Elliptic Curve Cryptography based Threshold Cryptography (ECC-TC) Implementation for MANETs

Levent Ertaul[†] and Nitu J. Chavan[††],
levent.ertaul@csueastbay.edu nchavan@gmail.com
California State University, East Bay, Hayward, CA, USA

Summary
A Mobile Ad hoc Network (MANET) consists of multiple wireless mobile devices that form a network on the fly to allow communication with each other without any infrastructure. Due to its nature, providing security in this network is challenging. Threshold Cryptography (TC) provides a promise of securing this network. In this paper, our purpose is to find most suitable ECC algorithm compared to RSA. Through our implementation of Elliptic Curve Cryptography -based Threshold Cryptography (ECC-TC), we have explored three most-efficient ECC encryption algorithms and put forth possibility of using these ECC-TC algorithms in different scenarios in a MANET. We compare all ECC-TC results and suggest an algorithm that would be most suitable for MANET. Finally, we put forth a new secret sharing alternative that limit communication overheads for transmitting multiple secrets at the same time.

Key words:
Threshold Cryptography, Elliptic Curve Threshold Cryptography, Security in MANETs.

1. Introduction

Mobile ad hoc network (MANET) is vulnerable to various attacks including denial-of-service attack because of wireless nature of this network [1], [2], [3], [4]. Devices with constraint resources add to its vulnerability. To ensure availability of nodes, threshold cryptography can be implemented in the network so that even if some of the information is lost still the actual message reaches the intended receiver without compromising security in terms of confidentiality, integrity, and authenticity.

Threshold cryptography (TC) involves the sharing of a key by multiple individuals engaged in encryption or decryption or splitting of message either before or after encryption. The TC avoids trusting and engaging just one individual node for doing the job. Hence, the primary objective is to share this authority in such a way that each individual node performs computation on the message without revealing any secret information about its partial key or the partial message. Another objective is to have distributed architecture in a hostile environment. A certain number of nodes called threshold, *t* are required to encrypt and/or decrypt a message. Thus, the TC enhances security till compromised nodes are less than *t* since it is difficult to decode partial messages if the number is less than the threshold [5], [6], [7], [8], [9], [20].

Threshold cryptography achieves the security needs such as confidentiality and integrity against malicious nodes. It also provides data integrity and availability in a hostile environment and can also employ verification of the correct data sharing. All this is achieved without revealing the secret key. Thus, taking into consideration these characteristics, implementing TC to secure messages seems a perfect solution in MANET.

Table 1: Key Sizes in Bits for Equivalent Levels

Symmetric	ECC	DH/DSA/RSA
80	163	1024
128	283	3072
192	409	7680
256	571	15,360

Table 2: Sample ECC Exponentiation over GF(p) and RSA Encrypt./Decrypt Timings in Milliseconds

	163 ECC	192 ECC	1024 RSAe	1024 RSAd	2048 RSAe	2048 RSAd
Ultra SparcII 400MHz	6.1	8.7	1.7	32.1	6.1	205.5
Strong ARM 200MHz	22.9	37.7	10.8	188.7	39.1	1273.8

ECC: rG operation, RSAe: RSA Public key operation, RSAd: RSA Private key operation.

RSA based TC has been implemented in computer networks to provide security solutions against various attacks e.g. threshold authentication [19]. These nodes have large storage capacity and computational power. In this paper, we discuss in brief why RSA based TC, commonly used in these networks, is unsuitable for MANETs. Elliptic curve cryptography has gained attention in recent years due to ability to provide equivalent security as RSA but at much smaller key sizes and at fast rates as seen in Table 1 [10]. ECC has been considered for applications such as smart card encryption due to less storage requirements and its computational efficiency [10] as seen in Table 2. Hence, we have selected three best algorithms that are ECC-based and can be implemented for TC. We make a case why and which ECC-based algorithms for TC will be more appropriate for

Manuscript received April 5, 2007
Manuscript revised April 25 2007

MANETs.

For all ECC-TC algorithms, multiple secrets are required to be transmitted over the network. But the packet size varies depending on the implemented algorithm thus adding communication overheads. To solve this problem, we propose a solution for sharing up to 4 secrets which results in constant packet size irrespective of the algorithm.

In next section, we briefly discuss our RSA-TC implementation using partial encryption i.e. encryption key is split and its performance results. Further, changes are suggested in RSA-TC implementation by splitting message after encryption to compare these results with ECC-TC.

2. RSA-TC Implementation

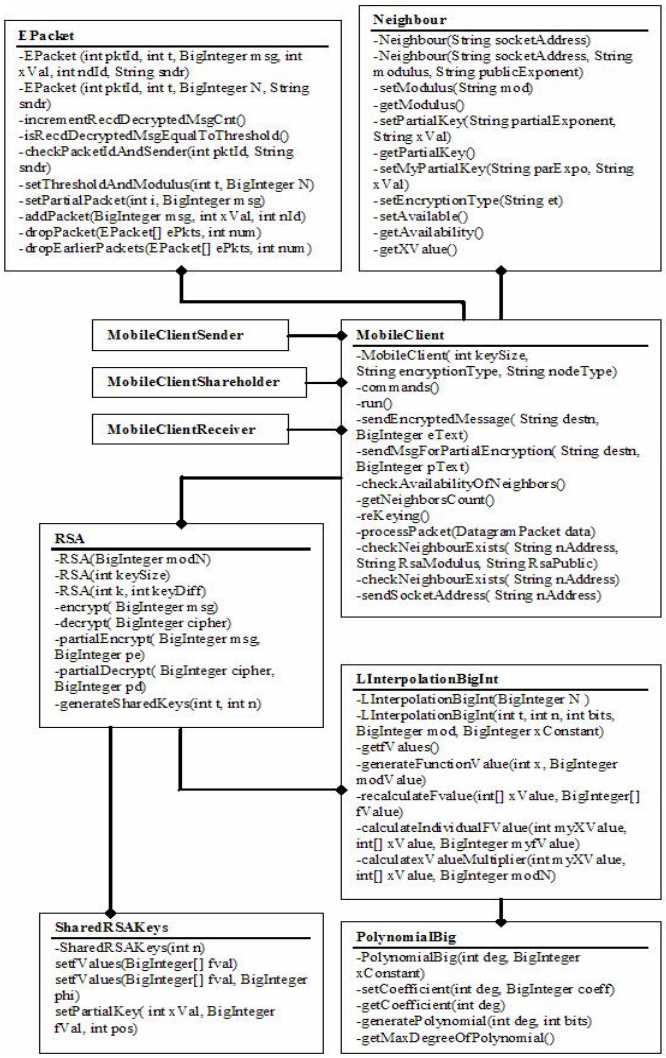


Fig. 1. Class Structure of MANET implementation with RSA-TC

Figure 1 represents *JAVA 1.4* implementation of MANET and its class hierarchy. All classes except RSA and Shared Keys form a basic infrastructure for a MANET node. PolynomialBig class generates and stores coefficients for a polynomial used to generate shares using Lagrange interpolation. LInterpolationBigInt class implements Lagrange interpolation scheme. From a secret, a generated polynomial, and a set of x-values, partial shares are derived. It also retrieves a secret when given a set of x-values and corresponding partial messages. Neighbour class stores information of each neighbour in the MANET such as encryption algorithm type, public key, threshold t, n, and x-value along with partial shared key for RSA-TC. EPacket class is instantiated only at the receiver where it stores partially encrypted messages along with encryption algorithm, public key, x-values, corresponding neighbour/shareholder, sender, packet id, threshold t, and n. MobileClient is the base class for all types of nodes in a MANET i.e. MobileClientSender, MobileClientReceiver, and MobileClientShareholder. SharedKeys class stores information of partial keys and its shareholder/neighbour at the sender. This class is instantiated within RSA class that carries out RSA keys and partial keys generation and partial encryption and decryption. Each node in the MANET has capability to carry out RSA-TC encryption.

