CHAPTER -5
5. EFFICIENT METHODS OF IMPLEMENTING SHARED KEY GENERATION ALGORITHMS

5.1 INTRODUCTION

This chapter discusses about the algorithms that improve the speed of the shared key generation algorithm introduced in the last chapter and also applied on shared key generation using RSA algorithms. The efficient algorithm for RSA implementation presented in 3.3 is used to improve the speed of the shared key generation using RSA. Montgomery methods that have been applied on truncated polynomials in 3.4 are applied over shared key generation using NTRU.

5.2 AN EFFICIENT METHOD FOR IMPLEMENTING SHARED KEY GENERATION BASED ON RSA

This section shows a method to increase the speed of computation involved in the generation of shared keys. The method makes extensive use of Montgomery multiplication algorithm in the process of generation of shared keys using RSA. It is found that throughout the process of shared key generation using RSA a lot of data needs to be exchanged among the communicating parties for secure establishment of the shared parts of the private keys. They have a constraint that during establishment of shared secrets, one party should not be able to compute the secret share of the other party. Hence in the process of exchanging data it is usually sent in encrypted form. During the process it is presumed that all the parties entering into communication have
an already established set of public and private key pair of a particular public key cryptosystem. Encryption or decryption takes place in using this PKCS. In the subsequent discussion it is assumed that all the parties use RSA algorithm.

In this section a multiparty communication among \( k \) parties is considered, with each party \( i \) having its own previously selected RSA public/private key pair \( e_i, d_i \) and modulus \( M_i \). \( M_i \) should be as large as possible close to the range of \( 2^N_{\text{max}} \) where \( N_{\text{max}} \) is the largest possible value of \( N \).

The process of shared key generation can be split into 4 modules. Module 1 deals with the generation of \( N \), which is the product of two primes \( p, q \). It also describes the method where each party can test if \( N \) is a product of two primes \((p, q)\) or not without knowing the values of \( p \) and \( q \). The second module describes the method of computing shared private keys for each party and a public exponent \( e \) uses the Benaloh's protocol[21]. The last module gives the algorithm for encryption/decryption of a message in a multiparty communication.

1: The computation of the value \( N \) which is the product of two prime numbers \( p, q \) is addressed in this section. Here each party contributes their own share of \( p_i \) and \( q_i \) to the values of \( p \) and \( q \). The following algorithm describes a method of computing the value of \( N \) using the contributions by each party, and at the same time not revealing the values of \( p \) and \( q \) efficiently.

The participants also produce a set of additives \( a_{ij} \) for \( j = 1, 2, \ldots, k \), such that each \( a_{ij} \) satisfies the inequality \( 0 < a_{ij} < N_{\text{max}} \), where \( N_{\text{max}} \) is the maximum possible value of \( N \).
The algorithm is as follows

1. Calculate $y_{ip} = \text{ModExp}(p_i, e_i, M_i)$

2. $y_{iq} = \text{ModExp}(q_i, e_i, M_i)$

3. Broadcast $y_{ip}, y_{iq}$ and $M_i$

4. At this point each participant has the public keys and the modulus of all the members involved in the communication. With the values received from the $j^{th}$ party, party $i$ encrypts the $p_i q_j, q_i p_j$ with the public key and modulus conveyed by $j$ by following the steps mentioned below.

   $z_1 = \text{ModExp}(p_j q_i, e_j, M_j)$

   $z_2 = \text{ModExp}(p_j q_i, e_j, M_j)$

   $z_3 = \text{ModMul}(a_j, 1, M_j)$

5. Let each participant generate $K$ numbers $x_k$, randomly chosen such that $\sum_k x_k = 1 \mod M_j$

6. For each $x_k$ calculate $S_k = \text{ModExp}(x_k, e_j, M_j)$

7. Now split $z_1$, $z_2$, $z_3$ into $k$ parts each using the formula

   $S_{ik} = z_i S_k \mod M_j$

8. Send to party $j$, $S_{ik}$ not in an order but in a random fashion, so that party $j$ does not know which part corresponds to each of the 3 numbers that $i$ is sending. To be more secure make $K$ as large as possible.

9. $j^{th}$ party will decrypt all the $3k$ values, he receives from $i^{th}$ party and sums them up, to get the value of $p_i q_j + q_i p_j + a_j \mod M_i$. As $M_i$ is very large $j$ gets the value of $p_i q_j + q_i p_j + a_j$. 

46
10. To the decrypted sum obtained from all the parties, i adds \(2p_iq_i + a_i\) and obtains \(N_i\).

11. Broadcast \(N_i\) to all the parties

12. \(N = \sum N_i / 2\)

13. To test if \(N\) is a product of two primes or not Select a random value \(x\), common to all parties.

14. Each party calculates \(F_i = \text{ModExp}(x, p_i + q_i, N)\). These values are then exchanged between the parties.

15. Calculate \(F_2 = \text{ModExp}(x, N+1, N)\)

16. If \(N\) is the product of two primes \(F_2 = \sum F_i\)

If \(N\) fails the test new values of \(p\) and \(q\) need to be chosen, by repeating from step 1.

