Efficient and Lightweight Polynomial-based Key Management Scheme for Dynamic Networks*

Mohammed Nafi[1,2], Samia Bouzefrane[2], and Mawloud Omar[3]

[1] Laboratoire d'Informatique Médicale (LIMED), Faculté des Sciences Exactes, Université de Bejaia, 06000 Bejaia, Algérie
mohammed.nafi@cnam.fr

[2] CEDRIC Lab, Conservatoire National des Arts et Métiers - CNAM, Paris, France
samia.bouzefrane@cnam.fr

[3] LIGM, ESIEE Paris, Université Gustave-Eiffel, Noisy-le-Grand, France
mawloud.omar@univ-eiffel.fr

Abstract. Wireless sensor networks and Internet of Things (IoT) are part of dynamic networks as new nodes can join while existing members can leave the system at any time. These networks mainly suffer from severe resource constraints like energy, storage and computation, which makes securing communications between nodes a real challenge. Several key establishment protocols have been proposed in the literature. Some of them are based on symmetric polynomials. However, the latter solutions have some limitations, such as the resilience to node capture attacks as well as the storage and computation overheads that are high for constrained nodes. In this paper, we propose a lightweight polynomial-based key management scheme for dynamic networks. The proposed scheme allows nodes to be able to establish secure communications between them, and ensures dynamism by supporting node addition and deletion after the setup phase. It also resists to node capture attack. The performance evaluation shows that our scheme reduces both the storage and computation overheads when compared to other related polynomial-based protocols.

Keywords: Polynomial · Key management · Lightweight.

1 Introduction

A dynamic network can be defined as a distributed system whose topology changes continuously over time, where new nodes join the network while existing members leave it at any time. Ad hoc networks, wireless sensor networks and the Internet of Things (IoT) are some examples of dynamic systems since they allow addition and removal of nodes after the deployment phase. These networks are widely used in many application domains such as military, patient and environmental monitoring, agriculture, smart cities and smart homes,

* Supported by General Directorate for Scientific Research and Technological Development, Ministry of Higher Education and Scientific Research (DGRSDT), Algeria.

etc. The nodes mainly share a wireless communication channel through which they exchange messages. However, this medium is vulnerable to various types of attacks if it is not secured. Securing such a channel should involve the use of cryptographic keys. However, managing these keys is a complex process, due to essentially high resource constraints of some nodes.

Several key predistribution approaches have been proposed in the literature for dynamic networks, especially for wireless sensor networks. Some of them rely on the use of polynomials [1–5], which allows two nodes to establish a pairwise key between them. In these schemes, nodes generally share a common symmetric t-degree polynomial so that any pair of nodes is able to compute the same pairwise key by evaluating the polynomial using as input the identifier of the other node. For instance, Blundo et al. [1] introduced a multi-variate t-degree polynomial-based group key predistribution scheme.

The main drawback of these polynomial-based schemes is that when an adversary physically captures a node, he or she may be able to access its memory and therefore get the shared polynomial. As a result, the adversary could calculate the pairwise key between any pair of nodes by evaluating the polynomial at the identifiers of those nodes. In other words, compromising one node could also affect other communication links between non-captured nodes.

In this paper, we propose a lightweight polynomial-based key management scheme for dynamic constrained networks. Our scheme mainly aims to address the aforementioned limitations of existing polynomial-based schemes. In fact, instead of using the identical polynomial to compute all the secret keys, nodes do not share the same polynomial in our scheme. Accordingly, compromising a node affects the links it shares with its neighbors, but has a little influence on those between non-compromised nodes. In addition, the proposed scheme significantly reduces the storage and computation overheads on the constrained node side. The main features of the proposed scheme are highlighted below:

- Our solution is more resistant to node capture attacks since nodes do not share the same polynomial. In fact, compromising a node has a little influence on the links between two non-captured nodes;
- It reduces the memory overhead. In fact, each node has to store $(d + 1)$ coefficients of a bivariate polynomial instead of $(t + 1)$ shares, where d is the number of neighbors;
- It reduces the computation cost. Indeed, to be able to establish a pairwise key, one node must evaluate only two adequate terms of its polynomial, rather than all the terms of a t-degree polynomial as in the other schemes;
- It enables dynamism following the addition and deletion of nodes. In other words, nodes can join or leave the network at any time. These operations have only impact on the nodes that are in the neighborhood.

