CHAPTER 5

ERROR DETECTION AND CORRECTION FOR CENTRALIZED KEY MANAGEMENT

Provision of security and quality of service are important challenges to be addressed in multicast key management and distribution protocols. The basic security requirement in a secure group communication includes group confidentiality which can be supported by including two types of security services namely forward secrecy and backward secrecy. The service quality can be integrated into the centralized key management schemes by including suitable encoding and decoding methods that deals with the problem of distributing group keys for secure group communication over an unreliable network which may lead to packet loss. The main advantage of including encoding and decoding methods in key distribution schemes is that the users, in a large and dynamic group communication over an unreliable network, can recover lost rekeying information on their own without requesting additional transmissions from the GC. This reduces network traffic and the risk of user exposure through traffic analysis and also decreases the work load of the GC. In this research work, secure communication is achieved through the use of effective key management techniques supported by error detection and correction methods.

The key idea of the error detection and correction method proposed in this thesis are based on the LDPC error correction codes proposed by Johnson et al (2003), Freundlich et al (2007), Jin et al (2010). LDPC is an error
correcting code that constructs a parity check matrix $M$, which is multiplied with the original data words, $d$ to provide a list of code words, $c$. If the original data word consists of 8 bits, then LDPC (8, 16) parity check matrix is generated. LDPC codes can also be described by their parity check matrix (Tanner et al 1981) or Tanner graphs. Hence, this approach provides a good approach for the group members to construct the original key even if the keying/rekeying information that are sent through the wireless channel are lost. In this proposed work, the time complexity of error correction procedure is significantly minimized.

5.1 KEY DISTRIBUTION PROTOCOL BASED ON GREATEST COMMON DIVISOR AND EULER’S TOTIENT FUNCTION

In this work, a new Key Distribution Protocol Based on GCD and Euler’s Totient Function has been proposed and implemented. It consists of three steps including GC initialization, member initial join and rekeying. For GC initialization and member initial join, the procedure used in section 4.2.1 and 4.2.2 are applied. After completing the first two steps the third step called as rekeying process is carried out as shown in section 5.1.1.

5.1.1 Rekeying Process

Whenever some new members join or some old members leave the multicast group, the GC needs to distribute a new GK to all the current members in a secure way with minimum computation time. When a new member joins into the service it is easy to communicate the new GK with the help of old group key. Since old group key is not known to the new user, the newly joining user cannot view the past communication. This provides backward secrecy. Member Leave operation is completely different from member join operation. In member leave operation, when a member leaves from the group, the GC must avoid the use of old GK or SGK to encrypt new
GK or SGK. Since old members, knows old GK or SGK, it is necessary to use each user’s secret key to perform rekeying information when a member departs from the services. In the existing key management approaches, this process increases GC’s computation time since aims only the security level. However, from the literature study (Jung-Yoon Kim and Hyoung-Kee Choi 2010, Yan Sun 2009) the security levels achieved in the existing works are not sufficient with the current computation facilities. Therefore this work focuses on increasing the security level as well as attempts to reduce the computation time.

The GC executes the rekeying process in the following steps:

1. GC defines a one way hash function \( h(k_i, y) \) where \( k_i \) is the user’s secret information, \( y \) is the users public information and computes the hash value as shown in Equation (5.1).

\[
h(k_i, y) = y^{k_i} (mod \ p) \tag{5.1}
\]

2. GC computes Euler's Totient function (William 2011) for the user \( u_i \) using the function \( f(k_i, y) \) as shown in Equation (5.2). Next, it computes \( f(k_i, y) \) for the user \( u_j \). Similarly, it can compute Euler's Totient function value for ‘n’ numbers of user if the message has to be sent to ‘n’ numbers of user.

\[
f(k_i, y) = \varphi(h(k_i, y)) \tag{5.2}
\]

3. It defines a new function \( g(k_i, y) \) which is obtained by appending with the GCD as shown in Equation (5.3). If the total number of values produced by the function \( f(k_i, y) \) is one, then by default its GCD value is considered as 1. If \( 1 = \gcd(f(k_i, y), f(k_j, y)) \), then \( f(k_i, y) \) and \( f(k_j, y) \) are said to be relatively prime. In such cases, the security level is
equivalent to the security level provided in (Wade Trappe et al 2003).

\[ g(k_i, y) = \gcd \left( f(k_i, y), f(k_j, y) \right) || f(k_i, y) \quad (5.3) \]

The purpose of concatenating the value of \( \gcd \left( f(k_i, y), f(k_j, y) \right) \) with \( f(k_i, y) \) is to increase the security and each user can recover the original keying information.