In RSA,

i) $C = M^{d} \bmod N$ and $M' = M = C^{e} \bmod N$

ii) $C = M^{e} \bmod N$ and $M' = M = C^{d} \bmod N$

In RSA-TC authentication/signature scheme,

$C' = \prod_{i=0 \text{ till } i=t} C^{xi * f'(i)} \bmod N$,

where $C_i = C^{xi} \bmod N$,

$f(x) = (a_0 x^0 + a_1 x^1 + \ldots + a_{(t-1)} x^{(t-1)}) \bmod \phi(N)$

and $a_0 = d$

$f'(x_i) = \prod_{j=0, j \neq i \text{ till } j=t} (x_j / (x_i - x_j)) * f(x_i) \bmod \phi(N)$

Thus,

$C' = M^{\{\sum_{i=0, j=0, j \neq i \text{ till } i=t, j=t} (x_j / (x_i - x_j)) * f(xi)\}} \bmod N$

$M' = M = C'^{e} \bmod N = C^{e} \bmod N$

Fig. 2. RSA and RSA-TC using Shamir's Lagrange Interpolation

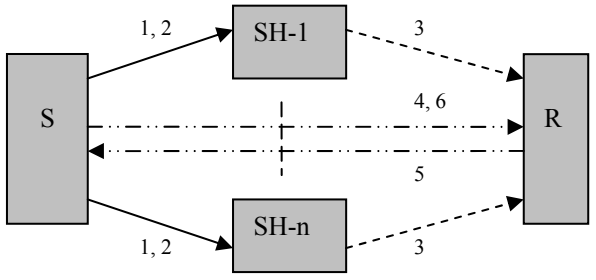
RSA-TC Implementation involved simulation of MANET consisting of a sender S, receiver R, and other *n* nodes called shareholders (SH). MANET was simulated in UNIX environment on *SUN Sparc Ultra 5_10* machines. Figure 2 explains the RSA-TC scheme [11]. Three main modules in this application required were generation of RSA keys, determination of threshold *t*, and share generation.

For RSA key generation, the prime numbers *p* and *q* are generated using available functions in JAVA for key sizes *512*, *1024*, and *2048* bits. Then the private key *(d, N)* and public *(e, N)* are calculated.

The *n* and *t* are fixed to *(10, {6,8,10}), (15, {8, 11, 15}),* and *(20, {11, 15, 20}).*

For RSA-TC, private key *d* is split using Shamir's *t-out-of-n* scheme based on Lagrange interpolation [7] to generate partial keys over modulus $\phi(N)$ such that any *t* out of *n* partial messages will allow retrieval of the original message. These keys are used to carry out partial encryption.

As shown in Figure 3, sender S carries generates partial keys $f(x_i)$ using Lagrange interpolation and polynomial generation over mod $\phi(N)$ as per Fermat's theorem. Sender S retains $\phi(N)$ but distributes partial keys to shareholders SH. The shareholders only apply their partial keys $f(x_i)s$ to the message *M* and forward these partial messages C_is along-with the x_i*-values* to the receiver. After receiving *t* or more C_is, the receiver selects *t* C_is for recovery of *C*. The receiver encrypts x_i*-values* using the sender's public key *(N, e)*, and sends it to the sender via more than one route. The sender calculates respective x_i*'-values* using Lagrange interpolation over *mod* $\phi(N)$ and sends them back to the receiver. The receiver then applies this x_i*'-values* to the respective partial messages and combines the results to recover the final *C*. It then computes C^e *mod N* to recover the final message *M*.



→ One hop between 2 nodes
- - → Multi-hop involves more than 2 nodes on single disjoint route
—·· → Multi-hop involves more than 2 nodes on one or more disjoint routes

1. Sender S distributes the shared keys along with x_i-values amongst its 'n' neighbors which will act as Shareholders SH. ($f(x_i)$ mod ϕ, x_i, N) assigned to each shareholder.
2. S sends message M securely to all SHs for partial encryption.
3. SHs apply $f(x_i)$ to M and send partial encrypted messages as $C_i = M^{f(xi)}$ mod N and x_i to Receiver R.
   Note: A few SHs may not be available or a few messages from SH may be lost during the transmission.
4. S notifies R about threshold t, N and e.
5. R sends selected x_i -values to the S for x_i' values.
6. S calculates x_i' values over mod ϕ(N) and sends them to R. R applies x_i' values to C_is and combines them to get the original C. C^e mod N then gives the message M.

Fig. 3. Model protocol for RSA based threshold encryption in Ad hoc network

In RSA-TC simulation, sender S and receiver R are available at all times. To simulate the propagation delay during network transmission, the messages are randomly delayed at the shareholders, thus, ensuring that set of x_i*'-values* received are always different. Issues of sharing the $\phi(N)$ with shareholders, the storing of message at the shareholders, and number of message exchanged between the shareholders and the receiver are resolved. The sender carries out computation of the x_i*'-values*. Thus, the shareholders need not know *t* and other x_i*-values* that are obtained by the receiver. Instead of sending the x_i*-values* to all the shareholders, the receiver would send it to the sender on multiple reverse routes, less than *t*, thus reducing the message-exchanges carried over the wireless network. Thus, it does not affect the message-exchange even if a few shareholders dropped out of the network after step 3.

2.1 Performance Results

Total RSA-TC encryption timings increased gradually for a given key size with increase in *n* and *t*. As the key-size increased, the encryption time increased exponentially.

Share generation time increased exponentially as the key-size was doubled. These timings included time to generate a polynomial with *t* coefficients and then to calculate *f(x)* for *n* different *x* values. Thus, as *t* value increased the share generation time increased gradually for a key size and *n*.

Combination time is the time required to combine partially encrypted message to retrieve original cipher text. For a given key-size, combination time and decryption time gradually increased with *n* and *t*. Further, increasing key-size results into exponential increase in these timings for a given *n* and *t*.

Success rate increases as *t* increases from *n/2* to *n*. For *t=n*, success rate is 100% [11]. Success rate varies as $\phi(N)$ is even number and all inverses do not exist in mod $\phi(N)$, when $t \neq n$.

The described RSA-TC requires knowledge of $\phi(N)$, to carry out share generation and partial message combination to retrieve cipher-text [11], [12]. Comparing the share generation timings with the actual encryption timings, it is observed that for smaller key sizes the share generation timings are greater or comparable with the encryption timings as *n* increases but for larger key-sizes, share generation takes longer time but is negligible in comparison with encryption time. Further, suggest that success rate cannot be guaranteed for any keys unless implemented.

To achieve 100% success rate in RSA-TC implementation, another method to implement threshold cryptography is to split the message before or after encryption. Results will be similar as above but with 100%

success rate when we implement message split before encryption because partial encryption requires n encryptions and one Lagrange operation [11]. Similarly, RSA-TC with message split before encryption would generate n partial messages using Lagrange interpolation once and then these partial messages are then encrypted using n encryptions.

Given the constraints with RSA-TC, in next section, we would discuss our ECC based threshold cryptography implementation based on three different algorithms.

3. ECC Based TC

Many variants of ECC based algorithms exist such as ECC El Gamal [15], EC Diffie-Hellman [16] (EC-DH), Massey-Omura (MO), Menezes-Vanstone (MV), Koyama-Maurer-Okamoto-Vanstone (KMOV), Ertaul, and Demytko [14]. These variants can be modified to implement ECC-TC in MANET.

Table 3[14] displays performance results for ECC-TC, implemented using Maple. These timings are approximation of results for carrying out point exponentiation and n represents number of shares on which the operation would be carried out for t-out-of-n sharing scheme. Table 4 [14] compares the complexity of all ECC-TC Split before and after encryption algorithms by considering number of times point exponentiation (rG), point addition (P+Q), and Lagrange operations are required. It also lists the number of packets and the packet size to be transmitted over the network for each algorithm. w represents length in bits for the largest number used which is p, so $w = \lceil log(p) \rceil$ [14]. n represents the number of shares a message is split into. Of the above listed three operations, theoretically point exponentiation i.e. rG takes maximum time and resources while point addition takes the least. From Table 3 and 4, DH, MV, and Ertaul have been identified as best ECC-TC algorithms suitable for MANETs. These algorithms are efficient in terms of complexity for both share split before and after encryption and have constant timings irrespective of n and t values.

Our goal is to implement ECC based DH, MV, and Ertaul (most efficient algorithms) for share as well as message splitting before and after encryption in simulated MANET environment. Then we will compare their performances based on timings of different operations that are required for carrying out these encryptions. These timings include timings for splitting the message, converting message to point, and the actual encryption at the sender. At the receiver, timings comprise of combination timings to retrieve the original message from partial messages using Shamir's secret sharing based on Lagrange, to convert point to message and the decryption time.

For ECC-TC, key is not shared here because the public as well as private keys are in form of points and we cannot apply Lagrange on the points altogether to split message or to combine it. Hence, either message is split before encryption and then the partial messages are encrypted into points or the message is encrypted into a point and then the point co-ordinates are split. First, we will briefly study the three ECC-TC algorithms in following sections.

