appendix B.

With the above simulation, we can prove that asynchronous RSA scheme is still secure in other phases rather than 1. Then the security of the asynchronous scheme is proved.