

RSA Threshold Cryptography

H.L. Nguyen

May 4, 2005

Dept. of Computer Science,
University of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB,
United Kingdom.
hn2503@bristol.ac.uk

Abstract

In this project, a new threshold signing scheme for RSA has been proposed. The scheme does not require a trusted third party and no secure information is leaked throughout the protocol. The time and storage complexity of the protocol is linear in the number of parties and no restriction is placed on the RSA moduli. Combined with the n -out-of- n key generation protocol of Boneh and Franklin, one has a complete solution for the threshold RSA problem with no trusted dealer. The complete protocol has also been implemented, a paper has been written and submitted to a conference on cryptography and coding.

Contents

1	Introduction	4
2	Applications of the protocol	6
2.1	Digital Signature	6
2.2	Distributed Certificate Authority	6
2.3	Electronic voting system and Internet card game protocols	6
2.4	Identification Scheme	7
3	Cryptography Techniques	8
3.1	Hard Problem	8
3.1.1	Factoring Problem	8
3.1.2	Discrete Logarithm Problem	8
3.2	Euler Theorem and Fermat Primality test	8
3.2.1	Euler Theorem	8
3.2.2	RSA case	8
3.2.3	Fermat Primality Test	9
3.3	RSA Algorithm	9
3.4	Shared RSA Threshold Decryption	9
3.4.1	Discrete Logarithm Approach	10
3.4.2	RSA Paillier Approach	10
4	Multi-party Computation Protocols	12
4.1	Shamir Secret Sharing Scheme and Lagrange Coefficient	12
4.1.1	Modulo non-prime	13
4.1.2	Sharing the final outcome	13
4.2	Benaloh Protocol	13
4.3	BGW Protocol	14
4.3.1	Privacy	15
4.3.2	Sharing the final outcome	15
4.3.3	Extension	15
5	Shared RSA Secret Keys Generation Protocol, an n-out-of-n Threshold Scheme	16
5.1	Problem Definition and Notation	16
5.2	Scheme Definition	17
5.2.1	Picking candidates and Distributed Sieving:	17
5.2.2	Distributed Computation of N:	18
5.2.3	Parallel Trial Division:	19
5.2.4	Load Balance Primality Test:	19
5.2.5	Private Key Generation:	20
5.2.6	Trial Decryption:	21
5.3	Discussion of the Above Scheme	22
5.3.1	Privacy	22
5.3.2	Smallest number of parties	22
5.3.3	Generating Prime number	22
5.3.4	Complexity	23

6	Partially Interactive Threshold RSA Signatures	24
6.1	Problem Definition and Notation	24
6.2	Scheme Definition	25
6.2.1	Dealing Algorithm:	25
6.2.2	Subset Presigning Algorithm:	26
6.2.3	Signature Share Generation Algorithm:	27
6.2.4	Signature Share Verification Algorithm:	27
6.2.5	Share Combining Algorithm:	28
6.3	Discussion of the Above Scheme	28
6.3.1	Interactiveness	28
6.3.2	Share Refreshing	28
6.3.3	Robustness	29
7	Design, Implementation and Testing	30
7.1	Choice of Language	30
7.2	Requirement Analysis	30
7.3	Network protocol	31
7.3.1	N-ary tree network structure	31
7.3.2	Fully connected network structure	32
7.4	Testing	34
7.5	Running the protocol	34
7.6	Limitation and Future work	35
8	Experiment Result and Discussion	36
8.1	Scaling to many parties	37
8.2	Doubling the length of RSA moduli	37
9	Conclusion	37

1 Introduction

Threshold decryption has been studied a lot for the last two decades. It is a branch of public key cryptography in general, and multi-party computation in particular. Essentially, in a k -out-of- n threshold crypto-system, denoted (k, n) where $1 < k \leq n$, for the RSA function [31], our aim is to generate and then split the secret decryption/signing exponent d into n different pieces, which are then distributed privately to n parties. This enables:

- Any k or more out of n total parties, when they come together, they can “reconstruct” the secret d in a way which enables them to decrypt or sign a message. This should be done in a way that does not reveal the value of d and its shares to any one in the scheme.
- Secondly, signing or decryption will be totally impossible in the circumstance where less than k parties are present.

The area of threshold cryptography has been pioneered by Adi Shamir in his 1978 paper [32], however the idea only took off when the problem was formally stated by Desmedt in [13]. Since then there has been much work devoted to the topic such as Desmedt and Frankel [14], Pedersen [29], Gennaro et. al. [21], and many more. However, the majority of these solutions are only for discrete logarithm based system that has a direct application to the Elgamal encryption and decryption algorithm [16]. The reason why discrete logarithm based threshold systems are easier to design is because the group in which one works has a publicly known order. Whereas, in the RSA signature scheme, the group we are working in has an unknown group order and so various technical problems arise. For example, standard polynomial interpolation over the ring $\mathbb{Z}_{\phi(N)}$ is hard as no party knows $\phi(N)$.

Another problem is that it is relatively easy to generate a shared discrete logarithm public/private key pair, but it is harder to generate a shared RSA public/private key pair, n -out-of- n threshold scheme, without the presence of a trusted third party. However, there was in breakthrough in the area of shared RSA key generation when both Boyd [7] and Frankel [19] independently proposed a simple and elegant solution for distributed RSA. The decryption key d is additively shared amongst n parties, $d = d_1 + d_2 + \dots + d_n$, signing is simply done as follows:

$$s = m^d = m^{d_1} \dots m^{d_n} \pmod{N},$$

and each $s_i = m^{d_i} \pmod{N}$ is called the partial signature or signature share.

Extending this idea, a number of new schemes for shared RSA key generation were proposed, for example, a complete solution for this problem was given in [11]. Unfortunately, the moduli N was assumed to be generated by a trusted dealer. The dealer, therefore can forge a signature on a message of his or her choosing. There was also something called general secure circuit evaluation techniques presented in [4, 8, 22, 34] as primality test can be done by using boolean circuit. However, this idea was too inefficient to be implemented in practice. So far, the best solution for this problem is probably the one that was built by Boneh and Franklin [5, 6], which does not require a trusted third party, and which can efficiently generate shared RSA keys that satisfy the above property. This solution is also the one, we have studied and implemented in the

first half of this project. The drawback of the scheme is that it only gives us a n -out-of- n threshold decryption that cannot be switched easily into a k -out-of- n threshold scheme.

In trying to solve the last piece of this problem, a number of threshold schemes for RSA have been proposed in the literature, most notable are Rabin's [30] and Shoup's [33] schemes. In Rabin's protocol, the author uses Shamir secret sharing to share the secret but on signing the k signing parties need to interact so as to recover the secrets of the non-signing parties. This removes the problem of working in a group of unknown order, but means the scheme leaks information about the additive shares of various parties. To get around this problem a *share-refreshing* protocol is given. All parts of Rabin's scheme require a large amount of interaction between the various parties.

Taking a different approach, Shoup provides a framework that leads to the possibility of applying the protocol in practice, where dealing, signature share generation, signature share verification and signature share combining are separated from each other and only the first part, i.e. dealing, requires interaction of the various parties. The scheme Shoup proposes is then fully non-interactive, bar the initial dealing phase. However, the drawbacks of his scheme are that it requires both a trusted dealer and strong RSA moduli. Hence, Shoup's scheme cannot be applied with the Boneh and Franklin shared key generation protocol.

In this project we give a new RSA threshold scheme [25] which does not require trusted dealers and which can be applied with the Boneh and Franklin shared key generation protocol. In addition we try to minimise the amount of interaction required between the parties and we eliminate the need for a share refreshing stage of the protocol.

2 Applications of the protocol

The reason why threshold decryption/signing is very useful in practice is because not only does it provide secrecy and reliability but also flexibility. In addition, the property of sharing the secret is ideally suited to applications in which a group of mutually suspicious individuals with conflicting interests must cooperate, for example, in an electronic voting system or any gambling games that will be explained in more detail later on in this section.

2.1 Digital Signature

One of many typical examples is in digital signature. Assuming there are 10 managers in a bank, if every manager has his or her own copy of the secret key then the system can be easily suffers from single point of failure or misuse due to compromise and machine break down. In contrast, if a valid signature requires the signatures of all ten managers in the company then it will be very secure but might not be convenient to use in real life. Therefore the best solution for this problem might be that as long as a document is digitally signed by any 5 or more out of 10 managers then it will be valid and that is exactly what a (5,10) threshold signing scheme tries to achieve. In addition, if an adversary wants to obtain a signature on a message, he must compromise at least 5 people in the scheme and that is a much harder thing to do compared to a traditional public protocol.

2.2 Distributed Certificate Authority

As another example one can consider a PKI, public key infrastructure, implementation where the CA, certificate authority, is distributed amongst many sites, so as to avoid a single point of failure. But the system must be robust against occasional system downtime or network problems. Hence, distributing the RSA signing key amongst a number of boxes which implement the CA in a threshold manner one can obtained security against a single point of failure and robustness against errors.

2.3 Electronic voting system and Internet card game protocols

Both of these two applications share many remarkable things in common, and they are also used as the bench mark to test new ideas in cryptography and information security. In an electronic election, as Cramer described in [10], where voters can vote on line, it is really crucial to make sure that no body, party or organisation can find out the final result by doing a decryption before the very end of the election. In traditional way, the secret resides or is under the control of a trusted third party. As a consequence, the system is susceptible to single point of failure because if the trusted dealer is compromised (with a very low probability) then that is the end of the election. On the other hand, sharing the secret key by multiple parties, each holds a share of the secret, can guarantee that decryption is done if and only if all parties agree to do it and therefore the scheme can give us a much higher level of security. The same thing can be applied to Internet Card Game protocols, such as the one presented by

Barnett and Smart in [1] where all players must agree to decrypt a card in order to do so.

