DISTRIBUTED PAIRWISE KEY GENERATION USING SHARED POLYNOMIALS FOR WIRELESS AD HOC NETWORKS

Anindo Mukherjee, Hongmei Deng and Dharma P. Agrawal
Center of Distributed and Mobile Computing
University Of Cincinnati, Cincinnati, OH 45229
{mukherao, hdeng, dpa}@ececs.uc.edu

Abstract: The infrastructure-less property of wireless ad hoc network makes the traditional central server based security management schemes unsuitable and requires the use of a distributed key management mechanism. In this paper, we propose a distributed pairwise key establishment scheme based on the concept of bivariate polynomials. In our method, any mobile node in an ad hoc network can securely communicate with other nodes just by knowing their corresponding IDs. The bivariate polynomials are shared in such a manner that the shares depend on the coefficient matrix of the polynomial, the requesting node's ID and the ID of the nodes that respond to the request. We study the behavior of our scheme through simulations and show that our scheme compares well with other schemes and has a much better performance when averaged over the lifetime of the network.

Key words: Security; Ad Hoc Networks; Symmetric Keys; Bivariate Polynomials; Threshold Secret Sharing

1. INTRODUCTION

Security in ad hoc networks is riddled with a constant change in paradigms. Murphy et al. [6] defined ad hoc networks as “A transitory association of mobile nodes which do not depend on any fixed support infrastructure.” Thus, an ad hoc network can be either a network of radios in a battlefield or a network of laptops in an office environment etc. This multitude of applications makes the deployment of a common security infrastructure a complex problem. In addition to the diversity in the kinds of applications, security in ad hoc networks is severely constrained due to the dynamic nature of the networks. Participants may join and leave the network at any time. The traditional central server based security management mechanism may not be directly applicable since the incoming participants need not have access to a central trusted server after the network has been deployed. Thus, a distributed key management mechanism is necessary in securing wireless ad hoc networks. Several approaches towards implementing a distributed key management scheme have been proposed in literature. Zhou and Hass [9] use a partially distributed certificate authority system, in which a group of special nodes is capable of generating partial certificates using their shares of the certificate signing key. In [5], Kong et al. proposed another threshold cryptography scheme by distributing the RSA certificate signing key to all the nodes in the network.

Both the approaches are based on *asymmetric* cryptosystem, which imposes a high processing overhead. In this paper, we consider a symmetric key based approach and focus on distributed pair-wise key generation. In a pairwise key scheme, each node pair shares a unique symmetric key. There are a number of applications for these types of keys. SRP for DSR [7] uses pair-wise keys for authentication. Also, any secure communication between two nodes in the absence of a public key system would require pair-wise symmetric keys between nodes.

Before discussing any further, we would like to state our assumptions and the problem. We base our system on the following assumptions:

- A trusted server is present which initializes a set of nodes before deployment. This server is not present after the nodes have been deployed. Any un-initialized node would need to get its keying material from the network.
- An incoming node has the computational power to generate a temporary public key-private key pair.
- A node that joins the network and tries to obtain keying material from its neighborhood has a mechanism to prove its authenticity to the nodes it requests the shares from.

With the above assumptions in place, we state the problem as: Given an operational ad hoc network with a set of nodes initialized (by a central authority) with the keying material, a node that wishes to join the network needs to securely obtain its own keying material without the help of the central authority.

Several solutions to the problem have been proposed in literature. In [4], the authors present a probabilistic key pre-distribution technique. This idea has been extended in [2] where the authors propose a q-composite key pre-distribution.

We propose a distributed mechanism to share keying material between n nodes such that any t nodes can get together and provide another incoming node with its keying material. At the same time, an adversary listening to all the ongoing conversation and having compromised less than t members would not be able to obtain any pairwise key.