Table 3: ECC secret sharing timings in milliseconds over prime fields

ECC	Share split before encryption				Share split after encryption			
	163-bit Sun	192-bit Sun	163-bit ARM	192-bit ARM	163-bit Sun	192-bit Sun	163-bit ARM	192-bit ARM
EG	18.3n	26.1n	68.7n	113.1n	18.3	26.1	68.7	113.1
MO	24.4n	34.8n	91.6n	150.8n	24.4	34.8	91.6	150.8
DH	6.1	8.7	22.9	37.7	6.1	8.7	22.9	37.7
MV	12.2	17.4	45.8	75.4	12.2	17.4	45.8	75.4
KMOV	12.2n	17.4n	45.8n	75.4n	12.2	17.4	45.8	75.4
Ertaul	18.3	26.1	68.7	113.1	18.3	26.1	68.7	113.1
Demytko	18.3n	26.1n	68.7n	113.1n	12.2	17.4	45.8	75.4

Sun: Ultra Sparc II 450 MHz ***ARM: Strong ARM 200 MHz***

Table 4: Complexity comparison of ECC-TC Encryption/Decryption algorithms

ECC TC Algorithm	Share split before encryption			Share split after encryption			Pkt size	Pkt #
	rG	P+Q	Lag	rG	P+Q	Lag		
ECCEG	3n	2n	1	3	2	2	5w	n
MO	4n	0	1	4	0	6	3w	3n
DH	0	2n	1	0	2	2	3w	n
MV	3	0	1	3	0	2	5w	n
KMOV	2n	0	1	2	0	2	3w	n
Ertaul	3	0	1	3	0	1	4w	n
Demytko	2	0	1	2	0	1	3w	n

Note: Lag= Lagrange Timings

3.1 ECC Diffie-Hellman (ECC DH) Encryption/Decryption Algorithm

ECCDH and its threshold implementation [14] is suggested as follows: The order of a point G on an elliptic curve $E_p(a, b)$ is q. P is a large prime. The secret key $K = n_A n_B G$ is generated using DH algorithm.

Encryption algorithm:

• Alice finds the point P_M corresponding to M, and sends $P_M + n_A n_B G$ to Bob.

Decryption algorithm:

• Bob subtracts $n_A n_B G$ from $P_M + n_A n_B G$, and converts P_M to the plaintext M.

3.1.1 Share split before encryption

- Alice uses Shamir's secret sharing method to split the secret M into n shares of secret M_t, $1 \le t \le n$.
- Alice converts each share M_t to a point P_t on the EC.
- Alice computes $P_t + n_A n_B G$ and sends it to Bob.
- Bob recovers P_t by subtracting $n_A n_B G$ from $P_t + n_A n_B G$.
- With at least t share of P_M, Bob is able to recover P_M.
- Finally Bob will convert the point P_M to the secret M.

3.1.2 Share split after encryption

- Alice converts the secret M to a point $P_M = (x, y)$ on the EC.
- Alice computes $P_C = n_A n_B G + P_M = (x_C, y_C)$.
- Alice uses Shamir's secret sharing method to split x_C and y_C into n shares of x_{C_t} and y_{C_t} respectively, $1 \le t \le n$.
- Alice sends n pieces of x_{C_t} and y_{C_t} to Bob.
- Bob combines t pieces of x_{C_t} and y_{C_t} separately to get (x_C, y_C), i.e. P_C.
- Bob computes $P_M = P_C - n_A n_B G$.
- Finally Bob will convert the point P_M to the secret M.

3.2 Menezes-Vanstone (MV) algorithm

MV [17] is a solution to the problem of encoding a message into a point on EC. It uses a point on an EC to mask a point in the plane. It is fast and simple. Let H be a cyclic subgroup of $E_p(a, b)$ with the generator G. Bob has a private key n_B, and a public key $n_B G$. The message M is converted into a point P_M = (x, y) in $GF(p)$.

Encryption algorithm:

- Alice select a random number r < |H|, and calculates $rn_B G = (x_k, y_k)$.
- Alice sends $(rG, x_k x \bmod p, y_k y \bmod p)$ to Bob.

Decryption algorithm:

- Bob calculates $n_B rG = rn_B G = (x_k, y_k)$.
- Bob recovers x and y by $x_k^{-1} x_k x \bmod p$ and $y_k^{-1} y_k y \bmod p$.
- Bob converts the point (x, y) to get the original plaintext M.

3.2.1 Share split before encryption

• Alice splits the message M into n shares of secret M_t, $1 \le t \le n$.
• Alice converts each share M_t into a point $P_t (x_t, y_t)$.
• Alice select a random number r < |H|, and calculates $rn_B G = (x_k, y_k)$.
• Alice sends $(rG, x_k x_t \bmod p, y_k y_t \bmod p)$ to Bob.
• Bob calculates $n_B rG = rn_B G = (x_k, y_k)$.
• Bob recovers x_t and y_t by $x_k^{-1} x_k x_t \bmod p$ and $y_k^{-1} y_k y_t \bmod p$.
• With at least t shares of P_M, Bob recovers P_M, and converts the P_M to the secret M.

3.2.2 Share split after encryption

• Alice converts the message M into a point P_M(x, y).
• Alice select a random number r < |H|.
• Alice calculates $rn_B G = (x_k, y_k)$, and calculates z = $x_k x$ mod p, and w = y_ky mod p.
• Alice splits z, w into n shares of z_t, and w_t respectively, $1 \le t \le n$.
• Alice sends rG and n pieces of z_t, and w_t to Bob.
• Bob combines t pieces of z_t and w_t separately to get (z, w).
• Bob calculates $\underline{n_B rG} = rn_B G = (x_k, y_k)$.
• Bob recovers P_M by $x_k^{-1} z = x_k^{-1} x_k x \bmod p$ and $y_k^{-1} w = y_k^{-1} y_k y \bmod p$.
• Eventually Bob converts P_M to the secret M.

3.3 Ertaul Crypto-system

P is the generator point while x is the private key, and Y = x*P is the public key.
$H((x_i, y_i)) = Hash(x_i \text{ xor } y_i)$ is a HASH function such as MD5, SHA-1.

Encryption algorithm:

• Alice selects a random value r from Z_q.
• Alice computes U = r*P and V = H(r*Y) xor M, and sends C = (U, V) to Bob.

Decryption algorithm:

• Given a ciphertext C = (U, V), Bob computes x*U= x*r*P = r*x*P.
• Bob computes V xor H(r*x*P) = H(r*Y) xor M xor H(r*x*P) = M.

3.3.1 Share Split Before Encryption

• Alice splits the secret M into n shares of secret M_t, $1 \le t \le n$.
• Alice selects a random value r from Z_q, and computes U = r*P.
• For each share M_t, Alice computes V_t = H(r*Y) xor M_t.
• Alice sends ciphertext $C_t = (U, V_t)$ to Bob.
• Given a ciphertext C_t, Bob computes x*U= x*r*P.
• Bob computes H(r*x*P) and V_t xor H(r*x*P) = H(r*Y) xor M_t xor H(r*x*P) = M_t.
• With at least k share of M_t, Bob is able to recover M.

3.3.2 Share Split After Encryption

• Alice selects a random value r from Z_q, computes U = r*P.
• Alice computes V = H(r*Y) xor M, splits V into n shares of secret V_t, $1 \le t \le n$.
• Alice sends ciphertext $C_t = (U, V_t)$ to Bob.
• Bob recovers V, and computes x*U= x*r*P.
• Bob computes H(x*r*P) and V xor H(x*r*P) = H(r*Y) xor M xor H(x*r*P) = M.

3.4 ECC-TC Implementation Model

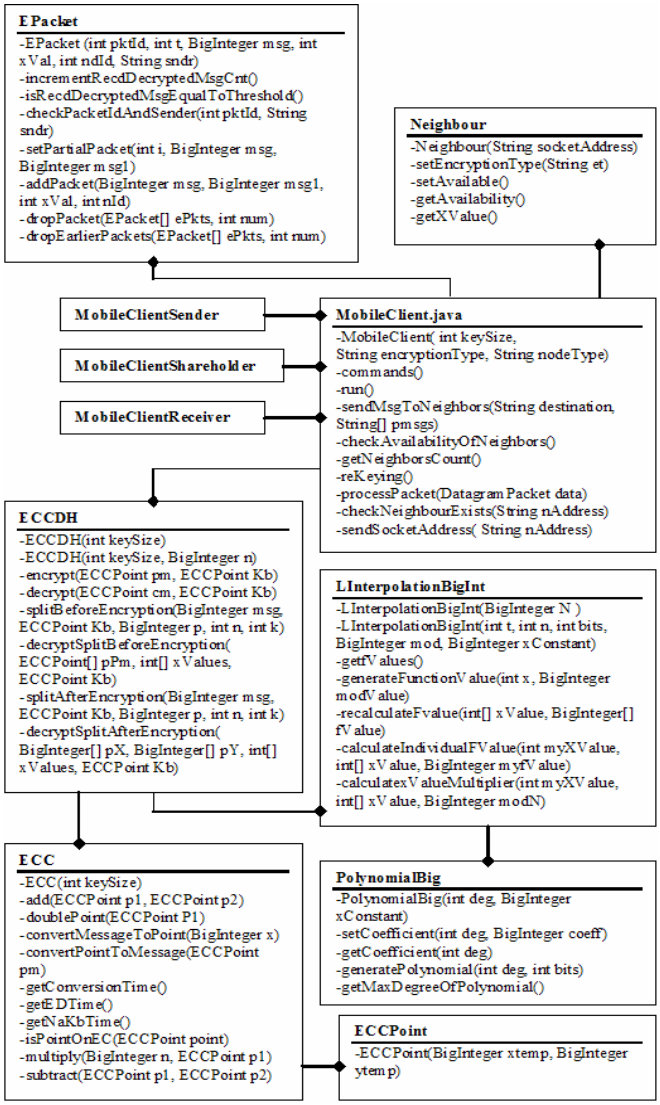


Fig. 4. Class Structure of MANET implementation with ECC-TC

Figure 4 displays class hierarchy of ECC-TC implementation using JAVA 1.4 in Unix environment on SUN Sparc Ultra 5_10(360 MHz) machines. ECC-TC algorithms are implemented in JAVA since it is widely applicable in mobile devices with resource restraints [18]. A MANET with varying node density, n, is simulated with a capability to send messages using earlier mentioned ECC-TC algorithms. As seen in Figure 1 for RSA-TC implementation, the basic MANET infrastructure is same here except the additional classes: ECCPoint, ECC, ECCDH, MV, and Ertaul. ECCPoint class represents a point on elliptic curve and stores its co-ordinates. ECC class implements basic elliptic curve point operations such as point exponentiation and addition, message to point and point to message conversion. As illustrated in Figure 4, ECCDH class carries out ECCDH based threshold split before or after encryption. Likewise, the implementation can carry out Ertaul and MV based threshold encryption.