2.4 Identification Scheme

There are many proposed identification schemes notably, Guillou-QuisQuater [23], Fiat-Shamir [18] and their extended and modified versions in [17, 20, 26, 27] that are achieved by asking a dealer to generate RSA moduli N . Clearly, the dealer must be trusted to generate N correctly as well as keeping them secret to all parties and in real life this is a too strong assumption. The protocol implemented by me will be able to eliminate the need of the trusted party since all parties can generate the moduli N by themselves and not knowing about the secret in the mean time.

3 Cryptography Techniques

3.1 Hard Problem

There are two hard problems, factoring big number and discrete logarithm, which we are going to describe in this section. Everything in this protocol is based on the assumption that these two problems are computationally infeasible to be solved in polynomial time.

3.1.1 Factoring Problem

The problem states that given a big number, about 1000 bits, it is computationally infeasible to factorise the number into prime factors. The best known algorithm has exponential complexity in term number of bits of the number.

For example: If $N = pq$ where p and q are big prime numbers (500 bits) then it is hard to find p and q given the value of N . This problem will be the basis of security of RSA encryption and decryption scheme.

3.1.2 Discrete Logarithm Problem

Given a big number N , of size 1000 bits, and y, g in the interval $[1, \dots, (N-1)]$ where $\gcd(g, N) = 1$. It is hard to find x that satisfies the equation:

$$y = g^x \pmod{N}$$

This problem will be the basis of threshold decryption and the signing scheme implemented in this project.

3.2 Euler Theorem and Fermat Primality test

Euler theorem is probably one of the most important theorem used in public cryptography.

3.2.1 Euler Theorem

Given a number $N = p_1^{a_1} p_2^{a_2} \dots p_n^{a_n}$ and p_1, \dots, p_n are prime numbers, there are exact $\phi(N)$ numbers between 1 and $(N-1)$ that are co-prime to N , where:

$$\phi(N) = \prod_{i=1}^n (p_i^{a_i-1} (p_i - 1))$$

3.2.2 RSA case

When $N = pq$, we have $\phi(N) = (p-1)(q-1)$ and for any e co-prime to N it satisfies that:

$$e^{\phi(N)} = 1 \pmod{N}$$

An interesting direct result of this case is that it gives us a way to test whether or not a number is a product of two large prime numbers indirectly. The method will be explained in more detail in section 5.2.4 as it actually forms the primality testing stage of this protocol.

3.2.3 Fermat Primality Test

When N is a prime number, $\phi(N) = N - 1$ and for any e co-prime to N , we have $e^{\phi(N)} = 1 \pmod{N}$.

This theorem also gives us a way to test whether N is prime or not. The test can be implemented very fast in practice by using binary power method (or Indian power) in conjunction with Montgomery multiplication. Unfortunately, there are still composite numbers that output 1 when being tested, but with a very low probability. This type of number is called Carmichael and there are infinite number of them.

3.3 RSA Algorithm

After the invention of public cryptography by Diffie and Hellman [15] in the 70s, the first practical scheme was due to Ron Rivest, Adi Shamir and Adleman in their jointed paper [31], and it has been the most popular and successful public algorithm since then. Given $N = p.q$ where p and q are large prime numbers then:

- The public components are moduli N and the encryption exponent e where e is co-prime to N .
- The private components are prime factors of N , which are p and q , and the decryption exponent d where:
$$e.d = 1 \pmod{\phi(N)}$$
$$e.d = 1 \pmod{(p-1)(q-1)} \quad \text{due to Euler theorem.}$$
- To encrypt a message m that is smaller than N , the cipher text, c , is computed as follows:

$$c = m^e \pmod{N}$$

- To decrypt a cipher text c , the message, m , is determined by computing:

$$m = c^d \pmod{N}$$

3.4 Shared RSA Threshold Decryption

Similarly to traditional RSA, in shared RSA scheme, we also have $N = pq$, and the public components are moduli N and the encryption exponent e where e is co-prime to N .

However, decryption is more complicated as there are more parties who get involved in the scheme. Assuming there are n parties and the prime factors of N remain unknown to every person. Each party P_i now only knows the tuple $\langle p_i, q_i, d_i \rangle$ and keeps it secret to any other parties, and they are also required to satisfy the four following conditions:

1. p is an unknown big prime number and $p = p_1 + p_2 + \dots + p_n = \sum_{i=1}^n p_i$.
2. q is an unknown big prime number and $q = q_1 + q_2 + \dots + q_n = \sum_{i=1}^n q_i$.
3. The unknown decryption exponent $d = d_1 + d_2 + \dots + d_n = \sum_{i=1}^n d_i$.

4. And $ed = 1 \pmod{\phi(N)}$.

The reader might wonder how can we generate such a scheme like this without the help of a trusted third party and still keep p , q , and d secret to everyone in the world? The answer is that such scheme is exactly what we want to achieve with the n -out-of- n shared key generation protocol in first half of this project. For now, I shall assume that we have achieved the properties and I am going to show you two different ways to encrypt a message and decrypt a cipher text based on Discrete Logarithm Problem and RSA Paillier presented in [28].

3.4.1 Discrete Logarithm Approach

As the name implies, this approach is based on the difficulty of Discrete Logarithm problem.

- **Encryption:** is identical to RSA: $c = m^e \pmod{N}$.
- **Decryption:** each party P_i computes $m_i = c^{d_i} \pmod{N}$ and then publishes m_i to all other parties. As it is hard to find d_i given m_i and c (discrete logarithm problem) so that d_i still remains secret to party P_i after decryption. Now, each party knows all m_i for $i = 1, \dots, n$ and therefore can recover m from the following formulae:

$$m = \prod_{i=1}^n m_i$$

Proof:

$$\begin{aligned} m &= m_1 m_2 \dots m_n && \pmod{N} \\ m &= c^{d_1} c^{d_2} \dots c^{d_n} = c^{\sum_{i=1}^n d_i} && \pmod{N} \\ m &= c^d = m^{ed} && \pmod{N} \\ m &= m && \pmod{N} \end{aligned}$$

An alternate way to do encryption and decryption can be based on Elgamal algorithm [16].

3.4.2 RSA Paillier Approach

This scheme has been recently proposed in [28] for a single party and extended to deal with multiple participants in another paper of Barnett and Smart [1]. All the parties generate a share of $\phi = (p-1)(q-1)$ by setting:

$$x_i = \begin{cases} n - (p_1 + q_1) + 1 & \text{If } i = 1 \\ -(p_i + q_i) & \text{If } i > 1 \end{cases}$$

Note that $\phi = \sum_{i=1}^n x_i$. The parties then commit to the value x_i by publishing $h_i = g^{x_i} \pmod{N^2}$ where $g = N + 1$. They then set publicly:

$$h = \prod_{i=1}^n h_i - 1 \pmod{N^2} = g^\phi - 1 \pmod{N^2}$$

- **Encryption:** To encrypt a message m , a user chooses a random number r in the range $[1, \dots, N]$ and $\gcd(r, N) = 1$. The cipher text is computed as follows:

$$c = g^{m r^N} \pmod{N^2}$$

- **Decryption:** each party P_i computes $m_i = c^{x_i} \pmod{N^2}$ and then publishes m_i to all other parties. Each party knows all m_i for $i = 1, \dots, n$ and therefore can recover message m by computing:

$$m = \frac{1}{h} \left(\prod_{i=1}^n m_i - 1 \pmod{N^2} \right) \pmod{N}$$

4 Multi-party Computation Protocols

A number of multi-party computation schemes that are used throughout the protocol have been implemented in this project. They are all described and analysed in this section.

4.1 Shamir Secret Sharing Scheme and Lagrange Coefficient

Shamir Secret Sharing scheme gives us a mechanism where a secret is split into n pieces and any $w \leq n$ shares when they are gathered together will be able to reconstruct the secret. Also note that after reconstruction, the secret becomes known to all parties.

A trusted dealer has a secret s and chooses a large prime number $M > n$, the number of servers. Unless otherwise stated, all arithmetic operations are done modulo M .

- **Step 1:** Let l is some number smaller than or equal to $(n - 1)$. The dealer picks a random degree l polynomial $f \in Z_M[x]$ satisfying $f(0) = s$.

$$f(x) = a_l x^l + \dots + a_i x^i + \dots + a_1 x + s \text{ where } a_i \in Z_P \text{ for all } i = 1, \dots, n$$

- **Step 2:** For all $i = 1, \dots, n$, the dealer computes $y_i = f(i)$. The dealer then privately sends f_i to server i for all $i = 1, \dots, n$.
- **Step 3:** To reconstruct the secret s , any w servers come together. They can recover the secret by solving the system of linear equations, provided that $w > l$.

$$\begin{array}{r} y_1 = a_l x_1^l + \dots + a_i x_1^i + \dots + a_1 x_1 + s \\ \cdot \quad \cdot \quad \quad \cdot \quad \quad \cdot \\ \cdot \quad \cdot \quad \quad \cdot \quad \quad \cdot \\ y_w = a_l x_w^l + \dots + a_i x_w^i + \dots + a_1 x_w + s \end{array}$$

As w parties all know x_i for all $i = 1, \dots, w$ so that they also can compute x_i^j for all $i = 1, \dots, w$ and $j = 1, \dots, l$.

Whilst the system can be solved by using either polynomial interpolation or Gaussian elimination, a more efficient method, Lagrange Coefficient, is always used in practice. The reason is because we are only interested in the secret $s = f(0)$ and not any other coefficients in the system. The computation is done as below:

- We first compute: $b_j = \prod_{1 \leq h \leq w, h \neq j} \frac{x_h}{x_h - x_j}$, for all $j = 1, \dots, w$.
- Then for j between 1 and w : $s_j = b_j y_j$.
- Finally the secret s is the sum of all additive shares s_j :

$$s = f(0) = \sum_{j=1}^w s_j = \sum_{j=1}^w b_j y_j$$

4.1.1 Modulo non-prime

In the previous case, M is chosen as a prime number. In fact, the scheme still works as long as M is a composite and does not have any prime factor that is smaller than or equal to the number of servers. The reader shall see this case is applied in the Distributed Sieving stage of the protocol.

