RSA and Elliptic Curve- ElGamal Threshold Cryptography (ECCEG-TC) Implementations for Secure Data Forwarding in MANETs

Levent Ertaul
California State University, East Bay
Math & Computer Science Department
Hayward, CA, USA

Nitu J. Chavan,
California State University, East Bay
Math & Computer Science Department
Hayward, CA, USA

Abstract— *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 these networks is challenging. Threshold Cryptography (TC) provides a promise of securing these networks. In this paper, we discuss our RSA-based Threshold Cryptography (RSA-TC) and ECC El Gamal Threshold Cryptography (ECCEG-TC) implementation. Through our implementation, we have put explored possibility of using RSA-TC and ECCEG-TC in MANETs. Finally, we compare RSA-TC and ECCEG-TC results and suggest why RSA-TC is unsuitable while ECCEG-TC is suitable for MANETs.*

Keywords—Threshold Cryptography, ECCEG-TC, RSA-TC

I. Introduction

Mobile ad hoc network (MANETs) are vulnerable to various attacks including denial-of-service attacks because of wireless nature of these networks [1], [2], [3], [4]. Devices with constraint resources add to its vulnerability. To ensure availability of nodes, threshold cryptography can be implemented in these networks 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 the threshold [5], [6], [7], [8], [9].

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 MANETs.

In this paper, we discuss our implementation on RSA based TC and Elliptic curve cryptography TC using ElGamal algorithms. We make a case why RSA-TC is unsuitable for these networks and using ECCEG-TC we put forth our idea on why ECC-based algorithms for TC will be more appropriate for these networks.

II. RSA vs Elliptic Curve Cryptology

RSA has been successfully implemented in computer networks for threshold authentication where nodes have large computational and storage capacity. From Table I, it is clear that ECC provides equivalent security as RSA

TABLE I
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 II
SAMPLE ECC EXPONENTIATION OVER GF(P) AND RSA ENCRYPT/DECRYPT TIMINGS IN MILLISECONDS

	MHz	163 ECC	192 ECC	1024 RSAe	1024 RSAd	2048 RSAe	2048 RSAd
Ultra SparcII 400MHz	450	6.1	8.7	1.7	32.1	6.1	205.5
Strong ARM 200MHz	200	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.

but at much smaller key sizes [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 II.

III. RSA-TC

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

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,

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

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

Fig. 1. RSA and RSA-TC scheme using Shamir's Lagrange Interpolation.

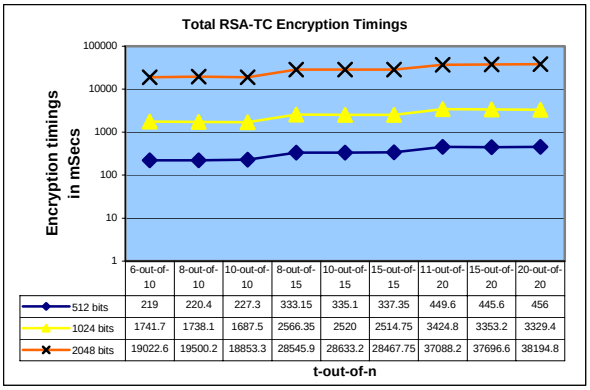
A. RSA-TC Implementation

RSA-TC has been implemented using *JAVA 1.4* in Unix environment on *SUN Sparc Ultra 5_10* machines. Fig. 1 explains the RSA-TC scheme [11].

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.

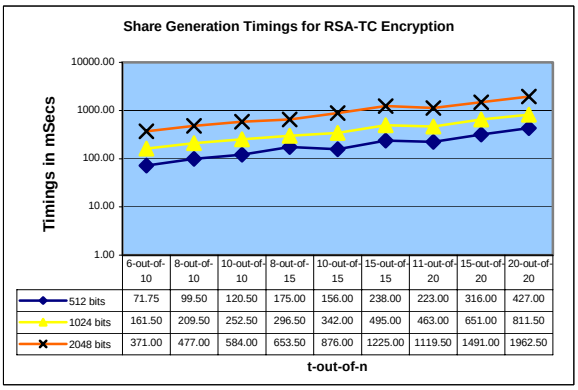
For RSA-TC, private key d is split using Shamir's *t-out-of-n* scheme based on Lagrange interpolation [7] to generate partial keys such that any t out of n partial messages will allow retrieval of the original message. These keys are used to carry out partial encryption.