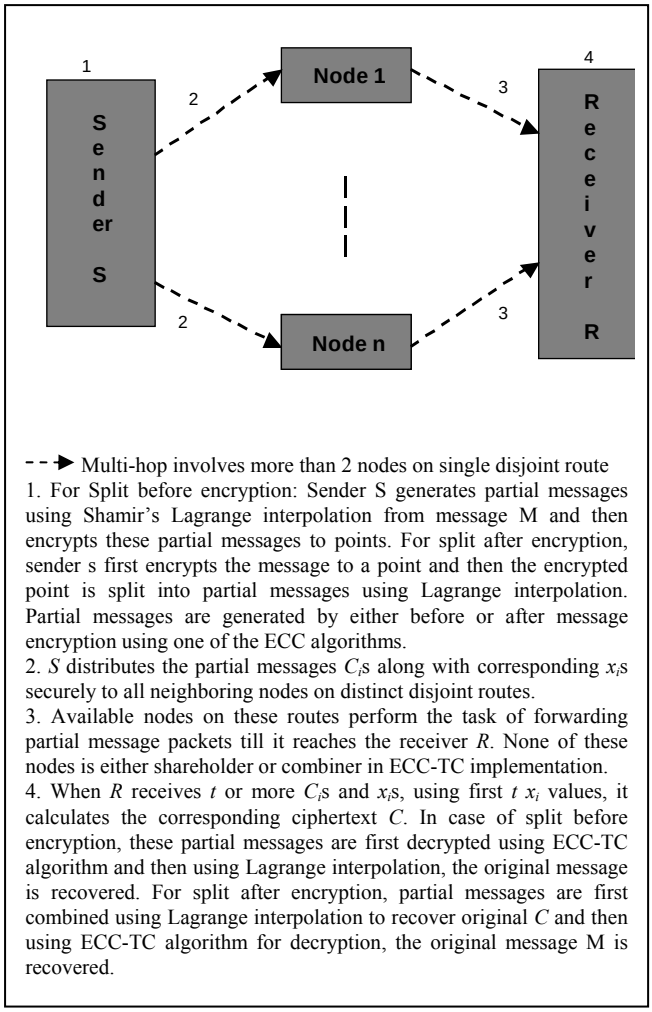


Fig. 5. Model protocol for ECC based threshold encryption in Ad hoc network.

Assumptions during implementation are that in any given scenario, there is a sender S, a receiver R, and multiple nodes on distinct routes that forward the message from S to R. All the nodes are assigned unique ids. Sender S and receiver R are already identified in the network. The key or partial message combination procedure in TC is impacted by the node availability and the connections in the network. So in this model, sender splits the message into number of partial messages while receiver does the job of combining partial messages and retrieving original message. Hence, no separate combiner defined in the network. In addition to this, we consider t or more nodes are always available, so receiver receives t or more partial messages. Multiple disjoint routes already traced. Here we

are not dealing with routing issues, so we assume that the multiple disjoint routes can be identified using any of the available multi-path routing protocols. Additionally, though with multiple partial messages traveling over different routes, we are not working on the communication overhead computations in this implementation. Instead of using multiple random 'r' values for each partial message in split before encryption scenario, a single random value r is used. Thus, the rG or rK_b multiplication timing is reduced drastically by n-1 times for each message exchange.

Figure 5 depicts the ECC-TC model where in sender S generates partial messages using Shamir's Lagrange interpolation and ECC-TC algorithm. For split before encryption, a message M first split into n partial messages that are individually converted to ECC point and then encrypted using one of the three ECC algorithms discussed earlier. But in split after encryption, the message M is first converted to ECC point and encrypted using ECC-TC algorithm. Next, this encrypted information is further split into partial encrypted messages using multiple Lagrange interpolation. Sender thus transmits each partially encrypted message on different route. The nodes on these routes forward these messages to receiver after adding a random delay to simulate propagation delay that ensures that set of t x_i values at the receiver is different each time. When receiver R collects t or more partially encrypted messages, then it recalculates the message M by combining them. For split before encryption, first these messages are individually decrypted, converted from ECC point to M_is and then combined to get M, while in split after encryption, these messages are first combined to recover C and then ECC-TC decryption is carried out to retrieve ECC point which is then converted to M.

3.5 Modules in ECC-TC Implementation

Important modules required to successfully implement ECC-TC are as follows:

3.5.1 Determination of ECC parameters

For implementation of the ECC-TC, widely accepted NIST curves were selected for implementation for 192, 224, and 256 bits [17] as shown below. For each algorithm, further respective parameters are determined beforehand for the sender and receiver.

Curve P-192
p = 6277101735386680763835789423207666416083908 7/00390324961279
r = 6277101735386680763835789423176059013767 1947/73182842284081
s = 3045ae6f c8422f64 ed579528 d38120ea e12196d5
c = 3099d2bb
bfcb2538 542dcd5f b078b6ef 5f3d6fe2 c745de65
b = 64210519
e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1
Gx = 188da80eb03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012
Gy = 07192b95ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811

Curve P-224
p = 26959946667150639794667015087019630673557916/260026308143510066298881
r = 26959946667150639794667015087019625940457807/7144243917216827223680 61
s = bd713447 99d5c7fc dc45b59f a3b9ab8f 6a948bc5
c = 5b056c7e 11dd68f40469ee7f 3c7a7d74 f7d12111 6506d031 218291fb
b = b4050a85 0c04b3abf5413256 5044b0b7 d7bfd8ba 270b3943 2355ffb4
Gx = b70e0cbd 6bb4bf7f321390b9 4a03c1d3 56c21122 343280d6 115c1d21
Gy = bd376388 b5f723fb4c22dfe6 cd4375a0 5a074764 44d58199 85007e34

Curve P-256
p = 1157920892103562487626974469494075735300 8614/
3415290314195533631308867097853951
r = 1157920892103562487626974469494075735299 695/
5224135760342422259061068512044369
s = c49d3608 86e70493 6a6678e1 139d26b7 819f7e90
c = 7efba166 2985be94 03cb055c75d4f7e0 ce8d84a9 c5114abc af317768 0104fa0d
b = 5ac635d8 aa3a93e7 b3ebbd55769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b
Gx = 6b17d1f2 e12c4247 f8bce6e563a440f2 77037d81 2deb33a0 f4a13945 d898c296
Gy = 4fe342e2 fe1a7f9b 8ee7eb4a7c0f9e16 2bce3357 6b315ece cbb64068 37bf51f5

3.5.2 Transformation between Message and ECC points

Table 5: Maximum Message Limits in ECC implementation

Key Size	Maximum Message Limit
192	3753746952096093776249666754145703350284177 65993507829221
224	6236924295242024210552351976708463701875106 196479058055504486591 23
256	1988206018555158328763890072084183569476534 939011764732119112202020563046577

For conversion of message to and from ECC point, method discussed by Kobiltz is used [13], [14] such that (kappa*M)mod p < x <(kappa*(M+1))mod p, where (x, y) is a point on elliptic curve. In our ECC TC implementation, kappa is fixed to 2^8. This is seen to accommodate the possible conversion of the ASCII characters represented as message M into ECC points such that M< Maximum Message Limit value, which is fixed for all ECC key sizes as shown in Table 5.

To retrieve a message from a ECC point (x, y), M=x/kappa mod p

3.5.3 ECC point operations

As discussed earlier, given ECC points, we can carry out point addition or multiplication/exponentiation. These operations are prerequisite for carrying out encryption using ECCDH, MV, and Ertaul algorithms.