4. GC computes the new keying information \( \gamma_g(t) \) for the new group key \( K_g \) to be sent to the group members as shown below.

\[ \gamma_g(t) = K_g(t) + \prod_{i=1}^{n-1} \left( g(k_i, y) \right) \quad (5.4) \]

5. GC sends the newly computed GCD value and \( \gamma_g(t) \) to the existing group members.

Upon receiving the GCD value and the encoded information from the GC, an authorized user \( u_i \) of the current group executes the following steps to obtain the new GK.

1. \( \gamma_g \): Calculate the value \( h(k_i, y) = y^{k_i \mod p} \) where \( K_i \) is user’s secret key and \( y \) is the old keying information which is known to all the existing users.

2. Compute \( f(k_i, y) = \varphi \left( h(k_i, y) \right) \)

3. Append the received GCD value from GC in front of \( f(k_i, y) \).

4. A legitimate user \( u_i \) may decode the rekeying information to get the new group key by calculating the following value.

\[ (t) \mod \left( \text{Gcd value} \ || \ f \left( k_i, y \right) \right) \quad (5.5) \]
This proposed method uses the Euler’s Totient Function together with GCD function for improving security and computation time. This helps to increase the key space in this research work by ten times in comparison with the original key space indicated in the existing approaches (Wong et al 2000, Wade Trappe 2003). Thus, it provides more difficulty for an adversary to break the keying information.

5.2 ERROR CORRECTION USING LDPC CODES

This section discusses the error detection and correction methods used for detecting and correcting the errors that happen during the dissemination of keying information in distributed key management protocol.

5.2.1 Encoding at Sender

Encoding process at sender consists of four phases.

Phase 1: Conversion of original key information bits into binary values (0’s and 1’s)

Phase 2: Construction of parity check matrix according to size of the key

Phase 3: Construction of encoding stopping set

Phase 4: Generation and distribution of code words to group members

Consider an example, where the size of key information is 8 bits. If the original key information is 8 bits [1 1 0 1 0 0 1 1] and its corresponding (8, 16) parity check matrix will be generated as mentioned in Phase 2 and used as shown in Figure 5.1. The group members are required to use the same size parity check matrix. From the parity check matrix, the sender can construct the Tanner graph. The algorithm converts the Tanner graph into
pseudo tree based encoding stopping set with maximum bit node degree 3 as explained in (Jin Lu et al 2010).

$$\begin{array}{cccccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
\end{array}$$

**Figure 5.1 Parity Check Matrix of (8, 16) LDPC Codes**

**Re-evaluated bits selection**

The reevaluated bits r1 and r2 are found in a twofold constraint encoding stopping set with key check nodes C7 and C8. The Key parity check Equations for the check nodes C7 and C8 are computed by using Figure 5.3.

$$C7 = X11 \oplus X16$$

$$C8 = X4 \oplus X6 \oplus X12 \oplus X15$$

**Encoding Process**

The stages of encoding are given below:

1. Fill the values of the information bits in the bottom most level, i.e., $[X5 \ X6 \ X7 \ X10 \ X11 \ X12 \ X14 \ X15 ] = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1]$. Initially assign $X4 = 0$ and $X16 = 0$. 

