# Implementation and Discussion of Threshold RSA

Hanna Lee, Hao Shen, Brian Wheatman

May 11, 2016

## Abstract

A threshold cryptosystem involves collaboration among $k$ of $n$ users to take some cryptographic action. These actions could include encrypting, decrypting, signing and verifying. Threshold schemes are advantageous in situations where the involved parties wish to divide the power to sign or decrypt a message, so that no one party can take action without the support of some other parties, and in situations in which the parties wish to minimize the damage that a single compromised secret could cause. Such schemes are closely related to and often dependent on secret-sharing protocols such as Shamir's polynomial-based secret sharing scheme. When analyzing threshold cryptosystems, it's important to consider factors such as interactivity, trusted third parties, amount of required re-computation, efficiency, and burden on outside users. For our project, we implemented Threshold RSA, a system that is a variant of standard RSA signatures and that enables a size-$k$ subset of $n$ parties to produce a valid RSA signature on a message. Our project contributes a Python implementation of the scheme as well as a discussion of its advantages, disadvantages and performance. We intended to further extend Threshold RSA to support anonymity and deniability, such that parties could not determine which $k$ parties participated in producing the signature. However, we did not implement this extra feature due to time constants.

## 1 Overview of Threshold RSA

In Threshold RSA, the RSA modulus $N$ and public key $e$ are publicly known. The two primes $p$ and $q$, as well as the private key $d$ are kept secret and are unknown to anyone. The parties engage in collaborative but untrusting protocols to generate additive shares, $p_i, q_i$ and $d_i$, of the secrets. The protocols are collaborative because they requires parties to broadcast certain calculated values to other parties, and are untrusting because all parties can verify certain properties about the broadcast messages to ensure that other parties are honest.

Threshold RSA has several desirable properties:

- No trusted third party; no trusted central server
- Distributed system; distributed computation can parallelize work for greater efficiency
- A subset of fewer than $k$ people cannot produce a valid RSA signature on a message
- An attacker cannot use an existing signature to gain information about how to produce future signatures
- Parties do not need to reshuffle or regenerate their secrets after each signature
- Receivers of signatures produced by Threshold RSA can verify those signatures in the exact same manner as they would signatures produced by standard RSA; no extra work required

One potential drawback of Threshold RSA is that the members of the size-$k$ subset that collaborates to sign a message is known to the other parties. We imagined that this could lead to coercion, in the case that one party does not wish for the message to be signed, and therefore pressures or even threatens other parties to not sign, and is capable of knowing

whether or not those other parties did or did not sign. In our intended extended version of Threshold RSA, parties would choose a random unique identifier for each attempted signature and communicate through Tor to preserve anonymity.

### 1.1 Use Cases

We consider several use cases for a system such as Threshold RSA. Some of these use cases are particularly applicable for the version of Threshold RSA that also supports anonymity.

- A vote at an international conference - for matters of international security and interest, it is important that votes are private so that countries cannot coerce each other
- A union board endorsing a presidential candidate - using Threshold RSA ensures that some threshold number of board members actually does support this candidate, and it's not just one board member speaking for all of them.
- Business deals - if not all parties can attend every meeting, the group can still go forward with a deal

## 2 Threat Model

The threat model for Threshold RSA involves an adversary who is attempting to forge an RSA signature on some message of his choice, or is attempting to interfere with an honest signature attempt and prevent a size-$k$ subset from outputting a valid signature. This adversary may have access to the network (although the network traffic may be encrypted using standard TLS). The adversary may also have compromised, either through technical or personal means, the secret shares of $k-1$ parties. Although a computationally unbound adversary can break RSA by brute-force searching for the secret keys, we assume that the adversary lacks the computational resources to do this.

We consider the following types of adversaries, and later discuss how Threshold RSA performs in the face of each of these types of adversaries.

1. A spectator who only has access to the information that is intended to be public
2. A listener who can access all communication network
3. A man in the middle who can listen and modify what travels over the network
4. A dishonest participant in the scheme
5. An outside coercer who can extract secret information out of multiple participants

## 3 Previous Work

We based our implementation off of the description of the algorithm in H.L. Nguyen's paper *RSA Threshold Cryptography*. We are also aware of the existence of other threshold cryptosystems such as ring signatures. There are several important protocols that Threshold RSA uses, such as Shamir's secret sharing protocol, the BGW protocol (which we describe later), and standard RSA.