Our scheme is based on the concept of bivariate polynomials, first outlined in [1]. We extend this scheme to a distributed scenario by modifying Shamir's threshold scheme [8], so that incoming nodes can be initialized by the network into getting the keying material from the network. To the best of our knowledge, no attempt has been made to share these polynomials in a secure and distributed manner. The proposed scheme has a number of attractive properties. First, in our scheme, any incoming node would be able to get its key shares from the network and need not rely on a central server. Second, the scheme is resilient to the compromise of *t-1* nodes. Third, a node which joins the network needs to communicate only with its immediate neighborhood in order to get all its keying material. Fourth, physical capture of a node would give away only the captured node's keying material without compromising the network. Finally, our scheme is simple and does not require complex protocols to be implemented.

The rest of the paper is organized as follows. In Section 2 we introduce some background knowledge for our scheme. In Section 3 we present the proposed distributed pair-wise key generation scheme in detail. We give simulation results and discuss possible extensions to the work in Section 4. Finally, Section 5 concludes.

2. BACKGROUND

In this section, we first take a brief look at the bivariate polynomial scheme introduced in [1], and also at the concept of threshold secret sharing introduced in [8].

2.1 Bivariate polynomial-based key pre-distribution

Consider a bivariate polynomials *f(x,y)* of degree *t,* defined as

$$f(x,y) = \sum_{i,j=0}^{t-1} a_{ij} x^i y^j \qquad (1)$$

where the coefficients a_{ij} are randomly chosen over a finite field GF(q). The bivariate polynomial has a symmetric property such that

$$f(x,y) = f(y,x) \qquad (2)$$

An initial server first proceeds to initialize a set of nodes by giving each node *m* the polynomial $g_m(y) = f(m,y)$, which is the polynomial obtained by evaluating $f(x,y)$ at $x = m$. That is, a deployed node would know

$$g_j = \sum_{i=0}^{t-1} a_{ij}.(m)^i \qquad (0 \le j \le t-1) \qquad (3)$$

where *m* is identity of the node being deployed, and g_j is coefficient of y^j in the polynomial $f(m,y)$.

Thus, in order for a node with ID *m* to calculate the pairwise key with a node with ID *n*, node *m* simply finds out the value of *f(m,y)* at *y=n.* Similarly, node *n* in turn evaluates the polynomial at *y=m.* Due to the symmetric property of the bivariate polynomial *f(x,y)*, they manage to establish a pairwise secret key known only to them.

The above procedure can be represented using a matrix notation as

$$K_{mn} = K_{nm} = \mathbf{X}^T \mathbf{A} \mathbf{Y} = \mathbf{Y}^T \mathbf{A} \mathbf{X} \tag{4}$$

where,

$$\mathbf{X} = [m^0, m^1, m^2, m^3, \cdots m^{t-1}]^T$$

A = Coefficient Matrix for *f(x,y)*

$$\mathbf{Y} = [n^0, n^1, n^2, n^3, \cdots n^{t-1}]^T$$

$\boldsymbol{K}_{mn} = \boldsymbol{K}_{nm}$ = Shared Key between node *m* and *n*

The deployed node *m* obtains the information **G= X^{T}A**=[g_0, g_1, …g_{t-1}]. Note that the elements of matrix **A** are not known to anyone except the initial server.

2.2 Threshold secret sharing

Secret sharing allows a secret to be shared among a group of users (also called shareholders) in such a way that no single user can deduce the secret from his share alone. One classical (*t*, *n*) secret sharing algorithm was proposed by Adi Shamir [8] in 1979, which is based on polynomial interpolation. In the scheme, the secret is distributed to *n* shareholders, and any *t* out of the *n* shareholders can reconstruct the secret, but any collection of less than *t* partial shares can not get any information about the secret. We use the scheme to share polynomials in such a manner that the coefficients of the polynomial would always remain secret. Any combination of *t* nodes would only derive the value of the polynomial at a certain point.

3. PROPOSED DISTRIBUTED KEY GENERATION SCHEME

In our scheme, we distribute the shares of the matrix A among *n* initial nodes such that:

- Any combination of *t* nodes would be able to derive the keying material for an incoming node. (Note that this does not amount to the nodes getting to know the coefficients of the matrix A. Instead, the incoming node would only be able to derive **X^{T}A**, as indicated in (4).
- Any combination of less than *t* nodes would not be able to derive any portion of the keying material for an incoming node.
- The central server initializes only a set of *n* nodes. A node which has not been initialized by the central server and which wishes to join the

network can do so without having to contact the central server provided at least t nodes send in keying material to this node.