3.5.4 Share generation

First *(n, t)* values are fixed to one of the following: (10, {6, 8, 10}), (15, {8, 11, 15}), or (20, {11, 15, 20}). Next,

for calculating the shares and for combining partial messages, Shamir's Lagrange interpolation scheme is implemented. For its polynomial of degree *t-1*, the coefficients are randomly generated over the modulus p. The co-efficient zero depends on the x and y values of ECC point information that needs to be transmitted based on ECC algorithm used. In ECC-TC implementation, the partial shares of the ECC point information are generated by the sender and forwarded via diverse paths to the receiver. Currently, x_i-values used for calculating the shares are 1 to *n,* rather than randomly picking these values.

3.5.6 Performance Results for ECC-TC algorithms

Before discussing performance results, let us first discuss various terms used in the graphs for the performance results:

Point exponentiation time is the time to calculate rn_BG, rG, n_AK_B., or U i.e.(r*P). This is represented by rG or nK_B or U time in the following figures depending on which one is required for a given ECC-TC algorithm.

Conversion time at the sender means time to convert a message to a ECC point. At the receiver, conversion time means time required to convert a ECC point to a message.

Lagrange time at the sender is the time required to split the message into partial messages while at the receiver it is the time required to combine t partial messages to retrieve original message.

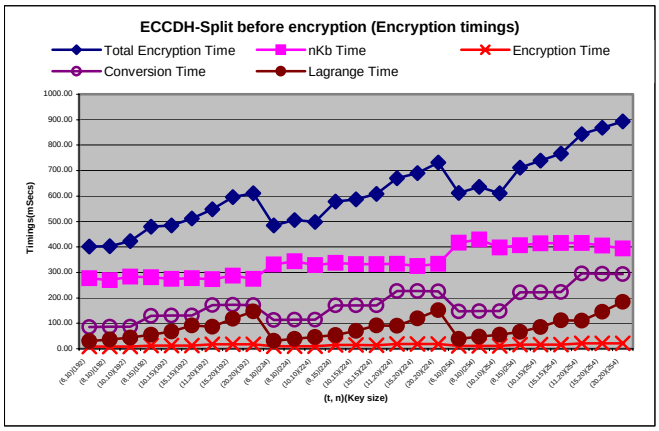
Encryption time is the time required to perform a encryption operation specific to a ECC-TC algorithm. E.g. in ECCDH, point addition encrypts the message ECC point by using operation $P_M + n_An_BG$. So here the encryption time equals to the time required to carry out point addition. Similarly for MV, encryption time is the time required to carry operations x_kx mod p and y_ky mod p. And for Ertaul, it is the time required to carry out XOR operation in H(r*Y)/M.

Further, decryption time is the time required to perform a decryption operation specific to a ECC-TC algorithm. E.g. in ECCDH, point addition decrypts a ECC point to a by using operation $P_M + n_An_BG$. So here the encryption time equals to the time required to carry out point addition. Similarly for MV, encryption time is the time required to carry operations $x_kx\ x_k^{-1}x_k$x mod p and $y_k^{-1}y_k$y mod p. And for Ertaul, it is the time required to carry out XOR operation in V/H(r*x*P).

Total encryption time means the sum of all the operations to encrypt a message in an ECC-TC algorithm, and total decryption time means sum of all the operations taken to decrypt a message.

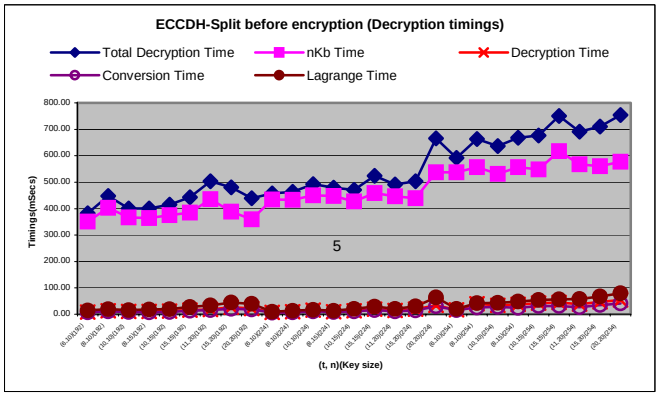
Total timing is the sum of the total encryption and total decryption timings i.e. time required to encrypt a message and to retrieve it back by decrypting it.

Y-axis in the graphs below represents timings in milli-seconds while X-axis represents {(t, n), key size}. As mentioned earlier, *t* and *n* are fixed to *({6,8,10}, 10), ({8, 11, 15}, 15),* and *({11, 15, 20}, 20)* while key sizes are 192, 224, and 254.



Processor: SUN Sparc Ultra 5_10 Timings for 200 runs

Fig. 6. Encryption timings for ECCDH-TC Split before encryption

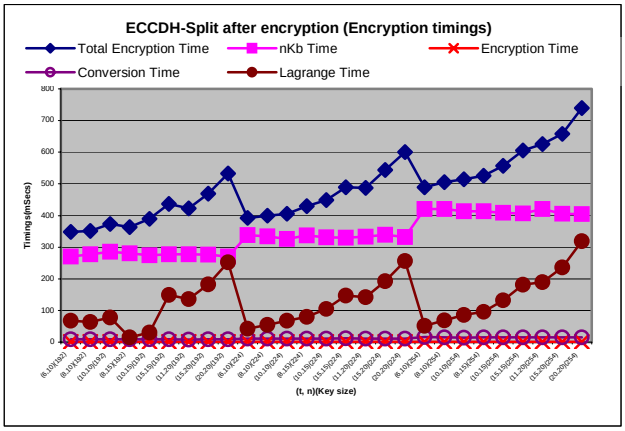


Processor: SUN Sparc Ultra 5_10 Timings for 200 runs

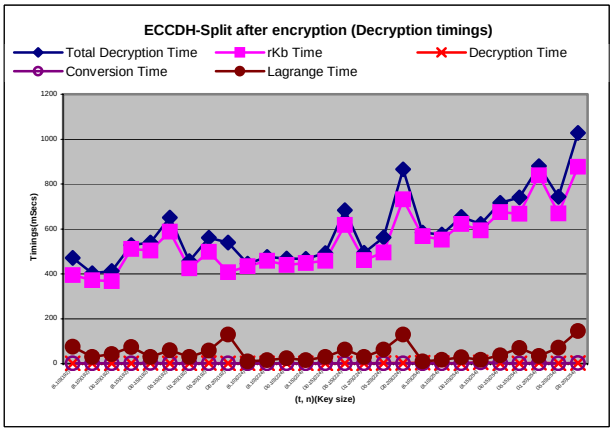
Fig. 7. Decryption timings for ECCDH-TC Split before encryption

As observed from Figure 6 and 7, for ECCDH TC schemes, encryption and decryption timings consist of large naKb timings. From Figure 6 and 7 for split before encryption, conversion timings contribute greatly and for most instances more than Lagrange timings. naKb, Lagrange and conversion timings increase as we increase *t, n* or the key-size.

From Figure 8 and 9 for split after encryption, during encryption as *t* and *n* increases Lagrange timings contributes more than naKb that is constant for any given key size irrespective of *t* and *n* values. For decryption, Lagrange timings are small but increase with *t, n*, and key sizes.

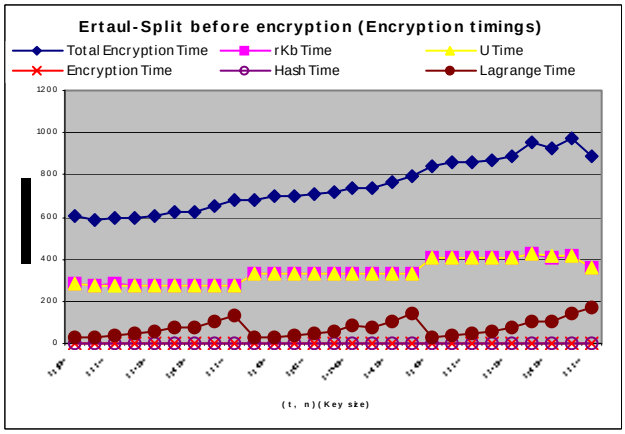


Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
Fig. 8. Encryption timings for ECCDH-TC Split after encryption

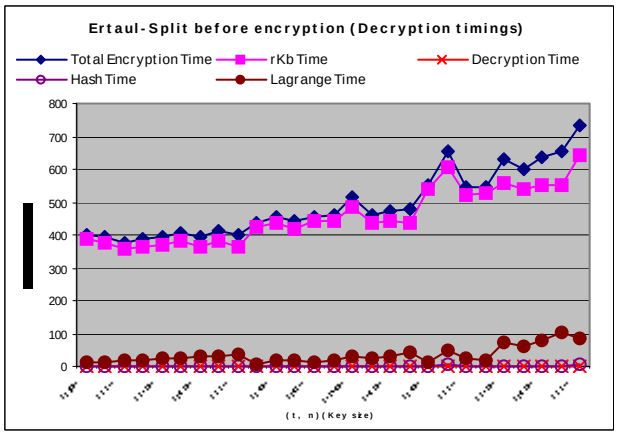


Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
Fig. 9. Decryption timings for ECCDH-TC Split after encryption