Algorithm 5.1

2: The public exponent \(e\) is agreed upon by all the parties.

1) Party 1 calculates \(\phi_1 = N - p_1 - q_1 + 1\), the remaining parties i calculate \(\phi_i = -p_i - q_i\) for \(i = 2, \ldots, n\)

2) Calculate \(\phi = \sum_{i=1}^{k} \phi_i\) using the protocol calculate

\[ l = \sum \phi_i \mod e \]

3) \(\zeta = l^{-1} \mod e\)

4) Party i computes its shares \(d_i = \lfloor -\zeta \cdot \phi_i / e \rfloor\).

5) \(d = \sum d_i + r\) using the Benaloh’s protocol mentioned in step 2.
6) To make \( d = \Sigma d_i \mod n \) one trial decryption is necessary. The following steps are to be followed to calculate \( d = \Sigma d_i \mod n \):

a) Party 1 picks a random message \( m \)

b) Since \( e \) is the publicly known compute \( C = \text{ModExp}(m,e,N) \)

c) Party 1 asks all parties involved to decrypt with their own share of \( d_i \). 
   Each party computes
   
   \[
   (1) m_i = \text{ModExp}(C, d_i, N) \quad \text{and sends} \quad m_i \quad \text{to party 1}
   \]

d) Party 1 computes

   i) \( x_1 = \text{ModExp}(C, r, N) \)

   ii) \( M = (\Pi m_i) \times x_1 \mod n. \)

   iii) For different values of \( r \) in the range \( 1 \leq r < k \) and obtains the value of \( r \).

   e) then party 1 updates \( d_1 \leftarrow d_1 + r \), Hence \( \Sigma d_i = d \mod \varphi \)

Algorithm 5.2

3.:

1. For a message \( m \) compute \( S_i = \text{ModExp}(m, d_i, N) \)

2. Send \( S_i \) to combiner who knows no secret. The combiner just makes

   \[
   S = \prod_{i=1}^{n} S_i
   \]

   and sends it as it is a ciphered document.

Algorithm 5.3
5.3 MONTGOMERY METHODS OVER SHARED KEY GENERATION BASED ON TRUNCATED POLYNOMIALS

This section presents an efficient method of shared key generation basing on NTRU using Montgomery methods. It is assumed that the present group consists of two members only. However the algorithm works for a group greater than two. It is assumed that previous to the generation of the shared secrets each party has its own set of public/private key pairs \((h_{1p}/f_{1p}, f_{1pp})\) and \((h_{2p}/f_{2p}, f_{2pp})\). Further it is assumed that NTRU PKCS is used as a common platform for communication purposes. Let the public key of the trusted TTP be \(h_{TTP}\) and its private key be \((f_{TTP}, f_{pp})\). Let \(p_{p}, q_{p}\), and \(N_{p}\), be the parameters already agreed upon previous to the establishment of the shared secret keys.

5.3.1 Generation of Shared Secrets

1. Each of the participants generate polynomials \(f_1, g_1\) and B generate \(f_2, g_2\)

2. Let each party choose one random polynomial each say \(r_1, r_2\)

3. let A calculate \(e_1 = \text{mon}_\text{NTRUMUL}(r_1, h_{TTP}, q_{P}) + f_1 \mod q_{p}\) be sent to the TTP

   Similarly B calculate \(e_2 = \text{mon}_\text{NTRUMUL}(r_2, h_{TTP}, q_{P}) + f_2 \mod q_{p}\)

4. The TTP will then decrypt the values sent by each of the parties as described below

   Upon receiving the ciphered text \(e_i\) the TTP will have to compute a basis on the
formula \[ a = \text{mon}_n \text{trumul}(f_{\text{TP}}, e, q) \]

Express the coefficients of \( a \) in the range \([-q/2 \quad q/2]\)

Compute \( b = \text{mon}_n \text{trumul}(a, f_{\text{TP}}, p) \)

Original message \( m = b = f_i \)

5. Once all the values of \( f_i \) are retrieved the TTP will calculate \( f = f_1 \times f_2 \)

6. Find the inverse \( f_p \) of \( f \). If the inverse does not exist repeat the above steps until inverse of \( f \) is found.

7. The TTP will factorize \( f_p \) into parts \( f_{p1} \) and \( f_{p2} \) and also calculate \( f_q \).

8. The TTP will then send the shares of \( f_p \) and \( f_q \) to A and B as follows:

   choose a random polynomials \( r_1 \) and \( r_2 \) and calculate

   \[ E_{1p} = \text{mon}_n \text{trumul}(r_1, h_{1p}, q_p) + f_{p1 \mod q_p} \]

   \[ E_{2p} = \text{mon}_n \text{trumul}(r_1, h_{1p}, q_p) + f_{q \mod q_p} \]

   Similar procedure will be executed to send \( f_{p2} \) and \( f_q \) to B.