4.1.2 Sharing the final outcome

Note that the additive share $s_j = b_j y_j = (\prod_{1 \leq h \leq w, h \neq j} \frac{x_h}{x_h - x_j}) y_j$ can be computed privately by party P_j . Therefore, the secret s can be additively shared amongst the servers rather than becomes publicly available. As a result, the servers do not perform the above step 3 of Shamir Secret Sharing scheme any more.

4.2 Benaloh Protocol

Suppose each of the n parties has a secret share, s_i . They wish to compute $s = \sum_{i=1}^n s_i \pmod{M}$ without revealing any further information about their secret shares modulo M . This can be done by Benaloh's protocol developed in [3], which is $(n - 2)$ private, and it works as follows:

- **Step 1:** Each party P_i picks n random elements $s_{i,j}$ for $j = 1, \dots, n$ such that $s_i = \sum_{j=1}^n s_{i,j} \pmod{M}$. For example, party P_i chooses $(n - 1)$ random numbers, $s_{i,1}, \dots, s_{i,(n-1)}$ and sets:

$$s_{i,n} = (s_i - \sum_{j=1}^{n-1} s_{i,j}) \pmod{M}$$

- **Step 2:** Each party P_i privately sends $s_{i,j}$ to party j for $j = 1, \dots, n$.
- **Step 3:** Each party P_j receives n shares $s_{i,j}$ for $i = 1, \dots, n$ and then computes:

$$\hat{s}_j = \sum_{i=1}^n s_{i,j} \pmod{M}$$

And then broadcasts \hat{s}_j to all other parties.

- **Step 4:** Each party receives $\hat{s}_1, \dots, \hat{s}_n$ and computes the required sum as follows:

$$s = \sum_{i=1}^n \hat{s}_i \pmod{M}$$

Proof:

$$\begin{aligned} s &= \sum_{i=1}^n s_i && \pmod{M} \\ s &= \sum_{i=1}^n (\sum_{j=1}^n s_{i,j}) && \pmod{M} \\ s &= \sum_{j=1}^n (\sum_{i=1}^n s_{i,j}) && \pmod{M} \\ s &= \sum_{j=1}^n \hat{s}_j && \pmod{M} \end{aligned}$$

The scheme is $(n - 2)$ private as if $(n - 1)$ parties collude then they will be able to find out the final share of the single left party modulo M .

4.3 BGW Protocol

The reader can skip this section and come back to it later when she has got to the Distributed Sieving and N Computation stages of the protocol.

This protocol was originally invented by Ben-Or, Goldwasser and Wigderson in [4]. The following protocol is a simplified version of it. This protocol makes use of Shamir Secret Sharing scheme, and therefore the reader is strongly recommended to understand Shamir's scheme first before attempting to read this one.

Suppose each one of n parties has p_i, q_i . They wish to compute:

$$N = \left(\sum_{i=1}^n p_i \right) \left(\sum_{j=1}^n p_j \right)$$

Without revealing any further information about their secret shares, p_i and q_i . That means at the end of the protocol, N is made public but p_i and q_i are known to only party P_i . In addition, $p = \sum_{i=1}^n p_i$ and $q = \sum_{j=1}^n p_j$ are also unknown to all parties.

Let M be a big number, $M > N$ and M does not have any prime factor that is smaller or equal to number of parties. Unless otherwise stated, all arithmetic operations are done modulo M .

- **Step 1:** Let $l = \lfloor \frac{n-1}{2} \rfloor$, each party P_i picks $4l$ random secret coefficients: $a_{i,1}, \dots, a_{i,l}$ and $b_{i,1}, \dots, b_{i,l}$, and $c_{i,1}, \dots, c_{i,l}, c_{i,(l+1)}, \dots, c_{i,2l}$ that form the three polynomials:

$$\begin{aligned} f_i(x) &= p_i + \sum_{j=1}^l a_{i,j} x^j \\ g_i(x) &= q_i + \sum_{j=1}^l b_{i,j} x^j \\ h_i(x) &= \sum_{j=1}^{2l} c_{i,j} x^j \end{aligned}$$

and therefore we have for all $i = 1, \dots, n$: $f_i(0) = p_i$, $g_i(0) = q_i$ and $h_i(0) = 0$. They compute the followings for all $j = 1, \dots, n$:

$$\begin{aligned} f_{i,j} &= f_i(j) \\ g_{i,j} &= g_i(j) \\ h_{i,j} &= h_i(j) \end{aligned}$$

- **Step 2:** Each party P_i sends tuple $\langle f_{i,j}, g_{i,j}, h_{i,j} \rangle$ to party P_j privately for all $j = 1, \dots, n$.
- **Step 3:** Each party j receives n tuples $\langle f_{i,j}, g_{i,j}, h_{i,j} \rangle$ for $i = 1, \dots, n$ and then computes:

$$\begin{aligned} N_j &= \left(\sum_{i=1}^n f_{i,j} \right) \left(\sum_{i=1}^n g_{i,j} \right) + \sum_{i=1}^n h_{i,j} \\ &= \left(\sum_{i=1}^n f_i(j) \right) \left(\sum_{i=1}^n g_i(j) \right) + \sum_{i=1}^n h_i(j) \end{aligned}$$

Party P_j publishes N_j to every one in the scheme.

- **Step 4:** At this point of the protocol, each party receives N_1, \dots, N_n from all other parties. Let:

$$N(x) = F(x)G(x) + H(x)$$

Where:

- $F(x) = \sum_{i=1}^n f_i(x)$ and the order of function $F(x)$ is equal to the order of function $f_i(x)$ that is l .
- $G(x) = \sum_{i=1}^n g_i(x)$ and the order of function $G(x)$ is equal to the order of function $g_i(x)$ that is l .
- $H(x) = \sum_{i=1}^n H_i(x)$ and the order of function $H(x)$ is equal to the order of function $h_i(x)$ that is $2l$.

That means the order of function $N(x)$ will be $2l$. As $l = \lfloor \frac{n-1}{2} \rfloor$ so $n \geq 2l$ and therefore knowing n values of $N(x)$ for different non-zero values of x can help each party to find $N = N(0)$ by Lagrange Coefficients as described in section 4.1.

Proof:

$$\begin{aligned} N &= N(0) \\ &= F(0) * G(0) + H(0) \\ &= (\sum_{i=1}^n f_i(0))(\sum_{i=1}^n g_i(0)) + \sum_{i=1}^n h_i(0) \\ &= (\sum_{i=1}^n p_i)(\sum_{i=1}^n q_i) + \sum_{i=1}^n 0 \\ &= (\sum_{i=1}^n p_i)(\sum_{i=1}^n q_i) \end{aligned}$$

4.3.1 Privacy

The protocol is $l = \lfloor \frac{n-1}{2} \rfloor$ private as all the polynomials f_i and g_i for $i = 1, \dots, n$ have order l , so even if $\lfloor \frac{n-1}{2} \rfloor$ are dishonest, not any coefficient of any polynomial f_i, g_i and h_i is revealed.

4.3.2 Sharing the final outcome

Similarly to Shamir's scheme, the final outcome N can be additively shared amongst the n parties. As a result, the parties do not need to perform Step 4 any more but instead each party only needs to compute:

$$n_j = (\prod_{1 \leq h \leq n, h \neq j} \frac{x_h}{x_h - x_j}) N_j$$

However, N remains unknown and $N = \sum_{j=1}^n n_j$.

4.3.3 Extension

Note that it is easy enough to extend the protocol further. If each party i of n parties has p_i, q_i, t_i , then they can compute:

$$N = (\sum_{j=1}^n p_j)(\sum_{j=1}^n q_j)(\sum_{j=1}^n t_j)$$

by using similar method as described above. However, the order of some of the polynomials are $\lfloor \frac{n-1}{3} \rfloor$ and therefore the scheme is only $\lfloor \frac{n-1}{3} \rfloor$ private.

5 Shared RSA Secret Keys Generation Protocol, an n-out-of-n Threshold Scheme

5.1 Problem Definition and Notation

In this section, I would like to give the reader a high level overview of the protocol before going into detailed discussion of each stage of the protocol.

What the protocol wants to achieve is that multiples parties, say n , will come together to generate a moduli N and make N and the encryption exponent, e public. No body knows the prime factors of N but everyone is convinced that N is a product of two large prime numbers. The scheme is n -out-of- n threshold scheme, and that means decryption requires the presence of all parties because each party keeps an additive share, d_i of the decryption exponent, d . As a result, this scheme allows a new Threshold Signing Scheme, (k, n) , to be added later on in this project. Also note that the value of d is unknown to all parties and after any number of decryptions. Throughout the protocol, a trusted third party is not required and all stages in the protocol need the contribution of all individual parties.

- **Picking candidates and Distributed Sieving.**

1. Each party i picks two secret numbers p_i and q_i .
2. All parties determine whether or not, the sums $p = \sum_{i=1}^n p_i$ and $q = \sum_{i=1}^n q_i$ are not divisible by any prime number between 0 and some bound, $B1$, by using distributed sieving method. If they are, the protocol will come back to part (1) of this stage. Note that the values of p and q remain totally unknown to all parties.

- **N Computation:** All parties come together to implement the distributed computation of

$$N = \left(\sum_{i=1}^n p_i \right) \left(\sum_{i=1}^n q_i \right)$$

N is public but p_1, \dots, p_n and q_1, \dots, q_n remain private.