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



	6-out-of-10	8-out-of-10	10-out-of-10	8-out-of-15	10-out-of-15	15-out-of-15	11-out-of-20	15-out-of-20	20-out-of-20
512 bits	219	220.4	227.3	333.15	335.1	337.35	449.6	445.6	456
1024 bits	1741.7	1738.1	1687.5	2566.35	2520	2514.75	3424.8	3353.2	3329.4
2048 bits	19022.6	19500.2	18853.3	28545.9	28633.2	28467.75	37088.2	37696.6	38194.8

Processor: SUN Sparc Ultra 5_10 Timings for 500 runs
Fig. 2. Total Encryption Timings for RSA-TC

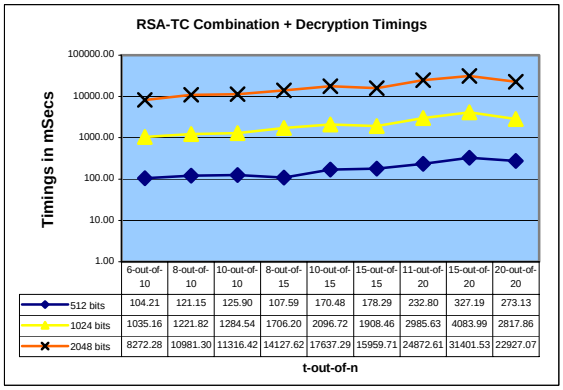


	6-out-of-10	8-out-of-10	10-out-of-10	8-out-of-15	10-out-of-15	15-out-of-15	11-out-of-20	15-out-of-20	20-out-of-20
512 bits	71.75	99.50	120.50	175.00	156.00	238.00	223.00	316.00	427.00
1024 bits	161.50	209.50	252.50	296.50	342.00	495.00	463.00	651.00	811.50
2048 bits	371.00	477.00	584.00	653.50	876.00	1225.00	1119.50	1491.00	1962.50

Processor: SUN Sparc Ultra 5_10 Timings for 500 runs
Fig. 3. Share Generation Timings for RSA-TC

B. Performance Results

Fig. 2 shows that 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.



	6-out-of-10	8-out-of-10	10-out-of-10	8-out-of-15	10-out-of-15	15-out-of-15	11-out-of-20	15-out-of-20	20-out-of-20
512 bits	104.21	121.15	125.90	107.59	170.48	178.29	232.80	327.19	273.13
1024 bits	1035.16	1221.82	1284.54	1706.20	2096.72	1908.46	2985.63	4083.99	2817.86
2048 bits	8272.28	10981.30	11316.42	14127.62	17637.29	15959.71	24872.61	31401.53	22927.07

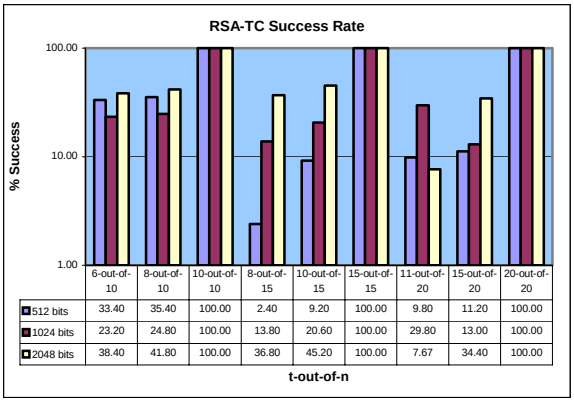
Processor: SUN Sparc Ultra 5_10 Timings for 500 runs
Fig. 4. Combination + Decryption Timings for RSA-TC

Fig. 3 displays that share generation time increased exponentially as the key-size was doubled. This 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.

Fig. 4 depicts behavior similar to encryption timings graph. Combination time is the time required to combine partially encrypted message to retrieve original ciphertext. For a given key-size, combination time and decryption time gradually increased with n

and *t*. Further, increasing the key-size results into exponential increase in these timings for a given *n* and *t*.

Fig. 5 shows that the success rate increases as *t*



	6-out-of-10	8-out-of-10	10-out-of-10	8-out-of-15	10-out-of-15	15-out-of-15	11-out-of-20	15-out-of-20	20-out-of-20
512 bits	33.40	35.40	100.00	2.40	9.20	100.00	9.80	11.20	100.00
1024 bits	23.20	24.80	100.00	13.80	20.60	100.00	29.80	13.00	100.00
2048 bits	38.40	41.80	100.00	36.80	45.20	100.00	7.67	34.40	100.00

Processor: SUN Sparc Ultra 5_10 Timings for 500 runs
Fig.5. %Success Rate for RSA-TC

increases from *n/2* to *n*. For *t=n*, success rate is 100%[11]. Success rate varies as $\phi(N)$ is a even number and all the 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 ciphertext [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 it is negligible in comparison with encryption time. Further, 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. We will get similar results as above but with 100% success rate when we implement message split before encryption because partial encryption requires *n* encryptions and Lagrange once. Similarly, RSA-TC with message split before encryption would generate *n* partial messages using Lagrange interpolation once and then these partial messages are encrypted using *n* encryptions.