The rest of this paper is organized as follows. Section 2 discusses some existing polynomial-based key establishment solutions. Our proposed polynomial-based key management scheme is described in detail in Section 3. Its performance evaluation is presented in Section 4. Section 5 concludes this paper and gives some future research directions.

2 Related work

Various mathematical key predistribution schemes have been proposed in the literature for wireless sensor networks. Some schemes make use of matrices [10–12] while others rely on polynomials [1–5]. In this section, we discuss some schemes belonging to the latter category.

Blundo et al. [1] proposed a non-interactive multi-variate t-degree polynomial-based group key predistribution scheme. This scheme uses symmetric polynomial of m variables, where m is the group size. When the group has only two members ($m = 2$), this scheme degenerates into a symmetric bivariate polynomial-based scheme. To obtain the same pairwise key, a pair of nodes evaluates the polynomial at the identifier of the other. This scheme is t-secure and no communication is required between nodes during the key establishment process. However, each node must store $(t+1)^{(m-1)}$ coefficients, and evaluate all the polynomial's terms when computing a secret key. In addition, compromising more than t nodes leads to the compromise of the entire network.

To improve the resilience to node capture attacks, D. Liu et al. [2] proposed a key predistribution framework based on a pool of polynomials. In this scheme, a set of random bivariate polynomials is used instead of a single polynomial. A subset of polynomials' shares is assigned to each sensor. A pair of nodes establishes a direct key if they have a common polynomial shares. Otherwise, they compute a path key by relying on intermediate nodes. This scheme is more resilient to compromising attacks. However, the storage overhead is increased when compared to the previous scheme.

A. Fanian, et al. [3] introduced a key establishment protocol, called SKEP, based on symmetric t-degree $(k+1)$-variate polynomials, where k is a credential. In their scheme, the network is first divided into separate virtual hexagonal cells. Then, each cell receives a group of sensors, which share the same symmetric t-degree bivariate polynomial. Two members of the same cell establish a direct key, while a pair of sensors from different groups computes an indirect key. In SKEP, each sensor stores the polynomial shares of two distinct t-degree polynomials. This scheme is scalable and resilient to node capture attack. However, the storage overhead remains high when compared to some other related protocols.

J. Zhang et al. [4] proposed a key predistribution scheme for wireless sensor networks that combines both random and polynomial-based key predistribution schemes. Each sensor is assigned with g t-degree polynomials picked from a pool of m elements along with their identifiers. The nodes are also given s keys with their identifiers. Two neighbors can compute a pairwise key if they share the same polynomial or key, or at least a key of one node is derived from the polynomial shares of the other node. This scheme has better resilience to node capture attacks. However, it introduces some additional storage and computation overheads.

A polynomial-based session key establishment scheme in dynamic groups, called NISKC, is presented by V. Kumar et al. in [5]. This scheme uses a multivariate t-degree polynomial, where the number of variables is equal to the group size m. So, each node can compute the session key by putting its private value

in the polynomial. This scheme is efficient in terms of communication overhead, and allows node addition and deletion after the deployment phase. However, the storage and the computation costs are still high.

Authors in [6] were the first to introduce a non-interactive group key distribution scheme with revocation that has the self-healing property. The key idea is that only active members have the ability to recover the missed session keys by combining the key shares received before and after the lost. These key shares are complementary to each other. The scheme allows users to be able to establish a group key in an unreliable environment. In addition, it enables users to join or leave the group as well. However, this scheme requires high storage and communication overheads. Moreover, Blundo et al. in [7] presented an attack against the first construction of this scheme.

Another efficient session key distribution scheme for unreliable wireless sensor networks was proposed in [8]. The scheme has also the self-healing and revocation features as the previous one. Each user stores a t-degree polynomial with the initial or current session identifier. From a broadcast message, each non-revoked user is able to compute the current session key, which is in turn used to recover the self-healing keys. The scheme reduces both the storage and the communication overheads. Moreover, authors mentioned that their scheme also ensures forward and backward secrecy since users that join the network at a subsequent session ignore the initial session identifier. However, authors in [9] showed that this scheme does not really resist against forward and backward attacks.