## 4 How Threshold RSA Works

### 4.1 Summary

First, all parties must agree on public parameters $N$, $n$, $k$, and $e$, where $n$ is the number of parties, $k$ is the threshold, $N$ is the RSA modulus, and $e$ is the RSA public key. The parties agree on an $N$ by first computing shares of $p$ and $q$, and then working together to verify that $N$ is the product of two primes, without ever revealing their own share of $p$ and $q$. This is a one-time start-up cost of the system. Following this, for every potential message, the parties each individually decide if they want to sign the message or not. If fewer than $k$ parties want to sign the message, the signature is aborted. Otherwise, some size-$k$ subset begins the process of producing a signature. This process begins with the subset-presigning algorithm, a protocol that occurs among the $k$ participating parties. During this algorithm each party computes and broadcasts values and commitments that allow the

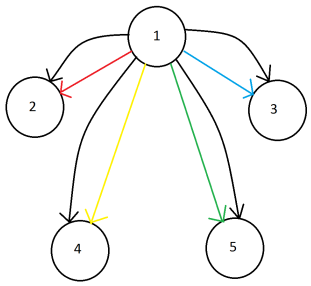


Figure 1: This shows the general sending flow. The colored lines indicate the different values sent to all other parties. The black lines show the commitments that are the same and sent to all parties

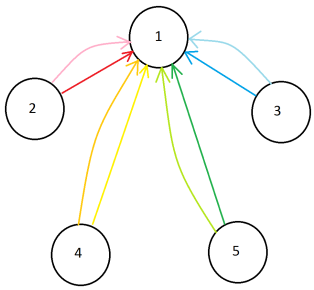


Figure 2: This shows the general receiving flow. Each party receives both computation output and a commitment from each other party and uses the commitments to check the computation is honest

other parties to determine if they are honest, and to gain knowledge about how to reform the secret key $d$ without the participation of the other $n-k$ parties. Then, each of the $k$ parties produces a signature share on the message, broadcasts its signature share to the other $k-1$ parties, and verifies the signature shares that it receives from the other parties. If all parties agree that the signature shares are valid, then they combine their signature shares to produce the final signature. This final signature is equivalent to the normal RSA signature on message $m$, which is $m^d$.

The basic structure of most parts is that each party does some computation and outputs both values to each individual other party, and broadcasts commitments, or proofs to everybody that their computation is correct and honest.

## 4.2 Agreeing on an N

### 4.2.1 Distributed Computation of Shares of p and q

The parties use the same process to generate $p_i$ as they use to generate $q_i$. We describe the process for $p_i$, and the process for $q_i$ is identical. The goal is for each party to have a share $p_i$ such that

$$\sum_{i=1}^{n} p_i = p \tag{1}$$

First, the parties agree on a large number $M$. All arithmetic for generating $p_i$ is done modulo $M$. Then, each party selects a secret random $a_i$. The product of the $a_i$ is what we set as $p$. However, no one party should know what $p$ is. The $a_i$ form a multiplicative sharing of $p$, since

$$\prod_{i=1}^{n} a_i = p \tag{2}$$

We need to alter this to an additive sharing, i.e. give each party a $p_i$ such that equation (1) holds. We achieve this by running several iterations of the BGW protocol, described below.

### 4.2.2 Distributed Computation of N

After each party has a $p_i$ and $q_i$, they wish to compute $N$, where

$$N = (\sum_{i=1}^{n} p_i)(\sum_{i=1}^{n} q_i) \tag{3}$$

They achieve this by running the BGW protocol once, and then broadcasting the resulting shares and summing them to get $N$.

### 4.2.3 BGW protocol

This protocol takes as input the $p_i$ values and all of the $q_i$ values, and produces an additive sharing of the

quantity $N = (\sum_{i=n}^{n} p_i)) * (\sum_{i=n}^{n} q_i)$, meaning that no one knows $N$ at the end of the protocol, but every party gets a share $N_j$, and the sum of the $N_j$ values is equal to $N$.

#### 4.2.4 Verification that $N = pq$

Verifying $N$

We first check that $N$ is not divisible by any prime up to some bound

We then want to make sure it is the multiply of two primes

$1 = g^{(p-1)(q-1)} = g^{pq-(p+q)+1} \mod N$

$1 = g^{(p-1)(q-1)} = g^{N-(\sum_{i=1}^{n} p_i + \sum_{i=1}^{n} q_i)+1} \mod N$

$1 = g^{(p-1)(q-1)} = g^{N-p_1-q_1+1} * \Pi_{i=2}^{n} g^{-(p_i+q_i)} \mod N$

## 4.3

Secret Share Generation Given a public $N$ and public key $e$, we wish to find values of $d_i$ such that the sum is $d$, the private key, without any one person having knowledge of it