To achieve the above, we modify Shamir's t-threshold scheme as outlined below. Let the newly arrived requesting node have an id of α, and the responding nodes have their ids as β_1, β_2, and so on.

We now present the scheme mathematically. In our discussion, we would also use an example to illustrate the steps. In the considered example, the node that wishes to join the network has an ID of 2, the responding nodes have IDs 1, 3 and 5 and the threshold value is 3 (i.e., t=3).

We first take a look at the shares of the key generation material that would be given to each of the initial set of nodes before deployment. The shares should have the property that no set of nodes less than or equal to t should be able to generate either the coefficient matrix A, or the vector $\mathbf{X^TA}$ for any other node.

Each node that is deployed would be initialized with a matrix A_i where A_i is of the following form:

$$\begin{bmatrix} s_{00} & s_{01} & .. & s_{0(t-1)} \\ s_{10} & s_{11} & .. & s_{1(t-1)} \\ .. & .. & .. & .. \\ s_{(t-1)0} & s_{(t-1)1} & & s_{(t-1)(t-1)} \end{bmatrix}$$

Each element s_{ij} is given by:

$$s_{ij} = a_{ij} + \sum_{m=1}^{t_1} b_{ijm}(\beta^m) \tag{5}$$

Here b_{ijm} are random numbers generated by the central server. These numbers are not known to anyone except the central server (Note that this central server is only present before deployment and would have no role to play after the network is in operation). t_1 is the threshold for sharing the matrix **A**. β is the ID of the node.

In our example, the above quantities for node 3 are given by:

$$s_{00} = a_{00} + 3.b_{001} + b_{002}.3^2$$
$$s_{01} = a_{01} + 3.b_{011} + b_{012}.3^2$$
$$.....$$
$$s_{21} = a_{21} + 3.b_{211} + b_{212}.3^2$$
$$s_{22} = a_{22} + 3.b_{221} + b_{222}.3^2$$

Also, note that t_1 and t are two separate quantities. While t_1 denotes the threshold for sharing each element of the matrix **A**, t denotes the number of terms in the vector $\mathbf{X^TA}$. For our purposes, we take t_1=t, because both t and t_1 essentially represent the maximum number of members that can be compromised in a network. Although t and t_1 represent two different thresholds, we require them to be the same to have a consistent threshold value for the network.

As soon as a new node with ID α (node '2' in our example) joins the network, it sends out a temporary public key, P_k to its immediate neighborhood. The format of the message sent is

{REQ_SHARE, α, P_k, TTL }

Here the field TTL specifies the number of hops that the message would be broadcasted to. We start with TTL = 1. If the required number of replies is obtained within a time given by T_{rep}, the process is stopped. Otherwise a new request is sent out with TTL = 2 and so on.

Any node (1, 3 and 5 in our example) in the immediate neighborhood that receives a REQ_SHARE message responds in the following manner: The node first computes

$$S_{ij} = s_{ij}(\alpha)^i \qquad (0 \le i, j \le t-1) \qquad (6)$$

The node now computes

$$H_{\beta j} = \sum_{i=0}^{t-1} S_{ij} \qquad (0 \le j \le t) \qquad (7)$$

and sends $E_{Pk}(H_{\beta 0}, H_{\beta 1}, H_{\beta 2}, \ldots, H_{\beta(t-1)}, \beta)$ to the node α. Here E_{Pk} implies encryption using the public key P_k. For example, the quantity H_{31} sent by node 3 to node 2 would be:

$(a_{00}+ 3\cdot b_{001}+3^2\cdot b_{002})\, 2^0+(a_{10}+3\cdot b_{101}+3^2\cdot b_{102})\, 2^1+(a_{20}+3\cdot b_{201}+3^2\cdot b_{202})\cdot 2^2$

Let V_j denote the quantity

$$V_j = \sum_{w=0}^{t-1} a_{wj}\alpha^w \qquad (8)$$

The row vector $\mathbf{V}$ = [V_0, V_1, ……., V_{t-1}] thus denotes the quantity $\mathbf{X}^{\mathbf{T}}\mathbf{A}$ by substituting $m=\alpha$ in Eq. (4). This is all that the node α would require in order to find out any pairwise key with any other node.