Note that in most existing polynomial-based key establishment protocols, nodes generally share the same t-degree polynomial. Consequently, if more than t nodes are captured, the whole network will be compromised. Moreover, the storage and computation overheads are considerable for constrained nodes. In this paper, we propose a polynomial-based scheme where nodes do not necessarily share the same polynomial, which makes it more resilient to node capture attacks and efficient in terms of storage and computation overheads.

3 ELiPKMS: The proposed scheme

In this section, we present our proposed scheme, called ELiPKMS (Efficient and Lightweight Polynomial-based Key Management Scheme), which allows nodes to establish secure links in their neighborhood. ELiPKMS consists of six phases, namely the setup, neighbor discovery, key generation, node addition, node deletion and key refresh phases, which are described in detail in the following. The notation used in this paper is summarized in Table 1.

3.1 Setup

The trusted server generates a bivariate symmetric $2n$-degree polynomial over the finite field $F(q)$, where n represents the number of nodes in the network during the deployment phase. The form of the obtained polynomial, let's say $f(x, y) = \sum_{i=1}^{n} a_i x^i y^i$, and the initial network key K_n are then preloaded into the memory of each node before its deployment.

Table 1. Notation

Notation	Description
x, y	Variables of the polynomial
i, j	Identifiers of nodes i and j
a_i	Secret value of node i (coefficient of the polynomial)
$F(q)$	Finite or Galois field of q elements
K_n	Network key
K_{ij}	Pairwise key shared between nodes i and j
t_i	Neighbor table of the node i
d_i	Degree of the polynomial of the node i
d	Degree of the node (number of neighbors)

3.2 Neighbor discovery

Once the deployment is done, each node proceeds to discover other nodes that are within its communication range. To do that, each node i generates a random secret value, denoted a_i, and then broadcasts a *Hello* message encrypted with the initial network key K_n. This message contains the sender's identifier and its generated secret value. Each node i holds a neighbor table t_i wherein it stores its identifier along with its secret value as well as those received from its adjacent nodes (j, a_j), with $j \in t_i$. At the end of this discovery phase, each node i orders its neighbor table in ascending order of the identifiers. Afterwards, it uses the polynomial's form to generate its own bivariate d_i-degree polynomial according to its direct neighbors, where $d_i = 2 * Max(j)$, with $j \in t_i$, as described in the equation Eq.1.

$$f_i(x, y) = \sum a_j x^j y^j, \quad with \quad j \in t_i. \tag{1}$$

where a_j is the secret value of the node j.

3.3 Key establishment

Two kinds of keys are used: direct and indirect keys that are established as in the following.

Direct key When a pair of neighbors i and j want to securely communicate with each other, they first compute a common pairwise key K_{ij} by evaluating only the two appropriate terms of their respective polynomials as follows:
$f_i(i, j) = a_i i^i j^i + a_j i^j j^j$ and $f_j(j, i) = a_j j^j i^j + a_i j^i i^i$, where a_i and a_j are the secret values of the nodes i and j respectively. As a result, both nodes obtain the same symmetric secret key as shown in Eq.2.

$$K_{ij} = f_i(i, j) = f_j(j, i) = K_{ji} \tag{2}$$

Path (indirect) key A node i that needs to communicate in a secure way with a non-neighbor node j, tries to establish a key path via intermediate nodes. To do so, the node i requests assistance from its direct neighbors by sending them a request encrypted with the appropriate direct keys. This request contains the identifier of the sender i as well as its secret value a_i and the identifier of the other node j. A neighbor k that receives such a request, decrypts it with the adequate direct key K_{ki}, and then checks its neighbor table to determine whether or not the node j belongs to its neighbors. If the verification is not successful, the message will simply be ignored. Otherwise, the intermediate node checks the freshness of that message by comparing the received secret value a_i with the one already stored in its neighbor table. If the verification fails, the node discards the message. Otherwise, the intermediate node computes a path key K_{ij} for both nodes i and j in the same way as it is performed between two neighbors by evaluating the corresponding terms of its polynomial. After that, this key is sent back to each of the two nodes within a response message encrypted with the appropriate pairwise keys. Upon receiving the previous message from the intermediate node, both nodes i and j decrypt it and store the path key K_{ij}. The latter nodes are then able to use this indirect key in order to establish secure communication between them.