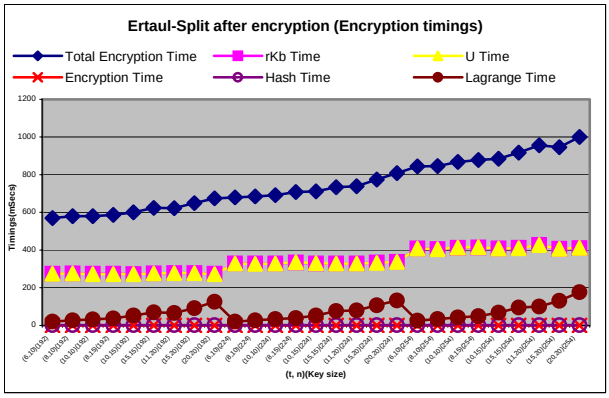
Figure 10 and 11 display encryption and decryption timings for Ertaul-Split before encryption while Figure 12 and 13 display timings for Ertaul-Split after encryption. For Ertaul-split before and after encryption, *rKb* and U calculation timings cost the most for encryption, while Lagrange contributes significantly to it for larger *t* and *n* values for all key-sizes. Hashing and encryption timings are negligible. There is no point conversion in Ertaul-TC scheme, hence the encryption timings are almost similar for both before and after encryption schemes.



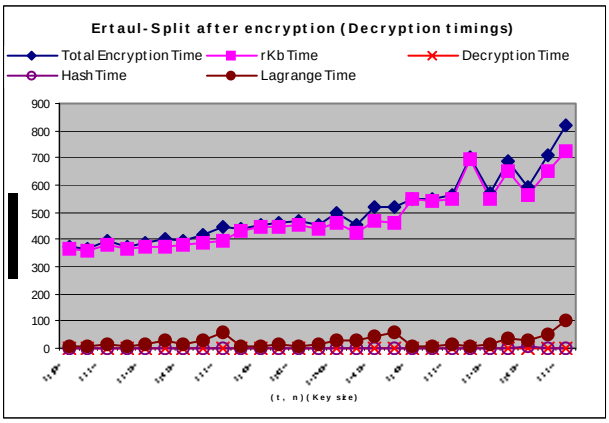
Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
Fig. 10. Encryption timings Ertaul-TC Split before encryption



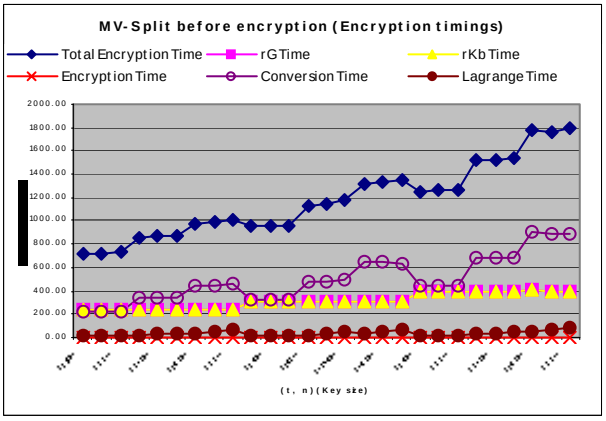
Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
Fig. 11. Decryption timings for Ertaul-TC Split before encryption



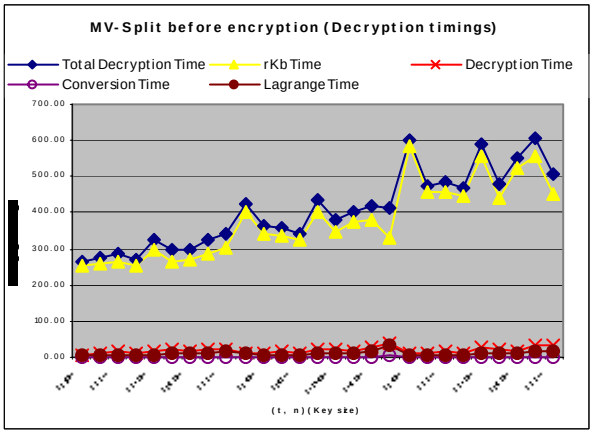
Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
Fig. 12. Encryption timings for Ertaul-TC Split after encryption



Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
Fig. 13. Decryption timings for Ertaul-TC Split after encryption



Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
Fig. 14. Encryption timings for MV-TC Split before encryption



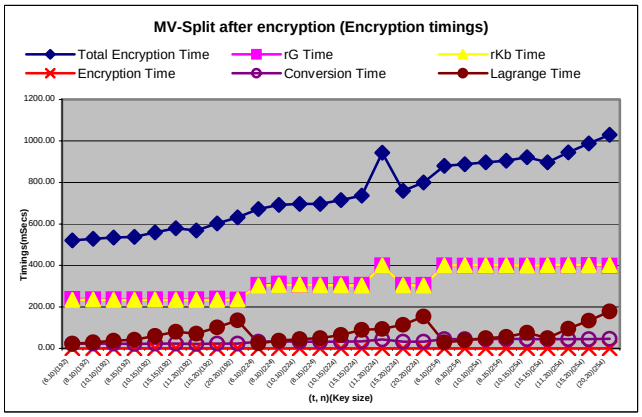
Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
Fig. 15. Decryption timings for MV-TC Split before encryption

From Figure 14 for MV split before encryption, rK_b and rG timings are similar for all *t, n,* and key size. Conversion timings are contributing significantly for this type of encryption and it increases with *t, n,* and key size.

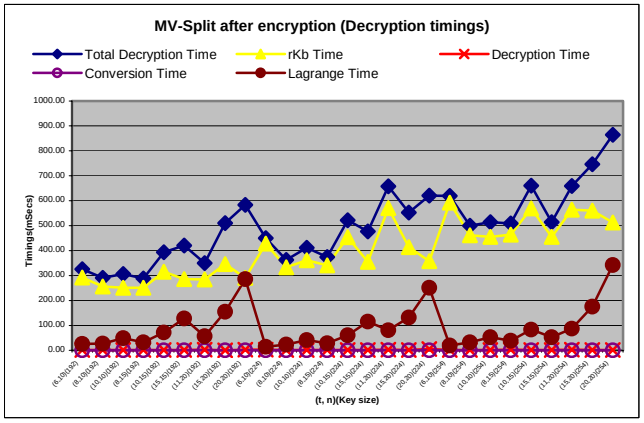
As seen in Figure 15, decryption timings mainly consist of rK_b timings that vary with changing *t, n,* and key-sizes. From Figure 14 and 15, overall encryption, decryption, and Lagrange timings are negligible compared to other timings for this TC scheme.

It is observed in Figure 16 for MV Split after encryption graph that rG and rK_b calculations are almost same and contribute the most to the encryption timings. Next, share splitting using Lagrange interpolation contributes significantly as *t*, *n*, and keysizes increase. Total encryption timings display a gradual increase as *t, n,* and keysizes are increased.

From Figure 17, total decryption timings for MV-split after encryption vary significantly and most of the timings is contributed by rK_b calculation. But as *t* value is increased, the Lagrange timings increase exponentially by contributing significantly at when $t=n$.



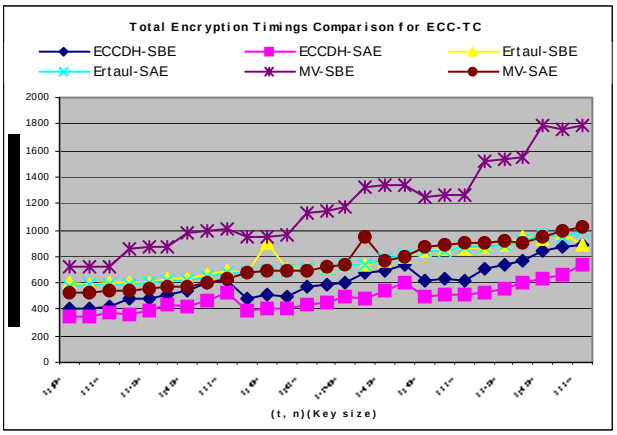
Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
Fig. 16. Encryption timings for MV-TC Split after encryption



Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
Fig. 17. Decryption timings for MV-TC Split after encryption

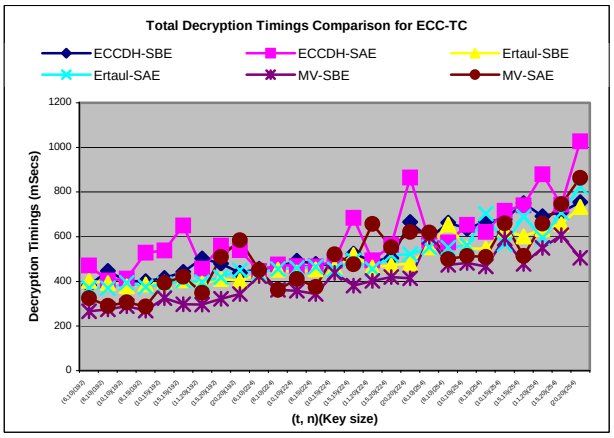
From Figure 16 and 17, overall conversion, encryption, and decryption timings are negligible compared to other timings for MV-split after encryption.