In our example, the values V_i for node 2 is given by:

$$V_0 = a_{00}2^0 + a_{10}2^1 + a_{20}2^2$$
$$V_1 = a_{01}2^0 + a_{11}2^1 + a_{21}2^2$$
$$V_2 = a_{02}2^0 + a_{12}2^1 + a_{22}2^2$$

and the vector V is $[V_0, V_1, V_2]$.

We now show how the node α computes V after it has obtained the shares from at least t nodes. Let $V_{i\beta}$ be the share of V_i obtained from node β. Thus, rewriting Eq. (7),

$$H_{\beta j} = \sum_{w=0}^{t} a_{wj}\alpha^w + \sum_{w=0}^{t-1}\left(\sum_{m=1}^{t-1} b_{wjm}\beta^m\right)\alpha^w \qquad (9)$$

as can be derived from Eqs. (5)-(7).

Equation (9) has two terms. The first term is the quantity V_j. The second term constitutes a set of t-1 terms . Thus, t such equations would enable node α to calculate the value of each term. Node α would keep only the first term as the value of V_i and discard all other values.

Using the vector $\mathbf{V} = [V_0, V_1, ...V_{t-1}]$, node α can now find out its pairwise key with any other node. (Note that solving for the vector **V** is only as complicated as finding t secrets in Shamir's threshold scheme). In the considered example, the above shares for V_0, V_1 and V_2 obtained by node 2 are shown in Table 1.

	Node 1	Node 3	Node 5
Shares for V_0	$(a_{00}+2\cdot a_{10}+2^2\cdot a_{20})+$ $1\cdot(b_{001}+2\cdot b_{101}+2^2\cdot b_{201})+$ $1^2\cdot(b_{002}+2\cdot b_{102}+2^2\cdot b_{202})$	$(a_{00}+2\cdot a_{10}+2^2\cdot a_{20})+$ $3\cdot(b_{001}+2\cdot b_{101}+2^2\cdot b_{201})+$ $3^2\cdot(b_{002}+2\cdot b_{102}+2^2\cdot b_{202})$	$(a_{00}+2\cdot a_{10}+2^2\cdot a_{20})+$ $5\cdot(b_{001}+2\cdot b_{101}+2^2\cdot b_{201})+$ $5^2\cdot(b_{002}+2\cdot b_{102}+2^2\cdot b_{202})$
Shares for V_1	$(a_{01}+2.a_{11}+2^2.a_{21})+$ $1\cdot(b_{011}+2\cdot b_{111}+2^2\cdot b_{211})+$ $1^2\cdot(b_{012}+2.b_{112}+2^2.b_{212})$	$(a_{01}+2.a_{11}+2^2.a_{21})+$ $3.(b_{011}+2\cdot b_{111}+2^2\cdot b_{211})+$ $3^2.(b_{012}+2\cdot b_{112}+2^2\cdot b_{212})$	$(a_{01}+2\cdot a_{11}+2^2\cdot a_{21})+$ $5\cdot(b_{011}+2\cdot b_{111}+2^2\cdot b_{211})+$ $5^2\cdot(b_{012}+2\cdot b_{112}+2^2\cdot b_{212})$
Shares for V_2	$(a_{01}+2.a_{11}+2^2.a_{21})+$ $1\cdot(b_{011}+2\cdot b_{111}+2^2\cdot b_{211})+$ $1^2\cdot(b_{012}+2\cdot b_{112}+2^2\cdot b_{212})$	$(a_{01}+2\cdot a_{11}+2^2\cdot a_{21})+$ $3\cdot(b_{011}+2\cdot b_{111}+2^2\cdot b_{211})+$ $3^2\cdot(b_{012}+2\cdot b_{112}+2^2\cdot b_{212})$	$(a_{01}+2.a_{11}+2^2.a_{21})+$ $5\cdot(b_{011}+2\cdot b_{111}+2^2\cdot b_{211})+$ $5^2\cdot(b_{012}+2\cdot b_{112}+2^2\cdot b_{212})$

Table 1. Example scenario with threshold t=3; the requesting node with ID 2 and the responding nodes with IDs 1, 3 and 5. The values in the table indicate the shares for V_0, V_1 and V_2 as sent by nodes 1, 3 and 5 to node 2. Using these values node 2 would be able to compute the values for V_0, V_1 and V_2.