2. Encode the pseudo tree as shown in Figure 5.2 and compute the parity bits as follows

\[ X_8 = X_6 \oplus X_{10} \oplus X_{14} \oplus X_{16} = 1 \]
\[ X_1 = X_7 \oplus X_{14} = 1 \]
\[ X_9 = X_5 \oplus X_8 \oplus X_7 \oplus X_{15} = 1 \]
\[ X_3 = X_6 \oplus X_1 \oplus X_7 \oplus X_{10} \oplus X_{11} \oplus X_{12} = 0 \]
\[ X_{13} = X_4 \oplus X_8 \oplus X_1 \oplus X_5 \oplus X_{10} \oplus X_{11} \oplus X_{15} \oplus X_{16} = 0 \]
\[ X_2 = X_4 \oplus X_9 \oplus X_5 \oplus X_3 \oplus X_{13} \oplus X_{12} \oplus X_{14} = 1 \]
3. Compute the values of key parity check Equations C7 and C8 for the diagram shown in Figure 5.3.

\[ C7 = X11 \oplus X9 \oplus X13 \oplus X16 \oplus X2 = 0 \]

\[ C8 = X4 \oplus X2 \oplus X3 \oplus X6 \oplus X12 \oplus X15 = 1 \]

4. Since \( C7 = 0 \), \( C8 = 1 \), correct value of re-evaluated bits \( X16 = 1 \), \( X4 = 0 \).

5. Compute all the parity bits again based on the new values of \( X16 \) and \( X4 \). The encoded code word is \([ X5 \ X6 \ X7 \ X10 \ X11 \ X12 \ X14 \ X15 \ X4 \ X16 \ X8 \ X1 \ X9 \ X3 \ X13 \ X2 ] = [1 \ 1 \ 0 \\
1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 ]\)

5.2.2 Decoding at Group Members Side

Error correction process at each group member’s area consists of four phases.

Phase 1: Receive the code words from the sender or support node.

Phase 2: Construction of encoding of stopping set as shown in Figure 5.3 according to the parity check matrix used by sender or support node.

Phase 3: Detection of errors by verifying the check node values.

Phase 4: Correction of errors.

Decoding Process

The code word that is received from the sender is \([ X5 \ X6 \ X7 \ X10 \ X11 \ X12 \ X14 \ X15 \ X4 \ X16 \ X8 \ X1 \ X9 \ X3 \ X13 \ X2 ] = [1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 ]\). After the receipt of the code word, the group members should place the values in the encoding stopping set, and also should verify
for the occurrence of errors. On encountering an error, the group members should find out the type of error where in the error can be a single bit error, two bit error...n bit error. For any type of errors, there are 3 cases available for the correction of errors and those cases are explained below.

**Case 1: Re-evaluation Bit**

The errors in this case are found to be in the reevaluated bit. For example, \([X5 \ X6 \ X7 \ X10 \ X11 \ X12 \ X14 \ X15 \ X4 \ X16 \ X8 \ X1 \ X9 \ X3 \ X13 \ X2] = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0].\) The reevaluated bit (X16) value 1 is changed to 0. This is a single bit error type. During the decoding process parity bit values are computed, and during such computation X16 = 1. This does not coincide with the received codeword, where the X16 bit value is 0. Hence, in such a scenario the single bit error has been found. After the conclusion of the occurrence of error, during the correction of errors the reevaluated bits are inverted as \([X4 \ X16]=[0 \ 1].\) Even now if the error is not corrected then perform two bit error correction process. Again, the parity bit values are computed from bottom to top (i.e.) calculate all the parity bit values up to reach the last check nodes. If the key check node \([C7 \ C8]\) values after parity computation are \([0 \ 0],\) then error has been rectified.

**Case 2: Key Information Bits**

This is the case, where the error has been occurred in the original key information bits or aggregation of information bits and reevaluated bits. Considering the example of received code words as follows, \([X5 \ X6 \ X7 \ X10 \ X11 \ X12 \ X14 \ X15 \ X4 \ X16 \ X8 \ X1 \ X9 \ X3 \ X13 \ X2 ] = [1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0],\) X7 information bit value 0 is changed to 1.
During the decoding process some parity bit values will be changed and hence the key check node values will not end up as [0 0]. Hence, on finding such error, the correction follows the steps given below:

Step 1: For correcting the error, all combination of reevaluated bits are changed and even after changing if the key check node [C7 C8] values does not become [0 0], migrate to step2.

Step 2: This is the final stage of correction wherein, each information bit from left to right in the leaf node are changed until the key check nodes [C7 C8] value becomes [0 0], and the error has been rectified. There are also some cases, where even during such a change the key check node values may not become [0 0] and in such a situation, combination of two, three…n information bits are changed in leaf node from left to right to obtain the key check node value to be [0 0].