Example Let's consider a network initially deployed with three nodes, as shown in Figure 1.

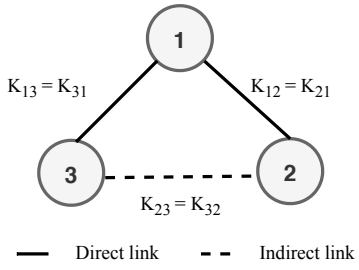

Fig. 1. Network with three nodes

After deployment, the nodes 1, 2 and 3 generate, for instance, the secret values $a_1 = 11$, $a_2 = 22$ and $a_3 = 33$ respectively. At the end of the discovery phase, each node generates its own polynomial as follows:
As the node 1 has two neighbors, hence its polynomial contains three terms:
$f_1(x, y) = a_1x^1y^1 + a_2x^2y^2 + a_3x^3y^3$.
As the degree of the node 2 is equal to one, hence its polynomial has two terms:
$f_2(x, y) = a_1x^1y^1 + a_2x^2y^2$.
The degree of the node 3 is also equal to one, hence its polynomial has two terms too:
$f_3(x, y) = a_1x^1y^1 + a_3x^3y^3$.

The nodes 1 and 2 compute the direct pairwise key by evaluating the appropriate two terms of their polynomials as follows:
Node 1 computes the key $K_{12} = f_1(1,2) = a_1x^1y^1 + a_2x^2y^2 = 11 * 1^1 * 2^1 + 22 * 1^2 * 2^2 = 110$.
Node 2 computes the key$K_{21} = f_2(2,1) = a_2x^2y^2 + a_1x^1y^1 = 22 * 2^2 * 1^2 + 11 * 2^1 * 1^1 = 110$.
In the same way, both nodes 1 and 3 compute their common direct key.
Node 1 computes the key $K_{13} = f_1(1,3) = a_1x^1y^1 + a_3x^3y^3 = 11 * 1^1 * 3^1 + 33 * 1^3 * 3^3 = 924$.
Node 3 computes the key $K_{31} = f_3(3,1) = a_3x^3y^3 + a_1x^1y^1 = 33 * 3^3 * 1^3 + 11 * 3^1 * 1^1 = 924$.

Finally, each pair of neighbors obtains the same direct pairwise key. As a result, they establish a secure channel and can communicate securely. Furthermore, when the node 2 needs to communicate securely with the node 3 that is not inside of its radio range, it solicits the help of the intermediate node 1. The latter computes for them a path key as follows:
$K_{23} = f_1(2,3) = a_2x^2y^2 + a_3x^3y^3 = 22 * 2^2 * 3^2 + 33 * 2^3 * 3^3 = 7920$.

3.4 Node addition

Before deployment, a new node has to request the trusted server to obtain the present network key K_n as well as the current polynomial's form used by the network members. After that, the new node generates a secret value, let's say a_n, and broadcasts a *Join* message that contains especially its identifier with that secret value. The *Join* message is encrypted using the network key K_n. A neighbor that receives this message uses the network key to decrypt it and updates its neighbor table by inserting the identifier and the secret value of the new node. Afterwards, it adds to its polynomial a term that corresponds to that new node, and responds back with an acknowledgment *Ack* message, which contains its identifier along with its secret value, let's say a_i. When receiving the *Ack* messages, the new node updates its neighbor table in turn, and constructs its polynomial. Finally, both the current member and the new node are able to compute a common key by evaluating the adequate two terms of their polynomials as follows: $k_{in} = f_i(i,n) = f_n(n,i) = a_ix^iy^i + a_nx^ny^n = a_nx^ny^n + a_ix^iy^i$.

It is important to note that at each node addition, the polynomials of all neighbors of the new node are increased by one term.