- **Trial Division:** This stage is done to make sure that N is not divisible by any number between $B1$ and $B2$, agreed by all parties.
- **Primality Test:** Extended Fermat Primality test is used to determine whether N is a product of two prime numbers. If the test failed, then the protocol would come back to the first stage, Picking candidates and Distributed Sieving.
- **Private Key Generation:** Having computed N and a public encryption exponent e , each party now computes its own private additive share, d_i , of the decryption key, d . So we have:

$$d = \sum_{i=1}^n d_i + x \text{ and } de = 1 \pmod{N}$$

Note that x is not known at the moment.

- **Trial Decryption:** As x can be proved to be in the range $[0, n]$ and therefore we can easily determine it by doing a trial decryption. This part is also responsible for eliminating candidates of N that passed the above Primality test but actually are not a product of two big prime numbers.

Thus, the main advantages of the scheme over previous protocols are that it does not require a trusted third party, and it still can generate the public/private keys pairs efficiently. The main disadvantage is that the protocol is fully interactive in every single stage as all parties need to know the identities of each other.

5.2 Scheme Definition

5.2.1 Picking candidates and Distributed Sieving:

The purpose of distributed sieving is to make sure that the sum of all parties' shares, $p = \sum_{i=1}^n p_i$ and $q = \sum_{i=1}^n q_i$ are not divisible by any prime number between 2 and some bound $B1$. In order to achieve this goal, firstly all participants must agree on bound $B1$ and compute M as the product of all prime numbers between n , the number of parties, and $B1$.

$$M = \prod_{j=1}^t m_j$$

where $n < m_1 < m_2 < \dots < m_t \leq B1$ and m_j is prime for all $j = 1, \dots, t$.

Then each party P_i picks a random secret integer a_i relatively prime to M , so that their product across all parties is also relatively prime to M .

$$\left. \begin{array}{l} a \\ \gcd(a_i, M) = 1 \end{array} \right\} \Rightarrow \gcd(a, M) = 1$$

However, what we want to have is that each party P_i keeps p_i secret and

$$a = a_1 \cdots a_n = p_1 + \cdots + p_n$$

That is equivalent to converting a multiplicative sharing (a_1, \dots, a_n) into an additive sharing (p_1, \dots, p_n) and it is done iteratively as follows:

- **Step 1:** Initially we have:

$$\left\{ \begin{array}{ll} u_{1,i} = a_1 & \text{and } v_{1,i} = 1 \text{ for } i = 1 \\ u_{1,i} = 0 & \text{and } v_{1,i} = 0 \text{ for } i \neq 1 \end{array} \right.$$

Note that party P_i keeps a_i secret. All parties run the algorithm of section 4.3 on the input:

$$\begin{aligned} a_1 &= (a_1 + 0 + \cdots + 0)(1 + 0 + \cdots + 0) \pmod{M} \\ &= (u_{1,1} + \cdots + u_{1,n})(v_{1,1} + \cdots + v_{1,n}) \pmod{M} \end{aligned}$$

The algorithm produces the following additive sharing:

$$a_1 = u_{2,1} + \cdots + u_{2,n} \pmod{M}$$

Also note that the value of a_1 at the end of this step still remains secret to party P_1 . However, now it is additively shared across the n parties as each of them knows $u_{2,i}$ and keeps it private.

- **Step i :** (for $(n+1) > i > 1$)

From the $(i-1)$ previous iterations, all parties know that:

$$a_1 \cdots a_{i-1} = u_{i,1} + \cdots + u_{i,n} \pmod{M}$$

They start this iteration with the below assignment.

$$\begin{cases} v_{i,j} = a_{i+1} & \text{if } j = i+1 \\ v_{i,j} = 0 & \text{if } j \neq i+1 \end{cases}$$

Again, the parties run the algorithm of section 4.3 on input:

$$\begin{aligned} a_1 \cdots a_{i-1} a_i &= (u_{i,1} + \cdots + u_{i,n})(0 + 0 + \cdots + a_i + \cdots + 0) \pmod{M} \\ &= (u_{i,1} + \cdots + u_{i,n})(v_{i,1} + \cdots + v_{i,n}) \pmod{M} \end{aligned}$$

The algorithm produces the following additive sharing:

$$a_1 \cdots a_{i-1} a_i = u_{(i+1),1} + \cdots + u_{(i+1),n} \pmod{M}$$

At the end of the algorithm, step n , all parties have:

$$a = a_1 \cdots a_n = u_{(n+1),1} + \cdots + u_{(n+1),n} \pmod{M}$$

And therefore they have achieved the required additive sharing of a by replacing $u_{(n+1),i}$ by p_i for all $i = 1, \dots, n$. The same method can be used to construct the share of the other prime factor q .

5.2.2 Distributed Computation of N :

Recall that each party i keeps its shares p_i and q_i secret. Now, they want to compute $N = pq = (\sum_{i=1}^n p_i)(\sum_{i=1}^n q_i)$. N is made public, however no partial information about any secret shares is revealed. This is exactly what the protocol BGH in section 4.3 can do. Therefore, all parties agree on a big prime number $M > N$ and then run the BGH protocol on the following input:

$$N = (p_1 + \cdots + p_n)(q_1 + \cdots + q_n) \pmod{M}$$

5.2.3 Parallel Trial Division:

Once, the parties have computed the public moduli N , they now want to test whether N is not divisible by any prime number between two bounds, B_1 and B_2 where $B_2 > B_1$ before invoking the expensive primality test described in the next section. We can store the list of all these prime numbers as an array s_1, s_2, \dots, s_t into each party permanently. So

$$B_1 < s_1 < \dots < s_t < B_2$$

In order to speed up the process by factor of n , trial division is done in parallel. So what it means is that party i is in charge of testing that N is not divisible by any prime number s_j in the above list for all $j = i \pmod{n}$ and $1 \leq j \leq t$. As a result of n -fold increase in speed, we can use a large trial division bound, B_2 which then increases the effectiveness of trial division.

5.2.4 Load Balance Primality Test:

In this stage, all parties need to determine whether N is a product of two large prime numbers or not. This can be done by using the extended Fermat primality test explained in section 3.2.2. This requires the cooperation of all parties as follows:

- **Step 1:** All parties agree on a random number g , where $\gcd(g, N) = 1$.
- **Step 2:** Party i th computes:

$$v_i = \begin{cases} g^{N-p_1-q_1+1} \pmod{N} & \text{If } i = 1 \\ g^{p_i+q_i} \pmod{N} & \text{If } i > 1 \end{cases}$$

and then publishes v_i . Note that because of difficulty of solving Discrete Logarithm Problem, an eavesdropper cannot find out the value of p_i and q_i for $i = 1, \dots, n$.

- **Step 3:** Each party receives all v_i for $i = 1, \dots, n$ and checks the equality:

$$v_1 = \prod_{i=2}^n v_i$$

If the equality holds then all parties are nearly convinced that N is a product of two large prime numbers.

Proof:

$$\begin{aligned} g^{(p-1)(q-1)} &= g^{pq-(p+q)+1} && \pmod{N} \\ &= g^{N-(\sum_{i=1}^n p_i + \sum_{i=1}^n q_i)+1} && \pmod{N} \\ &= g^{N-p_1-q_1+1} \prod_{i=2}^n g^{-(p_i+q_i)} && \pmod{N} \\ &= \frac{v_1}{\prod_{i=2}^n v_i} && \pmod{N} \end{aligned}$$

So if both p and q are prime then

$$g^{(p-1)(q-1)} = g^{\phi(N)} = 1 \pmod{N}$$

Therefore:

$$v_1 = \prod_{i=0}^n v_i$$

Unfortunately, there are still cases where p and q are not both prime but $g^{(p-1)(q-1)} = 1 \pmod{pq}$, however, this only happens very rarely. Furthermore, all the cases will be eliminated by trial decryption done at the very end of this protocol.

Load balance optimisation: In practice, we have to carry out this test with many different candidates for N to make sure that at least one of them is correct. As the reader might notice that the length of $(N - p_1 - q_1 + 1)$ is twice as long as $(p_i + q_i)$ and therefore it takes party 1 a longer time to finish step 2 of the primality test.

So it makes sense to assign the task equally to all parties when the number of N is large to get a factor of 2 speed up. For example: if there are $n * t$ different N s, then:

- The first party will carry this task for the first t candidates of N .
- The second party will carry this task for the next t candidates of N and so on.
- The n^{th} party will carry this task for the last t candidates.

5.2.5 Private Key Generation:

At this point in the scheme, N has been computed and all parties agree on the public encryption exponent, e . Now, they want to find their additive shares of the decryption exponent, d . Another word, each party i keeps d_i secret and:

$$\begin{cases} d &= \sum_{i=1}^n d_i & \pmod{N} \\ ed &= 1 & \pmod{\phi(N)} \end{cases}$$

No body knows $\phi(N)$ and d . This can be achieved by the following algorithm:

- **Step 1:** All parties generate a share of $\phi(N)$ bet setting:

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

$$\text{So } \phi(N) = (p - 1)(q - 1) = N - \sum_{i=1}^n p_i - \sum_{i=1}^n q_i + 1 = \sum_{i=1}^n \phi_i$$

- **Step 2:** By using Benaloh's protocol described in section 4.2 on input $\phi_1, \phi_2, \dots, \phi_n$. All parties can find:

$$\psi = \phi(N) \pmod{e} = \sum_{i=1}^n \phi_i \pmod{e} \text{ and } \psi^{-1} \pmod{e}$$

As the public encryption exponent e is small and therefore only a few bits are leaked in this stage of the protocol.