In the next section, we would discuss our ECC implementation based on ElGamal algorithm.

IV. ECC ELGAMAL TC (ECCEG-TC) IMPLEMENTATION

In following sections, our goal is to implement ECC based ElGamal threshold cryptography (ECCEG-TC). In this algorithm, key is not shared 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, ECCEG-TC for message splitting before encryption is simulated for MANET environment and then it is compared with performance of RSA-TC. The ECC El Gamal Threshold cryptography (ECCEG-TC) algorithm is briefly explained.

A. ECCEG-TC Message Split before Encryption Algorithm

Suppose that the ECC has a point G on an elliptic curve $E_p(a, b)$, and the order of *G* is *q*. *p* is a large prime.

Bob's private key and public key are n_B, $0 < n_B < q$, and $K_B = n_B G$.

- First we choose a prime number $p > max(M, n)$, and define $a_0 = M$, the message. Then we select $k - 1$ random, independent coefficients $a_1, a_2, \ldots a_{k-1}$, $0 \leq a_j \leq p-1$, defining the random polynomial $f(x)$ over *Zp*, a Galois prime field *GF(p)*.
- We compute *n* shares, $M_i = f(x_i) \bmod p$, $1 \leq i \leq n$, *where* x_i can be just the public index *i* for simplicity, and convert them to points P_i on elliptic curve $E_p(a, b)$.
- Alice picks a random number *r*, and sends *rG* and $P_i + rK_B$ to Bob with index *t*.
- Bob recovers each elliptic curve point by calculating $P_i + rK_B - n_B rG = P_i$.
- Bob converts P_i to M_i, and deduces *M* by using Lagrange interpolation formula M.

B. ECCEG-TC Implementation

ECCEG-TC has been implemented using *JAVA 1.4* in Unix environment on *SUN Sparc Ultra 5_10* machines. To select the ECC parameters, i.e. *a*, *b*, *p,* widely accepted NIST curves were selected for implementation for *192*, *224*, and *256* bits.

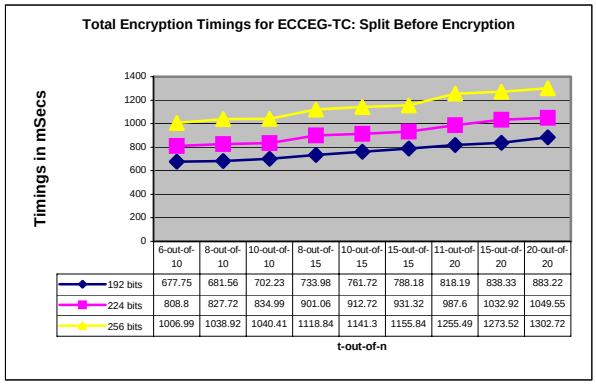
For conversion of message to and from ECC point, method discussed by Kobiltz is used [13], [14] such that $(kappa*M) \bmod p < x < (kappa*(M+1)) \bmod p$, *where (x, y)* is a point on elliptic curve. In our ECCEG-TC implementation, *kappa* is fixed to 2^8. To retrieve a message from a ECC point *(x, y)*, $M = x/kappa \bmod p$ is used.

For calculating the shares and for combining partial messages, Shamir's Lagrange interpolation scheme is implemented. For its polynomial, 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. As against RSA algorithm where we are sharing the keys, in ECC-TC implementation, the partial shares of the message are generated and then

encrypted to get ECC point.

C. Performance Results

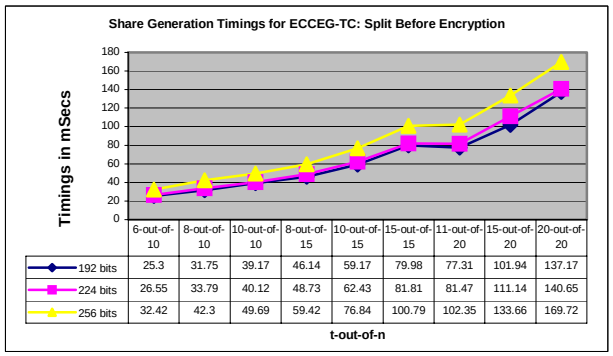
Fig. 6 illustrates that with increase in ECC key size, the total encryption timings increase gradually for given n and t. For constant key size and n, the encryption timings increase with t as the time to generated Lagrange polynomial and respective message shares increases accordingly.