In this manner, neither an eavesdropping node, nor the incoming node would have any knowledge about the matrix **A** since the coefficients of **A** are never revealed. Also, capture of a node simply compromises that node's pairwise keys and provides no information about the pairwise keys between any other two nodes. We thus have a fully distributed key generation scheme resilient to t-1 nodes getting compromised.

4. PERFORMANCE EVALUATION

In this section we compare the performance of our scheme with that of other key distribution schemes which employ similar messaging systems. Based on our observations, we also suggest a variation to the key exchange mechanism to optimize on the message overhead and latency. We first look at the number and length of messages that need to be sent to a node when it joins the network.

Considering a finite field of length q, length of each share is $log_2(q)$. Number of shares sent by a node is t. Thus, the length of the message sent by a responding node is $t*log_2q$. The number of such messages is at least t. We thus have a total of at least $t^2\, log_2q$ bits reaching the requesting node.

We simulated the working of our scheme to measure the performance for various network densities and threshold values. The simulations were carried out in ns-2 with the number of nodes kept at 40 over an area of

1000x1000m. The communication range was 250m. The threshold values were varied from 3 to 11.

We compared our scheme with the Threshold RSA (TRSA) scheme suggested in [5] and the ID-based scheme suggested in [3] to compare the performance in terms of the delay and message overhead. The basic idea behind the TRSA scheme is to generate authorized certificates in a distributed manner. A node joining the network requests for its share of the certificates from its neighbors and builds a certificate for its public key. The node now sends its public key along with the certificate to all the nodes in the network. In the ID based scheme, the node uses its ID as its public key and obtains shares for its private key from the network. For any communication, a session key is now established between a pair of nodes.

Both the above schemes are similar to our scheme as far as the message overheads are concerned. In all the three schemes, a node joining the network requests for its keying material from its neighbors (immediate or multihop). The behavior of our scheme is different from that of the ID-based scheme or TRSA since in our scheme, the size of the packets depends on the value of the threshold. In the other two schemes, the message size is independent of the threshold values. Thus, for small values of *t*, our scheme performs better and for large values, the performance of TRSA and ID-based schemes are better.

Also, our scheme introduces minimal post-key-exchange overhead on the network since no key announcements or session key establishment is needed prior to communication. This means that the performance of our system would be much better when averaged over the lifetime of the network.

We would like to note that although both the TRSA and the ID-based schemes deal with public key cryptosystems, we use them as metrics for comparison as these schemes give a good measure of the message overhead involved in obtaining keying material from the network. Also, for any communication to take place, a session key has to be established.

Our simulation goals were as follows:

- To observe the effects of the threshold values on the latency in obtaining the key shares
- To observe the effects of the threshold values on the network overhead.
- To see how our scheme performs under mobile conditions

Compare our scheme with TRSA and the ID-based scheme while considering the above parameters. We first looked at the latencies involved in getting all the required shares for the keys. The speed of the nodes is kept fixed at 30 m/s and the value of T_{rep} for our scheme was kept fixed at 1s. Figure 1 shows the obtained results. We call our scheme DBK for Distributed Bivariate Keying.