- **Step 3:** All parties generate a share of the decryption exponent, d , by setting:

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

So at the end of this step:

$$\sum_{i=1}^n d_i = \lfloor \frac{1 - \phi_1 \psi^{-1}}{e} \rfloor + \sum_{i=2}^n \lfloor \frac{-\phi_i \psi^{-1}}{e} \rfloor$$

Lifting up the lower bound symbols on the right hand side, we shall need to introduce a dummy variable x that is in the short interval $[0, n]$, and we get:

$$\begin{aligned} \sum_{i=1}^n d_i + x &= \frac{1 - \psi^{-1} \sum_{i=1}^n \phi_i}{e} \\ \sum_{i=1}^n d_i + x &= \frac{1 - \psi^{-1} \phi(N)}{e} \end{aligned}$$

Multiplying both sides by e :

$$\begin{aligned} (\sum_{i=1}^n d_i + x)e &= 1 - \psi^{-1} \phi(N) \\ (\sum_{i=1}^n d_i + x)e &= 1 \pmod{\phi(N)} \end{aligned}$$

The decryption exponent d is equal to $(\sum_{i=1}^n d_i + x)$ where d_i is kept privately by party i^{th} . The unknown x can be easily found by doing a trial decryption done in the next section.

5.2.6 Trial Decryption:

The purpose of trial decryption is to find the value of x left unknown in the previous section and more importantly is to eliminate all incorrect candidates of N that were able to get through the primality test. The below algorithm is done multiple times on different messages, m .

- **Step 1:** All parties agree on a message m and then each party P_i computes:

$$m_i = (m^{d_i})^e \pmod{N}$$

He or she then sends m_i to party P_1 .

- **Step 2:** Party P_1 receives m_i for $i = 1, \dots, n$ and computes their product:

$$m' = m_1 m_2 \cdots m_n \pmod{N}$$

He then tries out all values of x from 0 to $n-1$ to find the one that satisfies this equation:

$$m = m' (m^x)^e \pmod{N}$$

After running the above algorithm a number of time and the values of x are always the same at the end of every iteration (on different message, m) then all parties can conclude that they have found a correct solution. Party P_1 now knows the value of x so will adjust its additive share, d_1 as follows:

$$d_1 = d_1 + x$$

The value of x can be made public at the end of this step.

5.3 Discussion of the Above Scheme

5.3.1 Privacy

The first drawback of this protocol is about privacy as it is only $\lfloor \frac{n-1}{2} \rfloor$ private where n is the number of participants. So that the attacker only can recover the secret key if it compromises at least $\lfloor \frac{n-1}{2} \rfloor + 1$ parties. In contrast, even if $\lfloor \frac{n-1}{2} \rfloor$ parties corrupt, no partial information about the secret key is revealed.

The reason for this drawback is because in both Distributed Sieving and N Computation stages of the protocol, Shamir Secret Sharing scheme has been used. The lowest order of a polynomial is $\lfloor \frac{n-1}{2} \rfloor$ and therefore if $\lfloor \frac{n-1}{2} \rfloor + 1$ parties are dishonest then they can find all coefficients of the polynomial and can reconstruct the secret keys.

5.3.2 Smallest number of parties

As the scheme is only $\lfloor \frac{n-1}{2} \rfloor$ private so n must be greater or equal to three because if $n = 2$ then

$$\lfloor \frac{n-1}{2} \rfloor + 1 = \lfloor \frac{2-1}{2} \rfloor + 1 = 1$$

Therefore either party can reconstruct the decryption key which is not what we want to achieve anymore.

5.3.3 Generating Prime number

The main reason that makes this protocol run significantly more slowly than traditional RSA scheme is because both the prime factors of RSA moduli N must be generated simultaneously as we do not know any way to test whether or not $P = \sum_{i=0}^{n-1} p_i$ is a prime number where P remains unknown to all participants and each party keeps its share p_i secret. Whereas we can use Fermat primality test in conjunction with Discrete Logarithm to test whether N is a product of two prime numbers or not as explained in section 5.2.4.

In another word, if it takes approximately n trials to generate a prime number of n bits then it will take about n^2 trials to find 2 prime numbers, each n bits, in the same time. Fortunately, due to Distributed Sieving, things are not so bad, the reader can see more detail about it in section 5.2.2.

This section directly leads us to an unsolved problem that is whether or not we can construct a very big prime number that is additively shared by multiple parties. This algorithm will need a way to test whether the sum of shares across all parties is prime or not with a linear complexity in term of the bit length of that prime number. If one could find such an algorithm then we would be able to improve the storage complexity from quadratic to linear as well.

5.3.4 Complexity

Due to generating prime number problem explained in the previous section, the protocol's run-time has complexity of $O(l^2)$ where l is the number of bits of prime factor P , and Q of RSA moduli N .

The storage complexity is $O(n * l^2)$. There are two reasons why I come up with this formula, firstly there are l^2 trial as we need to find 2 large primes concurrently. Secondly, I use Shamir Secret Sharing scheme, and the highest order of a polynomial is equal to the number of parties in the protocol, which is n , therefore I need to store at least n coefficients for each trial.

6 Partially Interactive Threshold RSA Signatures

6.1 Problem Definition and Notation

We assume that a set of n players wishes to generate a number of threshold RSA signatures. Using a scheme, I have described above, the n players can generate a shared RSA moduli N , a public exponent e and n shares d_i of the secret exponent d , such that

$$d = d_1 + d_2 + \dots + d_n.$$

This shared RSA key generation protocol can be executed without the need for a trusted dealer.

The parties now wish to use these shares so as to generate a threshold RSA signature scheme, with threshold value k . The resulting signature should be compatible with existing hash-and-sign RSA signatures such as RSA-FDH [2]. This means that we want any k parties to come together so as to be able to sign a document. We let $I = \{t_1, \dots, t_k\} \subset \{1, \dots, n\}$ denote the set of parties who wish to come together to sign the document and $I' = \{1, \dots, n\} \setminus I = \{t_{k+1}, \dots, t_n\}$ denote the other parties. There are essentially two existing ways of doing this, both with disadvantages.

In [30] Rabin gives a scheme which does not require a trusted dealer, as we require, but which works by the k signing parties, when signing a document, reconstructing the $n - k$ secrets d_{t_i} for $i = k + 1, \dots, n$. This means that after signing one message, if a different subset is going to sign for the second message, one needs to re-key in some way. Hence, Rabin defines a share refreshing protocol which occurs after a signing operation. The signing stage requires the interaction of all k signing parties, so as to reconstruct the $n - k$ missing secrets, and the share refreshing protocol requires the interaction of all n parties.

In [33] Shoup gives a scheme which is completely non-interactive in that each party signs the message independently and a separate, public, share combining algorithm is used to combine k of the signature shares into a valid full RSA signature. There is no need for a share refreshing protocol and once the scheme is set up there is no need for any of the parties to interact. However, the set-up phase of the Shoup scheme requires a trusted dealer who knows the factors of the RSA moduli N .

Our scheme gives a threshold RSA scheme which combines some of the advantages of the previous schemes, and tries to reduce their disadvantages. In particular we define the following algorithms, with the following properties:

- **Dealing Algorithm:** An interactive protocol amongst the n players. Each player has an input d_i , which is their share of the unknown RSA private key d . At the end of this protocol the players agree on a threshold value k and some global public information S . In addition each player also obtains a public/private share (P_i, S_i) of the data needed to implement the threshold signature scheme. We note that this stage does not require a trusted third party in our scheme.
- **Subset Presigning Algorithm:** This is an interactive protocol amongst k members $I = \{t_1, \dots, t_k\}$ of the n parties. The protocol results in public

data D_I which is used by the share combining algorithm to generate a valid full RSA signature from the signature shares. The protocol results in each of the k parties holding some secret information S_{I,t_i} which depends on the subset I . This protocol is interactive, but only needs to be run once for each subset I .

- **Signature Share Generation Algorithm:** This algorithm takes as input a subset I as above, the secret information S_{I,t_i} and a message m . The result is a partial signature σ_{I,t_i} on the message m .
- **Signature Share Verification Algorithm:** This takes as input a signature share σ_{I,t_i} on the message m and verifies that it is validly formed using the public information D_I and P_{t_i} .
- **Share Combining Algorithm:** This takes as input the public information S , P_i and D_I , plus a message m and the partial signature shares σ_{I,t_i} for all $t_i \in I$, and then produces a valid full RSA signature σ . Or returns fail if one of the signature shares is invalid.

Hence, the main advantages of our scheme over those of Rabin and Shoup are that we do not require a trusted dealer (as Shoup's scheme does) and we do not require to rekey or interact once the Subset Presigning algorithm has been implemented for a given subset I . The main disadvantage is that the signature share generation algorithm needs to know which subset of shares are going to be combined later on.

6.2 Scheme Definition

In this section we describe our scheme and justify that it works.

6.2.1 Dealing Algorithm:

The parties $i = 1, \dots, n$ first agree on a number of parameters.

- A prime number M where $M > N$.
- The threshold value k where $1 < k < n$.
- An element g of high order in \mathbb{Z}_N^* .

Each party i picks a random degree $(k - 1)$ polynomial $f_i \in \mathbb{Z}_M[x]$ where

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

This i th party then computes $f_{i,j} = f_i(j)$ and then privately sends $f_{i,j}$ to party P_j , for all $j = 1, \dots, n$. Note that the $f_{i,j}$ for $j = 1, \dots, n$ are the standard k -out-of- n Shamir sharing [32] of d_i . In addition, the i th party also computes $b_{i,j} = g^{a_{i,j}}$ (mod N) for $j = 0, \dots, (k - 1)$, where we let $a_{i,0} = d_i$. He then broadcasts $b_{i,0}, \dots, b_{i,(k-1)}$ to all other parties, these are the commitments of all the coefficients of the polynomial $f_i(x)$.

At this point each party j receives $f_{1,j}, \dots, f_{n,j}$. Player j verifies that:

$$\begin{aligned} g^{f_{i,j}} &= g^{f_i(j)} \pmod{N} = g^{[a_{i,k-1}j^{k-1} + \dots + a_{i,1}j + d_i]} \pmod{N} \\ &= \prod_{t=0}^{k-1} (g^{a_{i,t}})^{j^t} \pmod{N} \\ &= \prod_{t=0}^{k-1} b_{i,t}^{j^t} \pmod{N}. \end{aligned}$$

If this does not hold then the protocol is aborted, as one knows that player i is not honest.