	6-out-of-10	8-out-of-10	10-out-of-10	8-out-of-15	10-out-of-15	15-out-of-15	11-out-of-20	15-out-of-20	20-out-of-20
192 bits	677.75	681.56	702.23	733.98	761.72	788.18	818.19	838.33	883.22
224 bits	808.8	827.72	834.99	901.06	912.72	931.32	987.6	1032.92	1049.55
256 bits	1006.99	1038.92	1040.41	1118.84	1141.3	1155.84	1255.49	1273.52	1302.72

Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
Fig. 6. Total Encryption Timings for ECCEG-TC

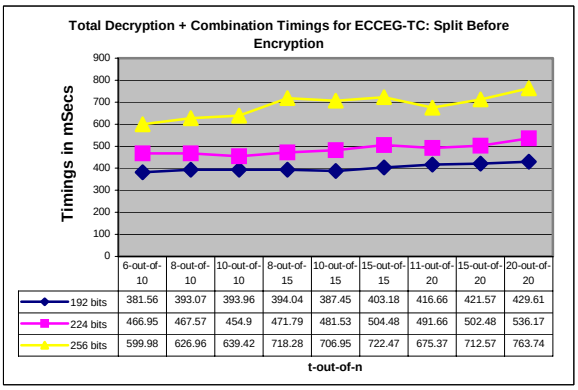
Fig. 7 shows that the share generation timings increase with increase in key size or with n or t. Share generation timings are very small compared to the encryption timings.



	6-out-of-10	8-out-of-10	10-out-of-10	8-out-of-15	10-out-of-15	15-out-of-15	11-out-of-20	15-out-of-20	20-out-of-20
192 bits	25.3	31.75	39.17	46.14	59.17	79.98	77.31	101.94	137.17
224 bits	26.55	33.79	40.12	48.73	62.43	81.81	81.47	111.14	140.65
256 bits	32.42	42.3	49.69	59.42	76.84	100.79	102.35	133.66	169.72

Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
Fig. 7. Share Generation Timings for ECCEG-TC

Combination time is the time required to combine t partial messages using Shamir's Lagrange interpolation method to retrieve original message. From Fig. 8, the total decryption and combination timings increase gradually with increase in t for constant key size and n. This increase is due to time required to decrypt and combine additional partial messages as t is increased. Increase in the key size results in proportional increase in the decryption timings irrespective of n and t.



	6-out-of-10	8-out-of-10	10-out-of-10	8-out-of-15	10-out-of-15	15-out-of-15	11-out-of-20	15-out-of-20	20-out-of-20
192 bits	381.56	393.07	393.96	394.04	387.45	403.18	416.66	421.57	429.61
224 bits	466.95	467.57	454.9	471.79	481.53	504.48	491.66	502.48	536.17
256 bits	599.98	626.96	639.42	718.28	706.95	722.47	675.37	712.57	763.74

Processor: SUN Sparc Ultra 5_10 Timings for 200 runs
Fig. 8. Decryption and Combination Timings for ECCEG-TC

Number of point addition of ECCEG-TC increases with n resulting into proportionate increase in addition timing in encryption and decryption as seen in Fig. 6 and Fig. 8.

The time required converting message to point and vice-versa is significantly small compared to encryption and share generation time and hence not shown separately.

In ECCEG-TC, the Lagrange is carried over prime field p, hence the success rate is 100% as all the partial messages are recovered without any issue of inverse calculation.

D. Comparison between RSA-TC and ECCEG-TC

By comparing data in Fig. 2 and Fig. 6 and Fig. 4 and Fig. 8, it is clear that RSA-TC is much expensive in terms of encryption and decryption timings irrespective of n and t values as compared to ECCEG-TC.

With increase in key-size for ECCEG-TC the security provided increases significantly as in case of ECC but the total timings required by this algorithm still require $O(n)$ computations and are in milliseconds. The increase in the timings is gradual as the key size and n are increased. As against this, the timings in RSA-TC increase exponentially with increase in key-size.

As ECC is known to provide equivalent security as RSA at much smaller key sizes, here ECCEG-TC would also provide equivalent security as RSA-TC. It is also evident that ECCEG-TC is much efficient algorithm compared to RSA-TC. Due to smaller key size the storage requirements during the encryption are very less for ECCEG-TC compared to RSA-TC.

Compared to RSA-TC, due to smaller key size in ECCEG-TC would result in less bandwidth consumption during transmission.

V. CONCLUSIONS

Through implementation of RSA-TC and ECCEG-TC, we have suggested an approach to provide security for MANETs. In section 6, by comparing the implementation results for both techniques, we have proved through that ECC-TC implementation of EG algorithm would be better for MANETs compared to RSA-TC implementation.

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