First, each party $i$ computes

$$\phi_i = \begin{cases} N - (p_1 + q_1) + 1 & \text{if } i = 1 \\ -(p_i q_i) & \text{if } i > 1 \end{cases}$$

Now we wish to generate the sum of $\phi_i \mod e$. To do this, each party breaks their $\phi_i$ into $n$ pieces $\phi_{i,j}$ such that $\sum_{j=1}^{n} \phi_{i,j} = \phi_i$. Each party distributes piece $k$ to party $k$. Each party $j$ sums up all $\sum_{i=0}^{n} \phi_{i,k}$ and takes it $\mod e$. These sums are broadcasted to everyone, allowing everyone to compute $\sum \phi_i \mod e$. Furthermore, no information is revealed about individual $\phi_i$ and only minimal information about $\phi$ is given as $e$ is small compared to $\phi$.

Now we define

$$\psi = \phi(N) \mod e = \sum_{i=1}^{n} \phi_i \mod e$$

and

$$\psi^{-}1 = \phi(N)^{-}1 \mod e$$

Each party $i$ can now compute

$$d_i = \begin{cases} \lfloor \frac{1-\phi_1 \phi^{-1}}{e} \rfloor & \text{if } i = 1 \\ \lfloor \frac{-\phi_1 \phi^{-1}}{e} \rfloor & \text{if } i > 1 \end{cases}$$

Given a value of $x$ in the range $[0, n]$ to account for the floors, $d_i$ satisfies the following.

$$\sum_{i=1}^{n} d_i + x = \frac{1 - \phi^{-1} \sum_{i=1}^{n} \phi_i}{e}$$

$$\left( \sum_{i=1}^{n} d_i + x \right) e = 1 \mod \phi(N)$$

.

The remaining value of $x$ can be caluclated by encrypting on a dummy message, $m$. Each party sends $m^{ed_i}$ to a given party. That party calculates $m' = \prod m^{e_d i}$ and performs calculations until $m'(m^x)^e = m \mod N$. That party then adds $x$ to its value of $d_i$.

## 4.4 Dealing Algorithm

Each party $i$ shares its section of the secret using Shamir Secret Sharing Algorithm.

$$f_i(x) = a_{i,k-1} x^{k-1} + ... + a_{i,1} x + d_i$$

then sends $f_i(j)$ to each party j. It also broadcasts $g^{a_{i,j}} \forall j$ to all parties as commitments on each coefficient. These can be used to prove the user was honest as follows.

$$g^{f_i(j)} = g^{a_{i,k-1} x^{k-1} + ... + a_{i,1} x + d_i} \mod N$$

$$g^{f_i(j)} = \Pi_{t=0}^{k-1} \left(g^{a_{i,t}}\right)^{j^t}$$

## 4.5 Subset Presigning Algorithm

The subset presigning algorithm needs to be run only once by the $k$ signing parties. This means that if $k$ parties wish to sign a message and they have never performed the subset presigning algorithm together, then they will perform it, but if they have ever performed the algorithm at some point in the past, then

they will not need to perform it. The algorithm generates values that are necessary for each party to produce its signature share on a message.

Each party $t_i$ generates a share $s_{t_i}$, such that the sum of the $s_{t_i}$ values is equal to the sum of the remaining private keys + some integer multiple, $x_I$, of $M$. Each of the parties then computes $x_I$ by searching through the space of all possible values, which is from $[k-n, k+1]$. The value of $x_I$ is publicly verifiable by all of the $k$ parties, and will be important for computing the final signature.

### 4.6 Signature Share generation

Each party produces a partial signature on the message

$$c_{t,i} = m^{d_{t_i}+s_{t_i}} \mod N$$

Along with a zero knowledge proof of knowing $d_{t_i} + s_{t,i}$. We note that $g^{d_{t_i}+s_{t,i}} = b_{t_i,0}h_{t_i}$ so we provide a proof that the discrete logarithm of $b_{t_i,0}h_{t_i}$ in base $g$ is equal to the discrete log of $c_{t,i}$ in base $m$. This proof can be used by Chaum and Pedersons protocol (Chaum, 1992). The formulation of this proof is very similar to Schnorrs Discrete Logarithm POK but provides two challenge messages rather than one.

At the end of this stage, all signers broadcast their signature to everyone in the signing party to verify.