9. Upon receiving the encrypted message A will decrypt the message using the following steps:

   a. After receiving the ciphered text \( E_{1p} \) and \( E_{2p} \), compute \( a1 \) and \( a2 \) by using formula given below

      \[ a1 = \text{mon}_n \text{trumul}(E_{1p}, f_{1p}, q_p) \]

      \[ a2 = \text{mon}_n \text{trumul}(E_{2p}, f_{1p}, q_p) \]

   b. Express the coefficients of \( a1 \) and \( a2 \) in the range \([-q/2 \quad q/2]\)
c. Compute \( b_1 = \text{mon}\_\text{ntrumul}(a_1, \ f_{1pp}, p_p) \)

\( b_2 = \text{mon}\_\text{ntrumul}(a_2, \ f_{1pp}, p_p) \)

d. The values of \( b_1 \) and \( b_2 \) are the values of the shares of the secret key \( f_{p1} \) and \( f_q \).

B will also execute the same procedure to recover the values of \( f_{p2} \) and \( f_q \).

10. The shares of the polynomial \( g_1 \) and \( g_2 \), generated by A and B will then be exchanged as follows.

a. Let A generate a random polynomial \( r \)

b. Cipher \( C_1 = \text{mon}\_\text{ntrumul}(r, \ h_{2p}, q_p) + g_1 \mod q_p \)

The above process needs to be repeated by B to transmit \( g_2 \) in encrypted form to A.

11. Upon receiving the values of encrypted parts of \( g \), A and B will need to decrypt to obtain the parts of \( g \) of the other participant using the following steps

a. Upon receiving the ciphered text \( C_j \), compute \( a \) using the equation given below

\( a = \text{mon}\_\text{ntrumul}(C_i, \ f_{jp}, q_p) \)

b. Express the coefficients of \( a \) in the range \(-q/2\) to \( q/2\)

c. Compute \( b = \text{mon}\_\text{ntrumul}(a, \ f_{jpp}, p_p) \)

d. The value \( b \) computed is the value of \( g_j \) sent by party \( j \)
i, j will assume values of 1 and 2

12. Each participant calculates \( g = n^g \), for \( i = 1 \) to \( 2 \).

13. Each party now calculates the public key \( h = p \cdot f_i \cdot g \mod q \).

14. The private key of A is \( (f_{p1}, f_1) \), and that of B is \( (f_{p2}, f_2) \)

Algorithm 5.4

5.3.2 Efficient Encryption Using Shared Key with NTRU

Any message \( m \) that is expressed as a polynomial \( m \) could be encrypted using the public key \( h \) as follows

\[
E = \text{mon_ntrumul}(r, h, q) + m \mod q
\]

Where \( r \) is a randomly chosen polynomial, \( p \) and \( q \) are agreed upon moduli.

5.3.3 Decryption Using Shared Key NTRU with Montgomery

The ciphered text \( e \) is sent to both A and B.

1. Let each participant (i.e. A and B) choose random small polynomials, \( r_1 \) and \( r_2 \).

2. Let A calculate \( z_{a1} = f_1 \cdot (e + r_1) \), and sends it to B by following the steps illustrated below

   a. Let A choose random polynomial \( r_a \)

   b. Compute \( e = \text{mon_ntrumul}(r_a, h_2, q_p) + z_{a1} \mod q_p \)
These values are then sent to B

B on the other side will perform a similar computation to
compute \( z_{b1} = f_2^* (e + r_2) \) and by similar procedure send it to A

3. B upon receiving this encrypted e will retrieve \( z_{a1} \) by using the following steps

a. Upon receiving the ciphered text e, compute

\[ a = \text{mon}_n\text{trumul}(e, f_{2p}, q_p) \]

b. Express the coefficients of a in the range \(-q/2\) to \(q/2\)

c. Compute \( b = \text{mon}_n\text{trumul}(a, f_{2pp}, p_p) \)

d. The values of b is the value of \( z_{a1} \).

Similarly A will retrieve the value of \( z_{b1} \)

After decryption \( z_{a1} \) will be known by B and \( z_{b1} \) by A

4. Now A will calculate \( z_{a2} = \text{mon}_n\text{trumul}(f_1, z_{b1}, q) \)

B will calculate \( z_{b2} = \text{mon}_n\text{trumul}(f_2, z_{a1}, q) \). A will then need to adjust the coefficients of \( z_{a2} \) in the range of \([q/2, -q/2] \). B will also need to repeat the same with \( z_{b2} \).

5. A will then calculate \( z_{a3} = z_{a2} \mod p \) and B will calculate

\[ z_{b3} = z_{a2} \mod p. \]

6. A will calculate \( z_{a4} = z_{a3} \ast f_{p1} \) and B will calculate \( z_{b4} = z_{b3} \ast f_{p2} \). Then these values of \( z_{a4} \) and \( z_{b4} \) will need to be exchanged between A and B,; these exchanges are to take place using the steps given in step 2.
7. A will retrieve the message \( m \) by computing \( m = f p_1 \ast z_{b_4} - r_1 \) and B will retrieve the message \( m \) by computing \( f p_1 \ast z_{a_4} - r_2 \).

Algorithm 5.5

This chapter has introduced mechanisms for speeding up the key creation, encryption and decryption procedures for shared key using the much both the RSA and NTRU algorithms.