In Figures 18, 19, and 20, we compare total timings for the three ECC-TC based algorithms.



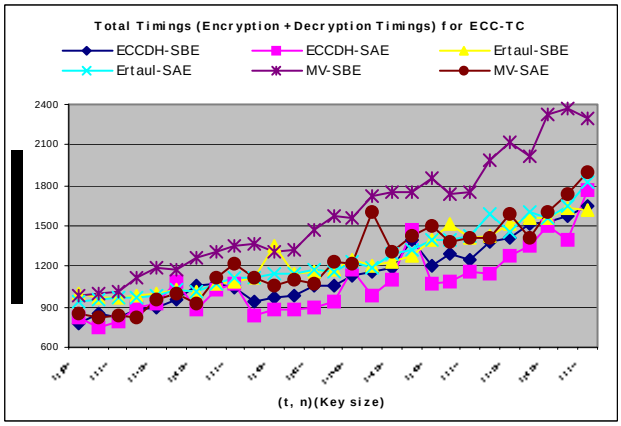
Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
SBE = Share split Before Encryption, SAE = Share split After Encryption
Fig. 18. Total Encryption timings for ECC-TC algorithms

Considering total Encryption timings for all ECC-TC algorithms, it is observed in Figure 18 that with increase in key size and (t, n), the encryption timings increase gradually for all algorithms. ECCDH is most efficient for both split before and after encryptions and hence can be used when sender has resource restraints. As against this, MV seems most inefficient with wide difference in the timings for split before and after encryption timings. For Ertaul, the timings are very close for both split before and after encryption. Thus, from Figure 18, ECC-DH is ideal for scenarios where the sender has resource constraints.



Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
SBE = Share split Before Encryption, SAE = Share split After Encryption
Fig. 19. Total Decryption timings for ECC-TC algorithms

Figure 19 exemplifies that for decryption timings, MV split before encryption seems to be efficient. MV-Split after encryption is efficient at lower *t* and *n* values but as we increase the *t, n,* and key size the decryption timings increase exponentially. Ertaul decryption timings seem to be stable for split before encryption as well as decryption scenarios. As against the above algorithms, ECCDH's decryption timings are worst with exponential rise as *t, n,* key-sizes are increased. Thus, MV is ideal for scenarios where the receiver has more power and CPU constraints while ECCDH is not.



Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
SBE = Share split Before Encryption, SAE = Share split After Encryption
Fig. 20. Total Timings for ECC-TC algorithms

From Figure 20, consider total timings to carry out threshold cryptography i.e. encryption and decryption timings combined. MV seems to be the worst algorithm as the changes in the total timings are exponential and variation increases at higher values of *t, n,* and key-sizes. ECCDH is ideal for lower *t* and *n* values irrespective of key-sizes but at higher *t* and *n* values, Ertaul split before encryption is a better choice.

By and large, it appears that the ECC point multiplication is the only operation that would cost significantly when carrying out ECC encryption or decryption. But from our results, we have proved that for a given key-size, as the *t* and *n* values are increased, ECC point multiplication timings i.e. rG or rK_b or n_aK_b timings remain constant. On the other hand, Lagrange timings start increasing and contribute significantly to encryption and decryption timings for threshold cryptography involving split after encryption as Lagrange is applied to multiple values. Also, for larger *n* values, the conversion timings to convert message to point also add significantly to the encryption time for split before encryption. Comparing results from split before and after encryptions, we have observed that split before encryption is inefficient than split after encryption schemes except for Ertaul where the timings are based more on the *t*, *n*, and key-size values. In Ertaul, the difference in the total timings for both schemes is small as against ECCDH and MV where one can observe vast difference in the timings. Given that the timings in Ertaul scheme are constant for encryption and

decryption and are close, it is ideal scheme to be implemented in a MANET where both sender and receiver have equal resources as well as power constraints.

For ECC-TC, more than one packet of size w is transmitted from sender to receiver in MANETs to be able to achieve threshold cryptography. In next section, we put forth an alternative to Shamir's secret sharing scheme using Lagrange interpolation wherein up to $4w$ information can be sent using just one split.

4. Alternative Secret Sharing Scheme for Threshold Cryptography in MANETs

In ECC, after encrypting a message, multiple point information is sent to receiver. For example, in ECC El Gamal encryption, $\{C_1 = rG,\ C_2 = K_bG + P_m\}$ are transmitted where $C_1=(x_1, y_1)$ and $C_2= (x_2, y_2)$. If threshold cryptography using Shamir's (t, n) secret sharing scheme [5], [6], [7] is used, then either 4 different messages, each representing one of the x or y values and of $3w$ packet size, or a single message, with $5w$ packet size, would be broadcasted over each of the n disjoint paths between the sender and receiver.

In MANET, bandwidth is limited and so we propose using Vandermonde matrix equations [21] with a slight modification for sending messages. The x and y values should be inserted into the polynomial of degree $t-1$ as coefficients a_0, a_{k-3}, a_{k-2}, and a_{k-1}, where $a_0 = x_1$, $a_{t-3} = y_2$, $a_{t-2} = x_2$, *and* $a_{t-1} = y_2$. If threshold $t > 4$, then remaining coefficients of the polynomial are randomly generated. So the polynomial would be:

$$f(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + a_{t-3}x^{t-3} + \ldots + a_1x + a_0 \bmod p, \text{ if } t>4.$$

Thus, sender would calculate $f(x)$ for different x-values and distribute $f(x)$ and corresponding x over n disjoint routes. Given that at least t different messages are received, rather than retrieving all the coefficients using Vandermonde matrix, receiver R would retrieve the C_1 and C_2 values as shown below.

R recalculates $C_1\ (x_1, y_1)$ and $C_2\ (x_2, y_2)$ as below. The message is recovered as $M = C_2 - n_bC_1$.

Let $L_i = \prod_{j=1\ldots t,\ j\,!=i} 1/(x_j - x_i)$, *then*

$$a_0 = x_1 = \sum_{i=1\ldots t} \left(\prod_{j=1\ldots t, j\,!=i} x_j\right) f_i L_i \bmod p,$$

$$a_{t-3} = y_1 = (-1)^{t-3}\,[[(x_2x_3 + x_2x_4 + \ldots + x_{t-1}x_t)f_1 L_1 \bmod p] + [(x_1x_3 + x_1x_4 + \ldots + x_{t-1}x_t) f_2 L_2 \bmod p] + \ldots [(x_1x_2 + x_1x_3 + \ldots + x_{t-2}x_t) f_{t-1} L_{k-1} \bmod p] + [(x_1x_2 + x_1x_3 + \ldots + x_{t-2}x_{t-1}) f_t L_k \bmod p]] \bmod p$$

$$a_{t-2} = x_2 = (-1)^{t-2}[\,[(x_2 + x_3 + \ldots + x_t) f_1L_1 \bmod p] + [(x_1 + x_3 + \ldots + x_t)\ f_2 L_2 \bmod p] + \ldots [(x_1 + x_2 + \ldots + x_{t-2} + x_t) f_{t-1} L_{t-1} \bmod p] + [(x_1 + x_2 + \ldots + x_{t-1}) f_tL_t \bmod p]] \bmod p$$

$$a_{t-1} = y_2 = (-1)^{t-1}[\,\textstyle\sum_{i=1\ldots t} f_i\ L_i \bmod p] \bmod p$$

The advantages of this alternative method are:

Instead of nX packets, where X is the number of x and y values of points to be transmitted, just n packets are distributed, thus, reducing bandwidth and storage consumption by $1/X$ for each message transmission. Also, the retrieval of message is faster since the wait-time between the messages is eliminated.

By adding this scheme once, instead of multiple Lagrange, after or before encryption, the packet size for all the ECC-based schemes can be reduced to be $2w$ while keeping the number of packets sent between sender and receiver nodes to n. In case of MO, the packet-size would be $2w$ but the number of packets exchanged is $3n$.

Based on computational complexity of retrieving the coefficients as discussed above, the order of coefficient selection, for securely sending multiple messages, should be as $a_{t-1}, a_{t-2}, a_{t-3}$, *and* a_0. The terms $(f_iL_i \bmod p)$ are common in all the above equations and required to retrieve all the 4 coefficients. Further, if we observe closely, this term is also present in Shamir's scheme. These terms can be calculated once and stored.