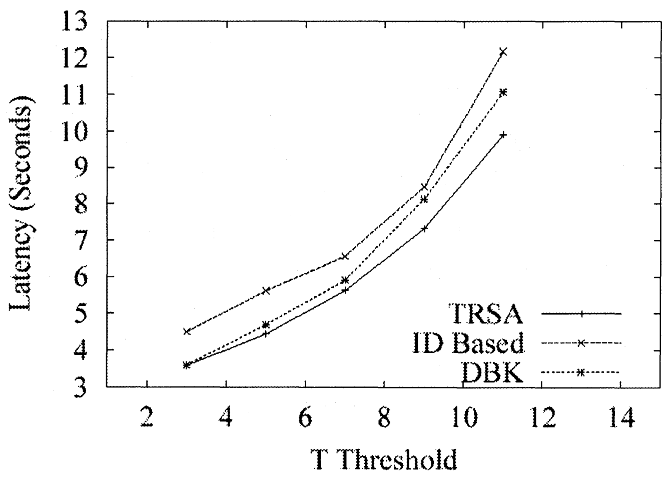

Figure 1: Latencies involved in obtaining the entire keying material from the network.

As can be seen, the latencies increase with an increase in the threshold values. An interesting point to be noted here is that as the threshold increases, the latency values for DBK approaches that of the TRSA scheme. This can be attributed to the fact that in DBK, the number of bytes being transmitted increases as the threshold values increase.

We next varied the threshold values and tried to obtain an estimate of the message overhead involved in obtaining the keying material from the network. Figure 2 plots the number of replies obtained by the requesting node. Again here the performance of DBK is midway between the performance of TRSA and the ID–based scheme. As the packet sizes increase, the latencies involved in the network also go up. Thus, with everything else remaining the same, with an increase in the packet size, packets take longer to reach their destination. If this delay becomes more than T_{rep}, a retransmission of the request packet happens and as a result, eventually more replies are obtained.

Figure 2 also shows the plots for DBK with $T_{rep} = 2$ and with $T_{rep} = 4$. The number of replies obtained for $T_{rep} = 4$ are much less than those obtained for $T_{rep} = 2$. This is evident from the fact that a larger wait would allow the node to get more replies before deciding to send out more requests at a higher latency cost.

We then see the effects of mobility on the latency. As can be seen in Figure 3, latency values drastically go down when the speeds are low. With an increase in speed, the latencies tend to become constant and stabilize to a value between 7 and 8. The reason for this is that with low mobility, nodes are able to obtain results faster from the two hop neighbors and do not send out additional requests for key shares. However, since the mode of sending requests is an *n*-hop broadcast (with *n* increasing from 1 onwards), the requesting node manages to obtain the replies faster by physically moving to new locations and new neighbors. As can be seen in Figure 3, the obtained values for DBK are lesser than TRSA and more than the ID-based scheme.

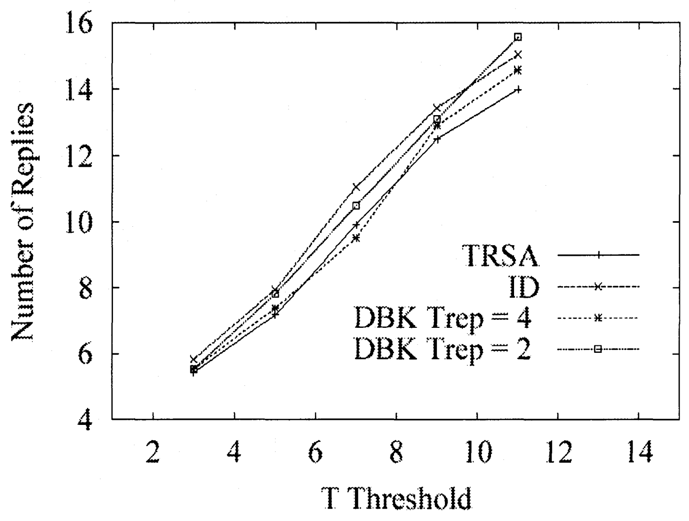

Figure 2: Total number of replies obtained by a node after sending out the request.