### 4.7 Signature Share verification

Each party in the signing party verifies the individual signatures and proof of knowledge of every other party to make sure the signature is valid

### 4.8 Signature Share combination

Each party has published their signature share. Now, each party computes

$$m^{-x_I M}\Pi_{i=1}^{k} c_{t_i} = m^{-x_I M}\Pi_{i=1}^{k} m^{d_{t_i}+s_{t_i}} \mod N$$

$$m^{-x_I M}\Pi_{i=1}^{k} c_{t_i} = m^{-x_I M \sum_{i=1}^{k} d_{t_i}+s_{t_i}} \mod N$$

$$m^{-x_I M}\Pi_{i=1}^{k} c_{t_i} = m^{\sum_{i=1}^{n} d_i} = m^d \mod N$$

$$m^{-x_I M}\Pi_{i=1}^{k} c_{t_i} = s$$

This value $s$, is the final signature. It is equal to $m^d$, which is the signature that would have been produced by standard RSA.

## 5 Security of Threshold RSA

We analyze the security of Threshold RSA against each type of attacker described in our threat model

### 5.1 Spectator

The spectator has access to the public information. This means he gets the public key and modulo. To him this scheme is identical to normal RSA. And thus it has the same security properties and we take as given the fact that an attacker with only the public key can not generate a signature. Also if he cannot modify any messages he cannot change the output of the system.

### 5.2 Listener

If a listener has access to all messages then he can use the messages sent around in the dealing algorithm along with knowledge of Shamir's Secret Sharing Algorithm to collect all of the $d_i$ and be able to sign any message we wants. And thus from now on we will assume that each pair of participants are able to communicate securely. This could be done with normal RSA and each party has a separate public, private key pair that is known to all parties. With this added assumption the Listener gains no information that the Spectator does not have so he can not either sign a new message or interrupt the signing process

### 5.3 Man in the Middle

A man in the middle can both listen to the network and modify the contents of messages. Keeping in mind that each message going over the network can be encrypted, when he modify the message he does not know what he is changing it to. A Man in the middle is able to gain no additional information to

a listener so he is also not able to generate a signature. We want to insure that he is not able to disrupt the process and cause an invalid signature to be outputted.

1. When generating N every party checks at the end to ensure that it is the product of 2 primes, so any interference with generating N would result in a new N being generated, allowing the
2. To ensure that parties generate a proper share of the secret key a known message is signed and then verified
3. During the dealing algorithm along with the shares for Shamir's Secret Sharing Algorithm, each user also publishes commitments on the coefficients he used that can then be verified by all other users.
4. During the Subset Presigning Algorithm each party also computes a commitment to their share of the secret that can be checked by the other parties
5. During Signature Share Generation each party must produce a non interactive proof using the Chaum and Pederson protocol that

   $$c_{t_i} = m^{d_{t_i}+s_{t_i}} \mod N \Rightarrow D\log_m c_{t_i} = d_{t_i} + s_{t_i}$$

6. this proof is then verified during Signature Share Verification
7. At this point each party can independently calculate the final signature

Thus we can see that at each step each user submits both his share of information and a proof that the information is correct, so if any piece is changed it will be noticed and the stop repeated. So while a man in the middle could just stop a signature from being generated, that is as simple as stopping all communication, so we will ignore this case.

### 5.4 Participant

An attacking participant would try and sign a message alone, or cause a signature to be unknowingly invalid. As we showed in the previous section it is unfeasible by changing his messages to not get caught so he would be forced to submit valid messages, so he cannot cause a invalid signature to be outputted. However he could always invalidate a step stopping the protocol from outputting anything, but as we said above we are not considering this case. Thus we consider if he could sign a document on his own. Signing a message requires either knowing every bodies $d_i$ or the $d_i$ of k people and the $s_{t_i}$ of those same people, or determining $p$ and $q$. We will show that it is impossible for a single participant to gain anybody else's $d_i$. We show this by relying on the security of Shamir Secret Sharing Algorithm which is information theoretically secure for any group less than k to gain information about the secret. We also relay on the difficulty on the discrete log problem so that no commitments can be undone. Generating $p$ and $q$ both use the BGW protocol that also rely on Shamir Secret Sharing Algorithm for its security. Due to both of these a single one of the participants cannot generate a signature on their own.

Also during the protocol no information about any participants secret share of the secret key is leaked so any adversary is not learning information to help them sign a document after multiple signatures.

### 5.5 Convincer

There are two cases for a convincer. The first is that they have less than k people. In this case we can fall back on the above participant case and see that for each point we relayed on Shamir Secret Sharing Algorithm which is perfectly secure for less than k people. The second case is that they have more than k people, in this case they can just follow the algorithm properly and sign whatever they want.