Example Let's assume that the new node 4 joins the network as depicted in Figure 2. During the setup phase, the node 4 is preloaded with the network key K_n and the form of the current polynomial before its deployment in the area of interest. Just after deploying it, that node generates a secret, let's say $a_4 = 44$, then broadcasts in its neighborhood a *Join* message encrypted with the network key. The nodes 1 and 2 are neighbors of the node 4, so they receive that message. These nodes will then decrypt it and update their neighbor table by adding the ID and secret value of the new node 4. After that, they respond back with an *Ack* message containing their IDs along with their generated secret values. Upon

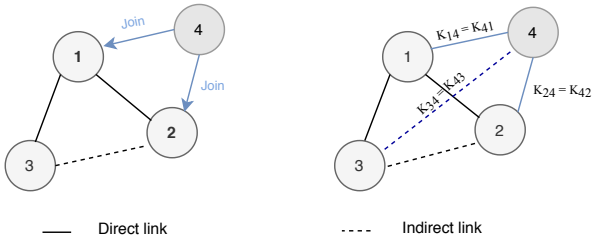

Fig. 2. Node addition

receiving the *Ack* messages, the node 4 decrypts them and adds the IDs as well as these secrets a_1 and a_2 to its neighbor table.
The nodes 1 and 4 respectively and separately compute their direct pairwise key as follows:
$K_{14} = f_1(1,4) = a_1 1^1 4^1 + a_4 1^4 4^4 = 11 * 1 * 4 + 44 * 1 * 256 = 11308.$
$K_{41} = f_4(4,1) = a_4 4^4 1^4 + a_1 4^1 1^1 = 44 * 256 * 1 + 11 * 4 * 1 = 11308.$
Note that nodes 1 and 4 obtain the same pairwise key: $K_{14} = K_{41}$.
The nodes 2 and 4 respectively and separately compute their direct pairwise key as follows:
$K_{24} = f_2(2,4) = a_2 2^2 4^2 + a_4 2^4 4^4 = 22 * 4 * 16 + 44 * 16 * 256 = 181632.$
$K_{42} = f_4(4,2) = a_4 4^4 2^4 + a_2 4^2 2^2 = 44 * 256 * 16 + 22 * 16 * 4 = 181632.$
Note that nodes 2 and 4 also get the same direct pairwise key: $K_{24} = K_{42}$. Moreover, the nodes 3 and 4 are also able to secure their communications by using the path key K_{34}, which can be computed by the intermediate node 1.

3.5 Node deletion

When a node is compromised or leaves the network with its willingness, all its neighboring nodes carry out the following operations:

- Update their neighbor tables by removing the entry corresponding to the leaving node;
- Delete the polynomial's term of that node;
- Erase the pairwise key they share with that node.

It is important to notice that at each node deletion, the polynomials of all nodes adjacent to the outgoing node, are decreased by one term.

3.6 Key refresh

To counter some kinds of security attacks, the network key and the established pairwise keys must be periodically refreshed. To do so, the server node regularly generates a new network key K'_n, and chooses a completely different symmetric polynomial's form. The latter are then transmitted to the network members in a secure way. Upon receiving these new security parameters, all members replace

the old network key and use the received polynomial's form to generate their new symmetric bivariate polynomials. After that, they use the obtained polynomials to compute the new pairwise keys and erase the old keys afterwards.

4 Performance evaluation and comparison

To evaluate the performance of the proposed scheme, we focus primarily on efficiency metrics such as the storage and computation overheads. We compare ELiPKMS with the polynomial-based protocols [3], [4], and [5] described in Section 2. The simulations are developed using MATLAB environment [14]. The nodes are randomly deployed in a square area of 100 m^2. The network size ranges from 10 to 100 nodes. Each node is equipped with a communication radio with a range equal to 10 m. We assume that the number of polynomials picked from the polynomial pool is equal to one ($g = 1$) in [4], while the t-degree polynomial used in [5] is bivariate. Note that these two last parameters are set to their minimum values. The simulation parameters are summarized in Table 2.