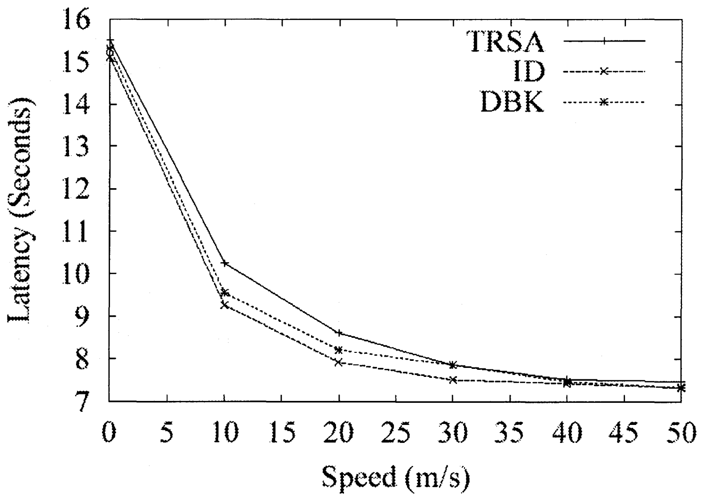

Figure 3: Variation in the latencies with the average speed of the mobile nodes.

As shown by the simulations, lower the value of T_{rep}, higher are the number of replies obtained and thus higher is the message overhead. However, increasing the value of T_{rep} introduces higher latencies into the system. We also observe that with higher mobility, latencies go down drastically.

- We end this section by noting the following:
- The rate of increase of the TTL values should depend on the mobility of the nodes. Higher the mobility, lower should be the rate of increase.
- The initialization of the TTL field should depend on the network density and the threshold value. For sparse networks and/or high threshold, TTL should be set to a high initial value to avoid sending multiple requests. For dense networks and/or low threshold values, TTL can be initialized to a low initial value as the replies can be obtained from the immediate neighborhood.
- The T_{rep} values should be fine tuned to meet specific network densities according to the required latencies and message overhead.
- We have not investigated the above points in detail in this work and they constitute our future work.

5. CONCLUSION

In this work we have demonstrated a distributed symmetric key exchange mechanism by sharing polynomials at fixed points using Shamir's *t*-threshold scheme. Using this we have shown how a distributed scheme can provide an incoming node with the keying material. Our scheme is secure to less than $t+1$ nodes getting compromised. We have shown through simulations that the message overhead and the latencies involved in the key exchange process is well within bounds of other similar protocols. Also, since no key announcements are needed and no session keys have to be established, our scheme has a much better runtime performance.

Our idea can also be extended to provide a hierarchical key generation mechanism based on the level of trust that a node wishes to provide to another node. Although we have not developed this idea, it can provide grounds for future work.

ACKNOWLEDGEMENTS

This work has been partially supported by the Ohio Board of Regents Doctoral Enhancement Funds and the National Science Foundation, under grant CCR-0113361.

REFERENCES

1. C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro, M. Yung, Perfectly-secure key distribution for dynamic conferences, *Advancesin Cryptology – CRYPTO* '92, LNCS 740 1993.
2. H. Chan, A. Perrig, D. Song, Random key predistribution schemes for sensor networks, *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 2003.
3. H. Deng, A. Mukherjee, and D. P. Agrawal, Threshold and Identity-based Key Management and Authentication for Wireless Ad Hoc Networks, *IEEE International Conferences on Information Technology (ITCC'04)*, April 5-7, 2004.
4. L. Eschenauer and V.D. Gligor, A key-management scheme for distributed sensor networks, *Proceedings of he 9th ACM Conference on Computer and Communications Security*, November 2002.
5. J. Kong, P. Zerfos, H. Luo, S. Lu and L. Zhang, Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks, *Proceedings of the IEEE 9th International Conference on Network Protocols (ICNP'01),* 2001.
6. A. L. Murphy, G.-C. Roman, G. Varghese, An Exercise in Formal Reasoning about Mobile Communications, Proceedings of the Ninth International Workshop on Software Specifications and Design, IEEE Computer Society Technical Council on Software Engineering, IEEE Computer Society, Ise-Shima, Japan, pp.25-33, April 1998.
7. P. Papadimitratos and Z. Haas, Secure Routing for Mobile Ad Hoc Networks, Proceedings of the SCS Communication Networks and Distributed Systems Modeling and Simulation Conference, January 2002.
8. A. Shamir, How to share a secret, *Communications of the ACM*, Vol. 22, pp. 612-613, November 1979.
9. L. Zhou and Zygmunt J. Haas, Securing ad hoc networks, *IEEE Network*, 13(6):24–30, 1999.