## 6 Implementation

We used Python to set up a simulation of a network with a set of $n$ parties that can communicate with each other. We then follow the protocol described above so that given $n$ parties agree on the public information. Then, a subset of them can agree and

sign a message. The parties are also able to determine if the other parties are honest by verifying what the other parties broadcast. We check that the final signautre produced by our scheme is in fact equivalent to the signature that would be produced using standard RSA.

We were not able to generate p,q,and N properly because of time constraints that will be discussed below. We also did not set it up to run on different machines because it was not necessary to analyze the system. Lastly, we had hoped to set up the system to run over the TOR network to test giving each user anonymity, but again were limited by time.

The full implementation can be viewed here `https://github.com/hlee95/threshold-rsa`.

# 7 Discussion

## 7.1 Interactiveness

Interactiveness is the involvement of everyone to reach an agreement on public information by interacting with each other. This scheme consists of both interactive and noninteractive phases. Specifically, the only phases which are interactive are the Dealing and Subset Presigning Phases. All other phases involve each party computing values on their own. In this sense, this scheme is deemed partially interactive.

## 7.2 Share Refreshing

In this system, the construction of $d_i$ is only generated once per set of people. This is due to the fact that no information about $d_i$ is leaked to others when a person signs. Therefore $d_i$ can be reused any number of times with any set of signers. However, other values such as $s_{t_i}$ are publically generated from the set of $I$. So whenever the set of signers change, the Subset Presigning Phases must be processed. Furthermore, whenever we consider a different set of people, such as when people leave or enter, we must generate values from the beginning as none of the values are correct.

## 7.3 Threshold RSA vs. other $k$-of-$n$ signing schemes

Threshold RSA is more trustworthy than many other threshold signing schemes because there is no trusted central server. This means that no entity knows the secret key $d$, or the primes $p$ and $q$. This removes the possibility of an adversary compromising a server to gain all of the secrets.

Additionally, Threshold RSA has limited share refreshing, making it more attractive for frequent use, since it avoids the overhead associated with generating new secret shares.

We considered implementing threshold ring signatures instead of Threshold RSA, but the ring signature scheme requires an exponential number of operations ($\mathcal{O}(n^k)$ operations for naive ring signatures and $\mathcal{O}(2^t n \log)$ (Bresson, 2002)), making it less efficient than Threshold RSA, which requires a quadratic number of operations (mostly for broadcasting messages).

## 7.4 Expensive startup cost to generate N

When we ran our simulation of the system, we found that all parts of the Threshold RSA protocol are time-efficient, with the exception of the initial startup cost to find a "good" $N$ and the corresponding $p_i$ and $q_i$ shares for each party. A "good" $N$ is an $N$ that is the product of two primes. We noticed that the protocol for trying to find an $N$ is essentially a guess and check method, where the probability of getting a "good" $N$ is on the order of $10^{-6}$. When we ran our algorithms to generate shares of $p$ and $q$ and to verify $N$, we found that even after 16 hours of attempts, we could find no suitable $N$. We are certain that our algorithms are correct because when we tried them on smaller numbers they were able to produce valid results. We therefore decided to bypass the long startup cost of generating $N$ by randomly selecting two primes $p$ and $q$, and breaking them up into additive shares. This is inconsistent with the protocol, but is necessary in the simulation to be able to demonstrate the system in a timely manner. We note that during real use of the system, we would not use

this method, and we would use the proper algorithm to generate $N$ while keeping $p$ and $q$ private.

# 8 Conclusion

Overall, our project resulted in a working simulation of Threshold RSA. It has an expensive one-time start-up cost that may make it impractical for some uses. Future work on this project would involve running multiple clients on different machines and setting up the network communication between them, and then moving to using Tor to provide anonymity. If we are able to use Tor for the communication between parties, then our system will be beneficial in situations where the parties want to have deniability, to prevent other parties from coercing them. We see this is as a very applicable real-world system, especially for groups in which the parties are fixed (to prevent needing to start-up the system more than once). We plan to continue working on the project, and perhaps release a fully functioning system in the future.

# 9 References

Nguyen, H. L. (2005). RSA Threshold Cryptography. Department of Computer Science, University.

Chaum, D., & Pedersen, T. P. (1992, August). Wallet databases with observers. In Advances in CryptologyCRYPTO92 (pp. 89-105). Springer Berlin Heidelberg.

Shamir, A. (1979). How to share a secret. Communications of the ACM, 22(11), 612-613.

Bresson, E., Stern, J., & Szydlo, M. (2002). Threshold ring signatures and applications to ad-hoc groups. In Advances in CryptologyCrypto 2002 (pp. 465-480). Springer Berlin Heidelberg.