Hindawi Publishing Corporation
Journal of Sensors
Volume 2016, Article ID 5071617, 15 pages
http://dx.doi.org/10.1155/2016/5071617

Research Article

Disjoint Key Establishment Protocol for Wireless Sensor and Actor Networks

AtaUllah Ghafoor,[1,2] Muhammad Sher,[2] Muhammad Imran,[3] and Imran Baig[4]

[1]*Department of Computer Science, National University of Modern Languages, Islamabad 44000, Pakistan*
[2]*Department of Computer Science and Software Engineering, International Islamic University, Islamabad 44000, Pakistan*
[3]*College of Computer and Information Sciences, King Saud University, Riyadh 12372, Saudi Arabia*
[4]*Department of Electrical & Computer Engineering, College of Engineering, Dhofar University, 211 Salalah, Oman*

Correspondence should be addressed to AtaUllah Ghafoor; ataullah4us@gmail.com

Received 25 March 2016; Accepted 5 May 2016

Academic Editor: Fanli Meng

Key distribution is essential for providing secure communication between commercial and sensitive applications of wireless sensor and actor networks (WSANs). It becomes more challenging when any of the intermediate sensor nodes is compromised by the adversaries as the messages carrying secure keys will be exposed and links will be unreliable. This paper presents a Disjoint Key Establishment Protocol (DKEP) that does not require transmitting keys across the nodes. In DKEP, each node is preloaded with one row and one column from a matrix. After the deployment, indices for row and column are exchanged between the two nodes and values at intersection of row and column index will be used to calculate the key on each node. DKEP is verified by performing formal analysis using Rubin Logic and validated using simulations in NS-2. Simulation results demonstrate the effectiveness and efficiency of DKEP compared to contemporary schemes in terms of reducing storage and communication cost and improving resilience against node compromise attacks. Moreover, the proposed scheme is implemented in a group-based mobile application scenario for secure message exchange.

1. Introduction

Recent developments in sensing, actuation, computing, communication, and networking have led to the emergence of wireless sensor and actor networks (WSANs) [1–3] that allow autonomous and intelligent interaction with the environment. These networks employ number of miniaturized sensors with scarce resources (in terms of computation, communication, and energy) besides fewer powerful actor nodes. Sensors continuously monitor an event of interest and report it wirelessly to corresponding actors for coordinated action. Example applications include border protection, search and rescue, fire containment, and autonomous monitoring and maintenance of lifeline infrastructures [4]. Figure 1 depicts a typical WSAN environment. While most of these applications are critical, however, nodes are vulnerable to a number of attacks including node compromising, traffic capturing, and DOS attacks [5, 6]. Among others, secure key establishment is one of the most prominent barriers in deploying WSAN for sensitive applications.

Key establishment is indispensable for secure communication between distant nodes because key is used for encryption and decryption [7, 8]. In many applications, matrix-based key distribution schemes are preferred because transmission of actual key is not required and nodes can independently calculate keys. Most of the existing matrix-based schemes either share randomly selected rows and columns values [9–11] from precalculated secret and public matrices or some values based on a pattern [12] from the matrices.

The main problem in these schemes is that a compromised node can reveal the rows and columns transmitted through it. At intermediate nodes, message is decrypted to plain text and then encrypted again using symmetric key of next node on the path. These messages cannot be encrypted because there is no prior end to end key between two distant nodes. The link is compromised before its establishment

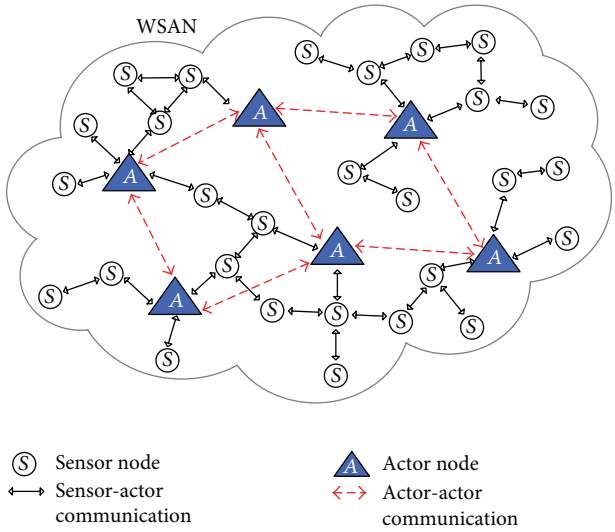


Figure 1: An autonomous WSAN environment for sensitive applications.