We finally set

$$\begin{aligned} S &= \{k, M, g\}, \\ P_i &= \{\{b_{j,l}\}_{j=1\dots n, l=0,\dots,k-1}\}, \\ S_i &= \{d_i, \{a_{i,j}\}_{j=1}^{k-1}, \{f_{j,i}\}_{i \neq j}\}. \end{aligned}$$

6.2.2 Subset Presigning Algorithm:

Recall in this stage we have as input $I = \{t_1, \dots, t_k\}$ and $I' = \{1, \dots, n\} \setminus I = \{t_{k+1}, \dots, t_n\}$. This protocol is executed amongst the k parties represented by the set I .

Each party $t_i \in I$ computes

$$\begin{aligned} \lambda_{t_i} &= \prod_{1 \leq j \leq k, j \neq i} \frac{t_j}{t_j - t_i} \pmod{M}, \\ s_{t_i} &= \left(\sum_{j=k+1}^n f_{t_j, t_i} \right) \lambda_{t_i} \pmod{M}. \end{aligned}$$

They also compute $h_{t_i} = g^{s_{t_i}} \pmod{N}$, which is the commitment to the share s_{t_i} of party t_i . This commitment will help other parties to verify the signature share generated by party t_i later on in the scheme.

Note that we have

$$\begin{aligned} s_{t_1} + \dots + s_{t_k} &= \left(\sum_{i=k+1}^n f_{t_i, t_1} \right) \lambda_{t_1} + \dots + \left(\sum_{i=k+1}^n f_{t_i, t_k} \right) \lambda_{t_k} \pmod{M} \\ &= \sum_{i=1}^k (f_{t_{k+1}, t_i} \lambda_{t_i}) + \dots + \sum_{i=1}^k (f_{t_n, t_i} \lambda_{t_i}) \pmod{M} \\ &= d_{t_{k+1}} + \dots + d_{t_n} \pmod{M} \end{aligned}$$

The last equality follows from the properties of Shamir's secret sharing scheme and the fact that M is a large prime.

Over the integers we then have that

$$\sum_{i=1}^k s_{t_i} = d_{t_{k+1}} + \dots + d_{t_n} + x_I M$$

for some integer $x_I \in [k - n, k]$. That x_I lies in such a small interval is because s_{t_i} and d_{t_i} are both positive and smaller than M for all i .

To determine the value of x_I the parties then produce their signature share σ_{I,t_i} on the dummy message, $m' = 2^e \pmod{N}$. That is they compute and broadcast,

$$c'_{t_i} = 2^{e(d_{t_i} + s_{t_i})} \pmod{N}.$$

plus a proof of its correctness. The signature share verification algorithm is then executed, to check that c'_{t_i} is valid.

The value of x_I can then be computed via exhaustive search by verifying which value makes the following equation hold,

$$\begin{aligned} \prod_{t_i \in I} c'_{t_i} &= 2^{e(\sum_{t_i \in I} d_{t_i} + s_{t_i})} \pmod{N} \\ &= 2^{e(x_I M + \sum_{i=1}^n d_i)} \pmod{N} \\ &= 2^{e x_I M + e d} \pmod{N} \\ &= 2 \cdot (2^{eM})^{x_I} \pmod{N}. \end{aligned}$$

We finally set

$$\begin{aligned} D_I &= \{x_I, \{h_{t_i}, c'_{t_i}\}_{t_i \in I}\}, \\ S_{I,t_i} &= \{s_{t_i}\}. \end{aligned}$$

Notice, that from c'_{t_i} the correctness of x_I can be verified publicly.

6.2.3 Signature Share Generation Algorithm:

The parties now want to obtain a signature on the message m . Each party t_i computes

$$c_{t_i} = m^{d_{t_i} + s_{t_i}} \pmod{N}$$

along with a (non-interactive) proof that

$$\begin{aligned} \text{DLog}_m c_{t_i} &= d_{t_i} + s_{t_i} \\ &= \text{DLog}_g (g^{d_{t_i} + s_{t_i}}) \\ &= \text{DLog}_g (b_{t_i,0} \cdot h_{t_i}). \end{aligned}$$

This proof can be produced using a standard adaption of the protocol of Chaum and Pederson [9] to a group with unknown order. We let this proof be denoted by $\mathcal{P}(m, c_{t_i}, t_i)$. The signature share is then given by

$$\sigma_{I,t_i} = \{c_{t_i}, \mathcal{P}(m, c_{t_i}, t_i)\}.$$

6.2.4 Signature Share Verification Algorithm:

Each signature share $\sigma_{I,t_i} = \{c_{t_i}, \mathcal{P}(m, c_{t_i}, t_i)\}$ can be verified by checking whether the proof $\mathcal{P}(m, c_{t_i}, t_i)$ holds.

6.2.5 Share Combining Algorithm:

At this point in the scheme, the signature shares

$$\sigma_{I,t_i} = \{c_{t_i}, \mathcal{P}(m, c_{t_i}, t_i)\}$$

have been published and verified to be correct. To produce the final signature on the message m we compute

$$\begin{aligned} m^{-x_I M} \prod_{i=1}^k c_{t_i} &= m^{-x_I M} \prod_{i=1}^k m^{d_{t_i} + s_{t_i}} \pmod{N} \\ &= m^{-x_I M + \sum_{i=1}^k d_{t_i} + s_{t_i}} \pmod{N} \\ &= m^{\sum_{i=1}^n d_i} = m^d \pmod{N} \\ &= s. \end{aligned}$$

6.3 Discussion of the Above Scheme

6.3.1 Interactiveness

If the k parties I want to sign a different message, they need to come back the Signature Share Generation Algorithm. This is because the values of x_I , s_{t_i} etc can be reused. signature share generation and combining the signature shares.

If there is change in the threshold set parties, from I to I' , then all the shares $s_{t'_i}$, for $t'_i \in I'$, will have to be calculated again as their values depend on the set I' . So that the parties need to come back to Subset Presigning Algorithm stage. A new value of $x_{I'}$ needs to be determined. Note that the threshold value, k , still remains the same in this case.

If all parties agree to decrease the threshold value, k , then all of them need to run the protocol from the beginning, i.e. the Dealing phase. Also note that the threshold value can only be decreased, but not increased.

Hence, we can the scheme partially interactive since of the five main stages of the protocol the only interactive stages are the Dealing and Subset Presigning Algorithms.

6.3.2 Share Refreshing

In Rabin's threshold scheme, [30][Figure 3], the additive shares of the absent parties must be reconstructed at one place or by a single party. The reason for this requirement arises from the difficulty in working with modulo $\phi(N) = (p-1)(q-1)$ that remains unknown to all parties.

As a consequence, all the n parties have to refresh or renew their additive shares of the secret exponent d whenever there is a change in either the threshold value of k or the set of the threshold parties, I . Note, that the value of d still remains the same, what differs are the individual shares of each party. If they do not renew their shares, then a certain single party might end up to know the shares of all other parties in the scheme, for example, if we have three parties P_1, P_2 , and P_3 and a (2,3) threshold scheme:

- In the first signature, party P_2 is away and its share is reconstructed by party P_1 . So P_1 now knows shares d_1 and d_2 .

- In the second signature, party P_3 is away and its share is reconstructed by party P_1 again. So P_1 now knows shares d_1 , d_2 , and d_3 . That means the first party knows the shares of all parties.

This problem leads Rabin's scheme to require a Share Refreshing stage, which is interactive and requires the presence of all parties who wish to continue in the scheme.

In contrast to Rabin's scheme, in our scheme each individual share, d_i , is reconstructed in a way that does not further reveal its value to any party after any number of changes in the set of threshold parties. As a result, the scheme not only does not leak any information about the share to any one else but also avoids the refreshing procedure that requires the interaction of all parties in the scheme.

6.3.3 Robustness

Even though the protocol is always $(k-1)$ private, i.e. no information is leaked when up to $(k-1)$ parties corrupt, it does not mean the combining signature process will be always successful.

If the number of dishonest parties is l , and $(n-k) < l < k$, then the number of honest parties is $(n-l)$ and $n-l < n-(n-k) = k$. That means a valid signature on a message cannot be obtained. As a result, to make sure that this never happens we require that $(n-k) > k$, i.e. $k < \lfloor \frac{n}{2} \rfloor$.

7 Design, Implementation and Testing

7.1 Choice of Language

The language chosen for this project is Java because Java offers several libraries that are not available or not well developed in many other languages such as C or C++ listed below:

- “BigInteger” library can manipulate unlimited size numbers used in public cryptography.
- It also has well developed library that supports the front end such as GUI and API and therefore they all can be done very easily and quickly in Java.
- Secure Socket Layer, SSL, is also can be integrated easily with socket. It is important to have this feature because information transmitted between any pair of parties needs to be kept secure and confidential sometimes even to the other parties in the scheme.
- The final important feature is the Threat library. The nature of the protocol is that any single party has to talk to all other parties in the same time, so each such connection is dealt by a separated threat. So that we really need a well threat supported library and that is exactly what Java is very good at.

All of these above features of Java have saved me so much time, and as a result more time has been spent on trying to understand the cryptography issues and improving the cryptographic algorithms.

7.2 Requirement Analysis

The main aim of this project is to implement the Shared RSA Key Generation protocol invented by Bonel and Franklin and then integrated it with my proposed Threshold Signature scheme to see whether these protocols are correct or not and if so how efficiently they are.

Therefore not much effort was put in dealing with the presence of adversary, I would assume every parties in the scheme is honest. Only the raw algorithm of encryption and decryption were implemented. The code also does not get involved into the complication of verification. As a consequence the program cannot detect where and who has not cooperated properly in the protocol.