Table 2. Simulation parameters

Parameter	**Value**
Network area	100 m × 100 m
Radio range	10 m
Network size	varies from 10 to 100 nodes
$F(q)$, $F(q')$	Finite fields of 100 elements each
Polynomials' degree	$t = 10$
Number of selected polynomials	$g = 1$
Number of polynomials' variables	$m = 2$

4.1 Storage overhead

In the simulations, we are interested in the storage space occupied by both polynomials shares and their identifiers (IDs). We assume that these coefficients and IDs are chosen from two pools of 100 elements each. Moreover, the polynomial degree is set to 10 in the other schemes. Figure 3 shows the storage overhead in function of the network size. We can clearly see that the memory space required to store polynomials' shares and IDs increases with increasing the number of nodes in the network. However, ELiPKMS requires less storage cost when compared to the other schemes. This is because in ELiPKMS, each node needs to store $(d + 1)$ polynomial's coefficients, whereas in the other schemes, each node must save at least $(t + 1)$ polynomial's shares. Accordingly, the total storage space is considerably reduced in our scheme.

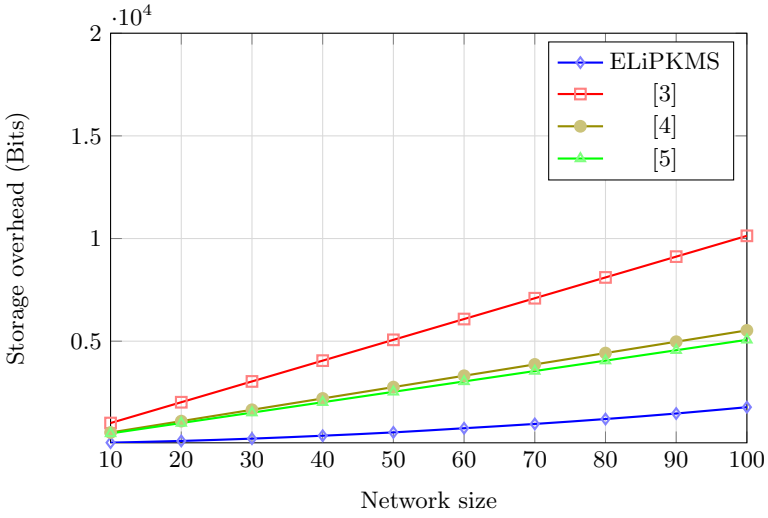

Fig. 3. Storage overhead vs. network size (t=10, g=1)

4.2 Computation overhead

In the experiments, we focused on the computation cost in terms of the number of polynomials' terms evaluated when each node establishes one pairwise key. Figure 4 illustrates the computation overhead depending on the network size. From the figure, we note that ELiPKMS largely outperforms the other schemes. This is due to the fact that unlike the most existing polynomial-based schemes where a node evaluates all the terms of the same t-degree polynomial in order to compute one pairwise key, in our scheme, each node needs to evaluate only two adequate terms of its polynomial. In this way, ELiPKMS significantly reduces the computation cost at the constrained node side.

4.3 Comparison

A fair comparison between our scheme and the related works discussed in Section 2 is shown in Table 3. This comparison is made according to the storage and computation overheads. As mentioned before, the storage cost is estimated based on the memory space occupied by polynomials' coefficients and identifications, while the computation overhead is related to the number of polynomials' terms evaluated during the pairwise key establishment. According to [13], each node has to store $(t+1)^{(m-1)}$ coefficients for a t-degree polynomial of m variables. We have assumed that the polynomial shares are chosen from the same finite field $F(q)$, and the identifications are selected from $F(q')$, with q and q' are two large numbers. The value s represents the number of communication sessions considered in [6].

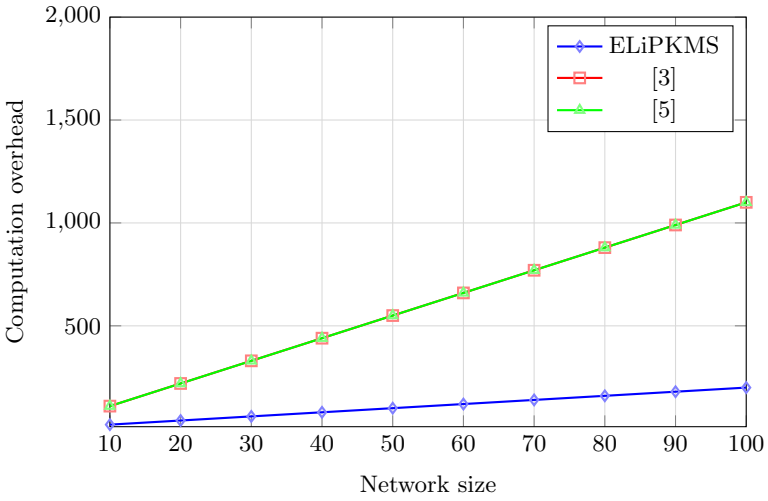

Fig. 4. Computation overhead vs. network size (t=10, m=2)