and many schemes such as [8–12] suffer from key exposure problem. Moreover, most of the existing schemes only rely on nonformal (i.e., simulations) approaches for validation and hardly employ formal techniques for verifying the correctness.

Formal methods are advanced mathematical mechanism that are based on some computer tools for designing a system model and then apply the formal specifications. The use of formal specifications has evolved in past few years with the advent of new analyzing tools to verify the characteristics of network and security protocols. Informal methods cause a number of ambiguities during the implementation phase where bugs are produced due to misinterpretation of steps [13]. Different formal modeling techniques including BAN Logic [14], PCL [15], and Rubin Logic [16] are used for formalizing the security schemes for the verification of authentication, integrity protection, send, receive, formatting, and key freshness.

This paper presents a Disjoint Key Establishment Protocol (DKEP) for secure communication between nodes. In DKEP, only row and column indices are exchanged along with nonce and timestamp between sender and receiver instead of complete row and column values. After receiving these credentials, sender and receiver identify the common values at the intersection of rows and columns and then obtain the symmetric key taking XOR with exchanged credentials. Both the nodes simultaneously obtain keys in a disjoint manner. It increases the security level because adversary will not be able to reveal the key from the transmitted messages because actual key is never transmitted on the network. We use formal approach to verify the correctness and validate the performance of the proposed protocol. Rubin logic [16] is used for formal modeling of DKEP protocol for verifying and analyzing the operation. Moreover, performance of DKEP is validated using simulation in NS-2.35. Simulation results demonstrate the performance supremacy of DKEP over contemporary schemes in terms of resilience against malicious nodes, storage, and communication costs. The performance results encourage using DKEP for many applications. We also implement DKEP to demonstrate its suitability for securing mobile applications.

The rest of the paper is organized as follows: Section 2 highlights some of the existing schemes related to this work. System model is described in Section 3. Section 4 describes the working of DKEP in various scenarios. Formal modeling and analysis of the proposed protocol using Rubin Logic are presented in Section 5. Simulation results and analysis are discussed in Section 6. Implementation of the proposed protocol on android based mobile phones is discussed in Section 7. Finally, the conclusion and future works are discussed in Section 8.

2. Related Work

Key management is mandatory for secure sharing of information among different nodes in the network by using encryption and decryption. Generally, key management schemes can broadly be categorized into asymmetric and symmetric, while the former provides higher degree of confidence to communicate securely over any open channel since it does not require transmitting private key for message decryption. However, it may not be suitable for resource-constrained sensor nodes as it requires high processing time (i.e., computationally slow) which reduces transmission speed. On the other hand, later is more appropriate for WSN as it uses same shared key for message encryption and decryption. However, secure key distribution over insecure channels is a challenging task especially in mission-critical application as some of the intermediate nodes might be compromised by the adversary. Few recent studies have investigated the challenges of key management in WSN [17, 18]. However, most of the existing key establishment schemes are proposed in the context of wireless sensor networks (WSNs) which may not be directly applicable to WSAN because of various unique characteristics such as actor mobility. Unlike most existing schemes, we focus on devising a matrix-based key establishment scheme that does not require directly exchanging the key between the communicating nodes. Therefore, we limit our discussion on matrix-based symmetric key establishment schemes afterwards.