As efficiency is one of the most important factor of the project and therefore so much thought has been taken to speed up the protocol as much as possible. The reason we have to be very careful with speed is because manipulating with big numbers, some thousand bits, such as mod power, multiplication is going to be very expensive. In order to achieve the goal, I have taken two different approaches concurrently described below:

- **Theoretical approach:** This involves analysing the algorithms used in the protocol to see whether or not I can improve their complexity. One of several typical examples is the usage of the technique of Distributed Sieving to speed up primality test.

- Practical and Programming approach:** In order to use the advantages of multiple parties participated in the scheme, in almost operations, the tasks of trial division, primality test, and many more are split equally to all parties. The reader might want to look at them in more detail in section 5.2.3 of the thesis. In addition, network topology has also been used to find the best solution for communication between all parties in the scheme to avoid the situation where too much information is transmitted via a single party as well as in dealing with deadlock, starvation etc , and the problem will be discussed in more detail in next section.

7.3 Network protocol

Network presents an interesting problem, it is not suitable for a central server to take a big responsibility in running the protocol due to the absence of a trusted third party in the scheme. Therefore the central server structure is definitely unsuitable in this circumstance.

As the information transmitted between any pair of parties must be kept unknown not only to outsiders but also to other parties in the protocol. Therefore Secure Socket Layer (SSL) has been put on top of the network to satisfy the requirement. All connection between two parties are direct and that means if party *A* wants to send information to party *B* then the information is transmitted directly to *B* and not via any one else in between.

As a result, the protocol often takes a short time at the beginning to set up so each party knows the IP addresses and port numbers of all other parties in the network.

In order to exploit the capability of all parties, nearly every major operation of the protocol is done in parallel. In general, there are two main types of network communication as follows:

7.3.1 N-ary tree network structure

As the reader can see from the figure 1,

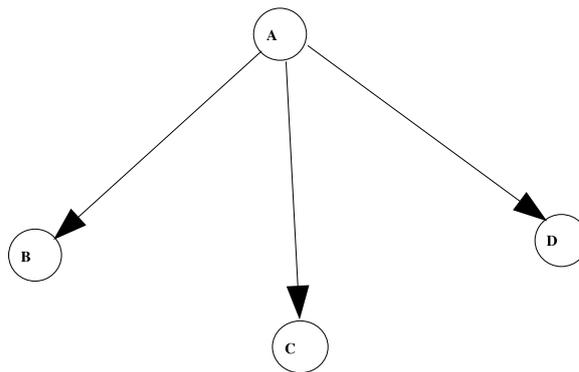


Figure 1: N-ary network structure.

in this structure, one party talk to all other parties. This kind of structure is used in various places in the protocol such as setting phase, trial division, and

primality testing.

Each connection is dealt by a separated thread created at the beginning of the connection, all parties, A , B , C and D can do calculation in the same time and therefore the total time is decreased by a factor of number of parties. As the most expensive part of the protocol is the primality testing, so the more the number of parties, the quicker the protocol will be, that means the protocol tends to run faster when there are more parties taking a part in. Readers can see more detail about this in section 8.

7.3.2 Fully connected network structure

Perhaps, I should start this section by a brief introduction of the nature of the problem.

There are N parties and each party needs to receive private information from all other parties. When it has received all required data, it will then do some calculation and then split the result into n pieces and then send each piece to each party. This whole process can be summarised as follows:

- In the first half, each party P_i needs to send information to all other parties in the network.
- In the second half, when each party P_i has received all information it needs and has done some calculation, it splits the result into n pieces: I_1, \dots, I_n and sends each piece I_j to party P_j .

The difficulty is that it seems there is no way to notify each thread used in the first half when all information have been received and computed by each party in the second half and therefore I cannot reuse the connection already established between any pair of parties in the first half of the procedure.

The second unexpected problem, I encountered in this part is that if I put SSL on top of the network then deadlock will occur when two parties A and B do the following things simultaneously:

- Client at A tries to establish a connection with server at B on port number X and
- Client at B tries to establish a connection with server at A on port number Y where X is different from Y .

The reason I mention the two problems here is because I took me quite a while to figure out what were happening and then went on to find a solution for this problem described below.

Assume, without loss of generality, there are 4 parties in total, labelled with A , B , C , and D , note that each party can be represented by a computer or a process.

In the first round of an iteration:

- Party A transmits information to parties D , and then C , and then B .
- Party B transmits information to parties D , and then C .
- Party C transmits information to parties D .

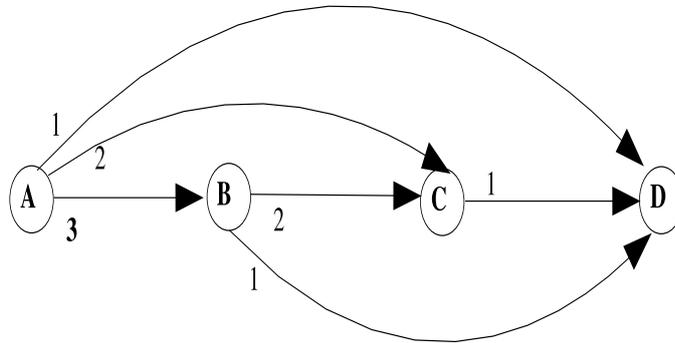


Figure 2: First round communication.

So graphically, communication between all parties looks like in figure 2.

Note that all *A*, *B* and *C* transmit information simultaneously and party *D* will be the one that first receives all required information. Whereas parties *A*, *B*, and *C* are still waiting information that are sent out in the second round of the iteration.

In the second round of an iteration:

- Party *D* transmits information to parties *C* and then *B* and then *A*.
- Only after *C* has received data from *D*, it transmits information to *B* and then *A*.
- Only after *B* has received data from *C*, it transmits information to *A*.

So graphically, communication between all parties looks like in figure 3.

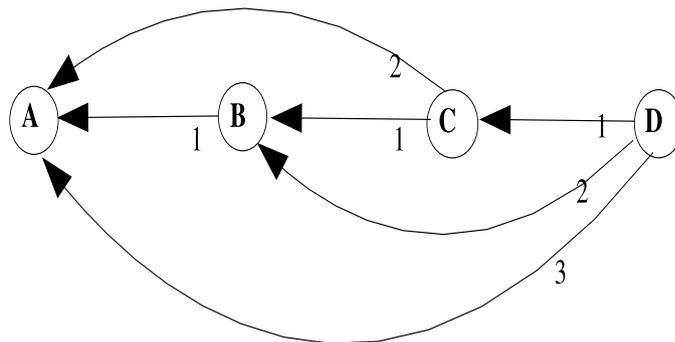


Figure 3: Second round communication.

The result of this algorithm helps me make sure that:

- *D* is always the party who finishes network connection firstly.

- *C* is always the party who finishes network connection secondly as it only finishes when *C* has received data from *D*.
- *B* is always the party who finishes network connection thirdly as it only finishes when *B* has received data from *C*.
- *A* is the final party that finishes network and calculation.

So why I have to come up with a rather complex network algorithm, the reason is because if I am going to reuse the network protocol one above the other then things must be in the right order otherwise in the worst situation, deadlock or starvation will definitely occur.

Clearly the payload complexity of the above scheme is $O(n)$ where n is the number of parties compared to $O(n^2)$ if I make all payload be transferred via a single party that will be then a bottleneck in a network system.

7.4 Testing

It is quite simple to test the functionality of the protocol as there are only two main operations that are encrypting a message and decrypting a cipher-text. And as long as we get the right message after decrypting the corresponding cipher-text then the scheme works correctly as the chance of getting the correct result of a false system is extremely small.

In contrast, it is much harder to test network synchronisation because bugs do not normally cause the program to hang until it has been run in a large scale where there are many parties/computers taking a part in the scheme and they communicate with one another for a very long period of time.

In order to assure that unexpected phenomenas such as deadlock or starvation do not occur, I have carried out testing the protocol rigorously, with large number of players, up to ten at the moment, the key lengths have been tested are 512, 1024 and 2048 bits. They run concurrently on different types of machines and computers by using variety of operating systems such as Linux, Unix and Window Microsoft (by using SSH). In addition, the machines are also located in different places, such as five laboratories in Merchant Venturers Building, department of Computer Science, the University of Bristol as well as machines in various libraries and other buildings.

Of course, it is not possible to say that the protocol will work up to any number of parties, however, as far as there are fewer than ten parties, it seems to be very robust.

7.5 Running the protocol

Very different from other types of software, one should be careful when he or she wants to run the program. The reason is because in order to generate the correct public/private key pair in an optimal time, all of the parameters need to be selected and inputted accurately, if not it is unlikely that the program will terminate. Having thought about this problem, I have written a document that helps new user to known how to use the manual to assign the parameters correctly. And as long as the key length is less that or equal to 2048 bits and the number of parties is smaller than 10, all the parameters will be assigned automatically. An alternative way is to look at the result tables presented at

the end of this report, as all the parameters were shown clearly for the purpose of comparison.

7.6 Limitation and Future work

As I am only interested in the accuracy and efficiency of the protocol, so this is not a finished product and there are many areas that are still missing or needed to be improved and upgraded in the future. The four main areas are:

- **Network protocol:** My program should be able to deal with cases like what happen when a disconnection occurs due to connection time out or machine break down or even when a party does not want to cooperate with other parties any more. The opposite case will be re-connection, and obviously, the scheme will have to be started from the beginning in this case.
- **API:** the API needs to be extended to allow a variety of functions that can be incorporated with the scheme such RSA-Full Domain Hash.
- **Verification:** This part has not been done in this project, so we shall need to develop it from the very beginning.
- **Big number manipulation:** At the moment, all numbers are of type “BigInteger”, the problem is that we do not know exactly how the library was implemented and whether it has been implemented by using the most efficient algorithm up to date or not. An alternative approach is to write our own library that is responsible for representing big number and its operations such as mod power, multiplication, and division by using the best currently known arithmetic method, Montgomery Arithmetics.