Table 3. Comparison

Scheme	**Storage**	**Computation**
[1]	$(t+1)^{(m-1)} * log\ q$	$(t+1)^{(m-1)}$
[2]	$s'(t+1) * log\ q + s' * log\ q'$	$(t+1)$
[3]	$2(t+1) * log\ q$	$(t+1)$
[4]	$g(t+1) * log\ q + g * log\ q'$	$\leq (t+1)$
[5]	$(t+1)^{(m-1)} * log\ q$	$(t+1)^{(m-1)}$
[6]	$s * (t+1) * log\ q$	$(t+1)$
[8]	$(t+1) * log\ q$	$\geq (t+1)$
ELiPKMS	$(d+1) * log\ q$	2

5 Conclusion

In this paper, we have proposed an efficient and lightweight polynomial-based key management scheme in dynamic networks with high resource constraints. Compared to existing polynomial-based protocols, our proposed scheme ELiPKMS significantly reduces the amount of key material stored in the memory of each node as well as the computation overhead since the number of multiplications and additions operations required to generate a pairwise key is reduced.

As future work, we first intend to assess the energy consumption of the proposed scheme during communication and computation operations, and then expand the security analysis by including a formal verification using AVISPA tool [15], in order to check whether this scheme resists well to some known attacks and satisfies the secrecy, integrity and authentication properties.

References

1. Blundo, C.: Perfectly-secure key distribution for dynamic conferences. Annual international cryptology conference, Springer, 471–486 (1992)
2. Liu, D.: Establishing pairwise keys in distributed sensor networks. ACM Transactions on Information and System Security (TISSEC), 41–77 (2005)
3. Fanian, A.: An efficient symmetric polynomial-based key establishment protocol for wireless sensor networks. ISeCure-The ISC International Journal of Information Security, 89–105 (2010)
4. Zhang, J.: Key establishment scheme for wireless sensor networks based on polynomial and random key predistribution scheme. Ad Hoc Networks, Elsevier, 68–77 (2018)
5. Kumar, V.: Polynomial based non-interactive session key computation protocol for secure communication in dynamic groups. International Journal of Information Technology, Springer, 283–288 (2020)
6. Staddon, J.: Self-healing key distribution with revocation. In: Proceedings 2002 IEEE Symposium on Security and Privacy. pp. 241–257. IEEE, (2002)
7. Blundo, C.: Design of Self-Healing Key Distribution Schemes. Designs, Codes and Cryptography **1**(32), 15–44 (2004)
8. Mukhopadhyay, S. : Improved Self-Healing Key Distribution with Revocation in Wireless Sensor Network. In: 2007 IEEE Wireless Communications and Networking Conference (2007).
9. Daza, V.: Flaws in some self-healing key distribution schemes with revocation. Information processing letters, 523–526. (2009)
10. Nafi, M.: Matrix-based key management scheme for IoT networks. Ad Hoc Networks, Elsevier, vol. 97, p. 102003 (2020)
11. Blom, R.: An optimal class of symmetric key generation systems. Workshop on the Theory and Application of Cryptographic Techniques, Springer, 335–338 (1984)
12. Du, W.: A key management scheme for wireless sensor networks using deployment knowledge.IEEE INFOCOM 2004, IEEE, vol. 1 (2004)
13. Harn, L.: Predistribution Scheme for Establishing Group Keys in Wireless Sensor Networks.IEEE Sensors J., vol. 15, no. 9, pp. 5103–5108 (2015).
14. MathWorks, https://www.mathworks.com/. Last accessed 20 August 2020
15. Armando, A.: The AVISPA tool for the automated validation of internet security protocols and applications. In: International conference on computer aided verification, pp. 281–285. Springer, (2005)