Next, to retrieve a_0 in the above scheme or a secret $f(0)$ in Shamir's scheme, the term $\left(\prod_{j=1\ldots t,\ j\,!=i} x_j\right)$ requires the computations of order $O(t^2)$. Similarly, even a_{t-3} requires the computations of order $O(t^2)$. But, for a_{k-2} and a_{k-1}, the computations are negligible. In Shamir's scheme, four $O(t^2)$ computations would be required as against two $O(t^2)$ computations in the above scheme. Thus, in MANET where computing power, memory, and battery life of devices is limited, this scheme would reduce the power consumption to half for threshold implementation.

Lastly, since r, the random number in rG, can be reused as r or its partial form is never exposed during transmission by using this scheme.

The only constraint identified in this scheme is that t and n must be greater than or equal to 4 ($t,\ n \geq 4$) to securely transmit $a_{t-1}, a_{t-2}, a_{t-3}$ *and* a_0 at one time.

Considering the bandwidth restrictions in MANETs, an alternative to Shamir's secret sharing scheme using Lagrange interpolation is suggested to reduce the packet size for all the above ECC algorithms to constant $2w$ i.e. partial share C_i and its corresponding x_i value. Using this method up to 4 secrets can be transmitted with constant packet size of $2w$ without adding minimum complexity irrespective of which algorithm is used. We have also identified that this method is applicable only when n and $t \geq 4$. Thus, using our ECC-TC implementation the complexity for the algorithms reduces to as shown in Table 6.

Table 6: Complexity comparison of ECC-TC Encryption/Decryption algorithms

ECC TC Algorithm	Share split before encryption			Share split after encryption			Pkt size	Pkt #
	r G	P+ Q	Lag	*r G*	P+ Q	Lag		
DH	2	2n	1	0	2	1	2w	n
MV	3	0	1	3	0	1	2w	n
Ertaul	3	0	1	3	0	1	2w	n

Note: n, t should be ≥ 4 for to achieve 2w packet size
Lag= Lagrange Timings

5. Conclusion

From earlier RSA-TC implementation, we have put forth reasons that justify why it is unsuitable for implementation in MANET.

Through implementation of three most efficient ECC-TC based algorithms ECC-DH, MV, and Ertaul, we have proved that for higher *t* and *n*, share generation and point conversion adds largely to encryption and decryption timings. We have suggested different scenarios where each ECC-TC algorithm could provide security for MANETs by comparing different timings. We have proved earlier that ECC-DH split before encryption is ideal for implementing at sender with resource constraints as encryption timings are lowest. Further, MV split before encryption is ideal for scenarios where receiver has resource constraints as decryption timings as lowest. It is confirmed that the encryption and decryption timings differ significantly for ECC-DH and MV in both split before and after encryption scenarios. From the results, compared to ECC-DH and MV, Ertaul TC has moderate encryption and decryption timings that are very close and do not vary significantly with changes in key-size, *t,* and *n* values for both split before and after encryption. By comparing the implementation results for all techniques, we have concluded in dynamic environment such as MANET where *t* and *n* values would be adjusted frequently, ECC-DH and MV TC will not prove effective as timings will change significantly and may hinder performance as nodes that may have resource constraints have to act both as sender and receiver. Thus, Ertaul would be better for MANETs compared to other two algorithms.

In this paper, we have presented an efficient substitute for Shamir's secret sharing that provides for multiple secret sharing scenarios such as ECC-TC. For *n, t* >=4, this scheme allows sharing of up to 4 secrets but the packet size is constant, 2*w*. It does not depend on any variable i.e. *n, t,* or ECC-TC algorithm which means that the communication overheads remain constant for all algorithms. Thus, selection of an efficient ECC-TC algorithm depends on the operations involved in it.

Applications of MANETs are on rise and hence it is necessary to provide security to this highly vulnerable wireless network. And by further exploring and implementing ECC based threshold cryptography algorithms, secure MANETs are feasible.

References

[1] A. Mishra and K. M. Nadkarni, "Security in wireless ad hoc networks – A Survey", in The Handbook of Ad Hoc Wireless Networks, M. Ilyas, Ed. Boca Raton: CRC Press, 2002, pp. 30.1-30.51.

[2] P. Papadimitratos and Z. Hass, "Securing Mobile Ad Hoc Networks", in The Handbook of Ad Hoc Wireless Networks, M. Ilyas, Ed. Boca Raton: CRC Press, 2002, pp. 31.1-31.17.

[3] H. Yang, H. Luo, F. Ye, S. Lu, and U. Zhang, "Security in Mobile Ad Hoc Networks: Challenges and Solutions", IEEE Wireless Communications, vol. 11, no. 1, Feb. 2004, pp. 38-47.

[4] W. A. Arbaugh, "Wireless Security is Different", IEEE Computer, vol. 36, no. 8, Aug. 2003, pp. 99-101.

[5] Y. G. Desmedt, "Threshold cryptography", *European Trans. on Telecommunications, 5(4),* pp. 449-457, July-August 1994.

[6] P. S. Gemmell, "An Introduction to Threshold Cryptography", *Cryptobytes,* 1997, pp. 7-12.

[7] Y. Desmedt and Y. Frankel, "Threshold cryptosystems", in Advances in Cryptology - Crypto '89, Proceedings, Lecture Notes in Computer Science 435, G. Brassard, Ed., Santa Barbara: Springer-Verlag,1990, pp. 307-315.

[8] Y. Desmedt, "Some Recent Research Aspects of Threshold Cryptography", *Information Security, Proceedings (Lecture Notes in Computer Science 1396),* Springer-Verlag 1997, Tatsunokuchi, Ishikawa, Japan, September 1997, pp. 158-173.

[9] J. Baek and Y. Zheng, Simple and Efficient Threshold Cryptosystem from the Gap Diffie-Hellman Group. Available at http://citeseer.nj.nec.com

[10] K. Lauter, "The advantages of Elliptic Curve Cryptography For Wireless Security", IEEE Wireless Communications, vol. 11, no. 1, Feb. 2004, pp. 62-67.

[11] L. Ertaul and N. Chavan, "Security of Ad Hoc Networks and Threshold Cryptography", in MOBIWAC 2005.

[12] M. Narasimha, G. Tsudik, and J. Yi, On the Utility of Distributed Cryptography in P2P and MANETs: the Case of Membership Control. [Online]. Available: http://citeseer.ist.psu.edu/688081.html

[13] N. Koblitz, A Course in Number Theory and Cryptography (Graduate Texts in Mathematics, No 114), Springer-Verlag, 1994.

[14] L. Ertaul and W. Lu, "ECC Based Threshold Cryptography for Secure Data Forwarding and Secure Key Exchange in MANET (I)," Networking 2005, LCNS 3462, University of Waterloo, Canada, May 2005, pp. 102-113.

[15] T. El Gamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," IEEE Transactions on Information Theory, vol. 31(4), July 1985, pp. 469-472.

[16] N. Koblitz, "Elliptic Curve Cryptosystems," Mathematics of Computation, vol. 48(177), pp. 203-209, 1987.

[17] "Recommended Elliptic Curves for Federal Government Use." [Online]. Available: http://csrc.nist.gov/CryptoToolkit/dss/ecdsa/NISTReCur.pdf

[18] K.Eodh, "Elliptic Curve Cryptography: Java Implementation," Proceedings of the 1st Annual Conference on Information Security curriculum development, October 2004, pp. 88-93.

[19] L. Zhou, F. B. Schneider, and R. van Renesse, "*COCA: A Secure Distributed On-line Certification Authority*", ACM Transactions on Computer Systems, vol. 20, no. 4, November 2002, pp. 329-368.

[20] G. D. Crescenzo, R. Ge, and G. R. Arce, "Improved Topology Assumptions For Threshold Cryptography In Mobile Ad Hoc Networks," Proceedings of the 3rd ACM workshop on Security of Ad Hoc And Sensor Networks, ACM Press, 2005, pp. 53-62.
[21] W. Trappe, L. C. Washington, *Introduction to Cryptography: with Coding Theory*, Prentice Hall, 2002.

Levent Ertaul received the B.Sc., M.Sc. and Ph.D degrees from Hacettepe University, Turkey, in 1984, 1987, and from Sussex University, UK, in 1994, respectively. After working as an assistant professor (from 1994) in the Dept. of Electrical & Electronics Engineering, Hacettepe University, he moved to California State University, East Bay in 2002. He is currently a full time Asst. professor at California State University Eastbay, in the department of Math & Computer Science. He is actively involved in security projects nationally and internationally. His current research interests are Mobile Agents Security, Wireless Security, Ad Hoc Security and Cryptography. He has numerous publications in Security issues.

Nitu Chavan received the B.Sc. in Electronics and Telecommunication Engineering and the M.B.A. in Computer Management from Pune University in 1997 and 1999, respectively. From 1999 till 2001, she worked on various software applications including applications for PDAs. Currently, she is working at IBM Inc. and pursuing M.Sc. in Computer Science at California State University, East Bay.