Number of Parties	Number of Trial	Sieve time	N Computation	Trial Division	Primality Test	Total Time
3	40	1	1	1	2	9
4	40	2	2	2	2	12
5	40	4	3	1	1	13
6	80	6	17	3	4	35
7	80	9	25	3	5	46
8	50	12	11	2	2	32
9	100	18	60	4	6	96
10	100	26	80	4	8	130

Table 1: Shared key generation time when moduli N is 512 bits, times are measured in second

Number of Parties	Number of Trial	Sieve time	N Computation	Trial Division	Primality Test	Total Time
3	60	1	2	3	15	26
4	50	5	8	8	35	57
5	80	4	14	5	56	93
6	80	6	20	5	41	77
7	80	9	29	4	38	88
8	80	13	37	4	43	105
9	90	19	59	5	49	140
10	60	47	58	10	56	188

Table 2: Shared key generation time when moduli N is 1024 bits, times are measured in second

8 Experiment Result and Discussion

There are six main stages in the Shared Key Generation protocol and four stages in the Threshold Signing Scheme. However, only four stages, Distributed Sieving, N Computation, Trial Division and Primality Testing that consume a lot of running time of the protocol. The rest can be done instantaneously.

I have measured the performance of shared key generation in a number of environments, which has been described previously in section 7.4.

From the three result tables, Distributed Sieving can be done very fast, as the number of trials is only in the range between 50 and 300. However, the number of trials in N computation stage is equal to the number of trials in Distributed Sieving squared and therefore N Computation takes much longer time to compute.

As a result of Distributed Sieving, all the candidates for N are not divisible by any prime number between 0 and some bound and it has a direct impact on the result of Trial Division. In the current protocol, trial division often decreases the number of candidate for N by a factor of three, it should have been much more efficient if I had taken Sieving out.

It is clear that the most expensive stage of the protocol is Primality Testing because of many modular power operations are carried out in all parties at this stage of the protocol.

Number of Parties	Number of Trial	Sieve time	N Computation	Trial Division	Primality Test	Total Time
3	50	3	7	11	260	283
4	60	2	8	6	163	192
5	50	10	25	15	450	542
6	85	15	76	31	1050	1471
7	100	18	100	23	1150	1248
8	80	35	105	30	1100	1274
9	100	35	117	29	1350	1551
10	100	48	156	25	1005	1261

Table 3: Shared key generation time when moduli N is 2048 bits, times are measured in second

8.1 Scaling to many parties

It seems to me that the protocol can be scaled well in term of number of parties. I have tested the protocol up to ten parties and the time difference between 10 and 3 parties are not very much compared to the increase when I double the length of the secret key, which will be discussed later. The reason for this is that the more number of parties participate in the protocol, the less work each party has to carry out in the long run due to nice distribution of computation in two stages, Primality testing and Trial Division. That is why you can see from the result tables that the time of Primality Testing and Trial Division actually decrease slightly when the number of parties increases, this is most clearly shown when the key length is large, 2048 bits. As they are the most expensive parts of the protocol and therefore it has a huge impact on the chance of using this protocol in practice.

8.2 Doubling the length of RSA moduli

In contrast to how little the impact of increasing the number of parties has on the overall run time, there is a big difference when I increase the length of N moduli from 512 to 1024 bits and particularly from 1024 to 2048 bits. So why it is the case here, first of all, the protocol has quadratic complexity in term of number of bits of RSA moduli due to the difficulty in finding two big prime numbers simultaneously. A more intuitive reason might be that when I double the length from 1024 bits to 2048, what I have done is to increase the moduli N by 2^{1024} times and it is a huge number, so that it is much harder to find a prime number at that size. Finally, manipulating with 2000 bits number such as division and multiplication is very slow.

9 Conclusion

A new threshold Signing scheme is proposed in this project that when combined with Shared RSA secret keys generation will leads us to a complete solution for the Threshold RSA problem. The complete solution has also been implemented successfully in this project.

References

- [1] Adam Barnett and Nigel P. Smart. Mental Poker Revisited. *Cryptography and Coding 2003*, Springer-Verlag LNCS 2898, pp. 370-383, 2003.
- [2] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. *Proc. 1st ACM Conference on Computer and Communications Security, 1993*, 62–73, 1993.
- [3] Benaloh. Secret sharing homomorphisms: keeping shares of a secret. *Advances in Cryptography - CRYPTO '86*. Lecture notes in Computer Science, vol. 263. Springer-Verlag, New York, LNCS 263, pp. 251-260.
- [4] Ben-Or, M. Goldwasser and Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *Proceeding of the 20th Annual ACM Symposium on Theory of Computing*. Chicago, Ill, May 2-4. ACM, Newyork, pp. 1-10.
- [5] D. Boneh and M. Franklin. Efficient generation of shared RSA keys. *Advances in Cryptography - CRYPTO '97*, Springer-Verlag LNCS 1233, 425–439, 1997.
- [6] Dan Boneh and Matthew Franklin. Efficient Generation of shared RSA keys. *J. ACM*, 48, 702-722, 2001.
- [7] C. Boyd. Digital Multisignatures. *Cryptography and Coding 1989*. Institute of Mathematics and its application, IMA. 241–246, Clarendon Press, 1989.
- [8] D. Chaum, C. Crepeau and I. Damgard. Multiparty unconditional secure protocols. *ACM STOC 1988*. 11–19, 1988.
- [9] D. Chaum and T. Pedersen. Wallet databases with observers. *Advances in Cryptology - CRYPTO '92*, Springer-Verlag LNCS 740, 89–105, 1992.
- [10] R. Cramer, M. Franklin, B. Schoenmakers and M. Yung. Multi-authority secret-ballot elections with linear work. *Advances in Cryptography - EURO-CRYPT '96*. Springer-Verlag LNCS 1592, pp. 223-238, 1999.
- [11] A. DeSantis, Y. Desmedt, Y. Frankel, M. Yung. How to share a function securely. *STOC 1994*. 522–533, 1994.
- [12] Yvo Desmedt. Threshold Cryptography. In G Brassard, editor, *European Transactions on Telecommunications*, 5(4): 449-457, July 1994.
- [13] Y. Desmedt. Society and group oriented cryptography: an new concept. *Advances in Cryptography - CRYPTO '87*, Springer-Verlag LNCS 293, 120–127, 1987.
- [14] Y. Desmedt and Y. Frankel. Threshold Crypto-Systems. *Advances in Cryptography - CRYPTO '89*, Springer-Verlag LNCS 435, 307–315, 1989.
- [15] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. on Info. Theory*, 31, 644–654, 1976.
- [16] ElGamal. A public key crypto-system and a signature scheme based on the discrete logarithm. *IEEE Trans. Inf. Theory* 31. pp 469-472.

- [17] U. Feige, A. Fiat and A. Shamir. Zero knowledge proofs of identity. *Journal of Cryptology* 1, 1988. pp. 77-94
- [18] Fiat and Shamir. How to prove yourself: Practical solutions to identification and signature problems. *Advances in Cryptography - CRYPTO '86*. Lecture notes in Computer Science, vol. 263, Springer-Verlag, LNCS 263, pp. 186-194, 1987.
- [19] Y. Frankel. A practical protocol for large group oriented networks. *Advances in Cryptology - EUROCRYPT '89*, Springer-Verlag LNCS 434, 56-61, 1989.
- [20] M. Franklin and S. Haber. Joint encryption and message-efficient secure computation. *Journal of Cryptology* 9, 1996. pp. 217-232.
- [21] R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin. Robust and efficient sharing of RSA functions. *Advances in Cryptology - CRYPTO '96*, Springer-Verlag LNCS 1109, 157-172, 1996.
- [22] O. Goldreich, S. Micali and A. Wigderson. How to play any mental game. *STOC 1987*, 218-229, 1987.
- [23] L. Guillou and J. Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimising both transmission and memory. *Advances in Cryptography - EUROCRYPT '88*. LNCS 330, pp 123-128, 1988.
- [24] M. Maklin, T. Wu, and D. Boneh. Experimenting with Shared Generation of RSA keys. *Proceedings of the Internet Society's 1999 Symposium on Network and Distributed System Security*. Internet Society, Reston, Va., pp. 43-56.
- [25] Hoang Long Nguyen. Partially Interactive Threshold RSA Signatures. *Cryptography and Coding*. Institute of Mathematics and its application, IMA. Unpublished, 2005.
- [26] K. Ohta and T. Okamoto. A modification of the Fiat-Shamir scheme. *Advances in Cryptography - CRYPTO '88*. LNCS 403, pp. 232-243, 1990.
- [27] H. Ong and C. Schnorr. Fast signature generation with a Fiat-Shamir like scheme. *Advances in Cryptography - EUROCRYPT '90*. LNCS 473, pp. 432-440, 1991.
- [28] P. Paillier. Public key crypto-systems based on composite residue classes. *Advances in Cryptography - EUROCRYPT '99*. Springer-Verlag LNCS 1070, pp. 72-83, 1996.
- [29] T. Pederson. A threshold crypto-system without a trusted dealer. *Advances in Cryptology - EUROCRYPT '91*, Springer-Verlag LNCS 547, 522-526, 1991.
- [30] T. Rabin. A simplified approach to threshold and proactive RSA. *Advances in Cryptology - CRYPTO '98*, Springer-Verlag LNCS 1462, 89-104, 1998.

- [31] T. Rabin, A. Shamir and L.M Adleman. A method for obtaining digital signatures and public-key crypto-systems. *Communications of the ACM*, **21**, 120–126, 1978.
- [32] A. Shamir. How to share a secret. *Communications of the ACM*, **22**, 612–613, 1979.
- [33] V. Shoup. Practical threshold signatures. *Advances in Cryptology – EUROCRYPT 2000*, Springer-Verlag LNCS 1807, 207–220, 2000.
- [34] A. Yao. How to generate and exchange secrets. *FOCS 1986*. 162–167, 1986.