Case 3: Parity Bits

In the second case, the error would have occurred in the parity bits: for example, considering the bits received are [X5 X6 X7 X10 X11 X12 X14 X15 X4 X16 X8 X1 X9 X3 X13 X2 ] = [1 1 0 1 0 0 1 1 0 1 0 1 1 1 1 0]. In this example X9 parity bit value 0 is changed to 1.
During the decoding process while calculating the parity bit values, X9 = 0 will be obtained which is not in coincidence with the received codeword, where X9 bit value is 1. This error can be rectified automatically while correcting the information bits. The following proof gives the information regarding the number of changes for the different types of errors.

**Lemma:**

Any arbitrary LDPC codes has $O(n)$ time complexity during decoding process for n bit errors.
**Proof:**

Let ‘s’ be the number of leaf nodes which includes ‘n’ information bits and ‘r’ re-evaluated bits received from the sender. The received bits are substituted in the encoding stopping set generated at group member’s side. Now the decoding process is applied in encoding stopping set.

An error is said to occur:

1. If the values of the level 1 check nodes (i.e., key check nodes) are not zero.

2. If the computed parity bit values and received parity bit values at each level, in encoding stopping set are unequal.

It is necessary to correct these errors. In case, if the re-evaluated bits are corrupted, then complexity of correcting the re-evaluated bit is $O(3)$. Depending upon the number of encoding stopping sets the complexity may increase. If there are two encoding stopping set then the decoding time complexity is $O(6)$, in which $O(3)$ for first encoding stopping set and another $O(3)$ computation for second encoding stopping set and so on. On occurrence of error in the information bits, the following procedure has to be followed. Since the number of corrupted bits and their position are unknown, correct them in a step by step procedure. First the 1st bit of the leaf nodes are changed from left to right. Next, the new parity bit values are computed. If the key check node values are equal to zero, then the error is corrected.

Even now, if the key check node values are unequal to zero, then the second bit of the information bit is changed and the procedure is repeated until reaching the last information bits in the leaf level is reached. From this, it is very clear that the complexity for correcting one bit error is $O(n)$. If still
error persists, the above procedure is repeated for all combination of two information bits. Now the time complexity becomes $O(n+(n(n-1)/2))$. Even then if the error is uncorrected, then the combination of ‘i’ $(i=3,4,\ldots,n)$ information bits are changed to calculate the new parity bit value, and the error is corrected. Hence, the time complexity for the decoding procedure is $O(n^2)$ as follows. For example, if the total number of received information bits is 4 bits and all the four information bits are corrupted, then the decoding time complexity can be computed as shown below.

$$= n + (n (n - 1)/2) + (n (n - 2)/2)+1$$

$$= n+((n^2 - n)/2) +((n^2 - 2n)/2)+1$$

$$=(2n^2 - n+2)/2 = O(n^2)$$

5.3 IMPLEMENTATION AND RESULTS

The proposed error correction method for centralized key management scheme has been implemented in Java for 8-bit key values. In these experiments, some errors have been introduced manually in the transmission side introduced manually. The errors are detected and corrected in this implementation using encoding stopping set in the receiver side. The screen shots shown in Figures 5.4 to 5.14 are the implemented and tested result of the proposed approach.
Figure 5.4 Parity Check Matrix Construction

Figure 5.5 Tanner Graph Construction
Figure 5.6 Encoding Stopping Set Construction

Figure 5.7 Encoding Process
Figure 5.8 Encoding Stopping Set with No Error

Figure 5.9 Decoding with Single Bit Error
Figure 5.10 Encoding Stopping Set with Single Bit Error

Figure 5.11 Single Bit Error Correction
Figure 5.12 Single Bit Error Correction in Encoding Stopping Set

Figure 5.13 Encoding Stopping Set with Multiple Bit Error
5.4 CONCLUSION

In this research work, new error detection and correction methods based on LDPC have been proposed and implemented. From the experimental results shown in this chapter, it can be observed that the main advantages of this proposed method is that it helped to enhance the security of multicast communication by detecting the errors occurred in the sender side and by correcting them in the receiver side. This leads to improvement in QoS since it reduces the number of retransmission and hence reduces bandwidth utilization.