Blom's proposed a matrix-based symmetric key establishment scheme that contained a public matrix G with a size of $N \times (\lambda + 1)$, where N is network size and λ is level of security because Blom's scheme was λ-secure [9]. Secret matrix D of size $(\lambda+1)\times(\lambda+1)$ and matrix $A = (D \cdot G)^T$ of size $N\times(\lambda+1)$ is generated where T is transpose. Sink node preloads a row from a matrix A and a column from matrix G in each sensor. During key K_{ik} establishment between sender S_i and receiver S_k, each node exchanges its column with neighbors in a plain text. Key is calculated by multiplying A's row with G's column as described in (1), where r and m are row and column indices. Due to symmetric matrix, keys calculated at both nodes are the same: that is, $K_{ik} = K_{ki}$:

$$K_{ik} = [A_{r,1}, A_{r,2}, \ldots, A_{r,\lambda+1}] \begin{bmatrix} G_{1,m} \\ \vdots \\ G_{\lambda+1,m} \end{bmatrix}. \quad (1)$$

Blom's scheme is λ-secure and an adversary needs to compromise λ nodes to compromise rows of matrix A for calculating a number of keys between any two nodes in the network. Blom's scheme increases memory storage, communication, and computation overheads. However, our scheme only exchange single rows and column indices and hence reduces storage, communication, and computational overheads. Moreover, our scheme is not λ nodes resistant and adversary cannot access other keys by compromising even a large number of nodes. The author in [10] proposed a modified Blom's scheme (MBS) where Vandermonde matrix [19] was replaced by adjacency matrix. It was filled with 1's and remaining 0's were replaced by $q-1$ where q was a prime number. It supposed a network size of 6 nodes with $\lambda = 3$ and $q = 29$. An adjacency matrix of size $N \times N$ with $N = 6$ was generated and then matrix G was created by taking $N \times (\lambda + 1)$ means 6×4 submatrix from the adjacency matrix. A secret matrix D of size $(\lambda + 1) \times (\lambda + 1)$ means 4×4 is generated to calculate $A = (D \cdot G)^T$ of size $N \times (\lambda + 1)$ which means 6×4 where T is transpose as elucidated in (2) which is taken from scheme.

Modified Blom's scheme using adjacency matrix is as follows:

$$\underset{\text{Adjacency Matrix}}{\begin{vmatrix} 28 & 1 & 1 & 28 & 28 & 28 \\ 1 & 28 & 28 & 1 & 28 & 28 \\ 1 & 28 & 28 & 28 & 1 & 28 \\ 28 & 1 & 28 & 28 & 28 & 28 \\ 28 & 28 & 1 & 28 & 28 & 1 \\ 28 & 28 & 28 & 28 & 1 & 28 \end{vmatrix}} \quad \underset{\text{Public Matrix } G}{\begin{vmatrix} 28 & 1 & 1 & 28 & 28 & 28 \\ 1 & 28 & 28 & 1 & 28 & 28 \\ 1 & 28 & 28 & 28 & 1 & 28 \\ 28 & 1 & 28 & 28 & 28 & 28 \end{vmatrix}} \quad \underset{\text{Secret Matrix } D}{\begin{vmatrix} 3 & 5 & 2 & 7 \\ 5 & 6 & 9 & 1 \\ 2 & 9 & 3 & 5 \\ 7 & 1 & 5 & 4 \end{vmatrix}} \quad \underset{A = (D \cdot G)^T \bmod 29}{\begin{vmatrix} 26 & 9 & 5 & 24 \\ 3 & 20 & 24 & 5 \\ 18 & 18 & 14 & 26 \\ 22 & 20 & 28 & 14 \\ 16 & 26 & 16 & 22 \\ 12 & 8 & 10 & 12 \end{vmatrix}}. \tag{2}$$

Sink node loads only a single row from matrix A where row id equals sensor id; that is, second row is loaded in sensor 2. In this scheme, column from adjacency matrix is not loaded in sensor because it can be calculated at node during key establishment. Sender S_i can set up a key by multiplying row R_i and column C_k whereas receiver S_k multiplies row R_k and column C_i where i and k are node ids and indices for rows and columns as well. It reduces storage overhead but the scheme is still λ-secure. Moreover, if an adversary captures a row of matrix A from some compromised node then key could be calculated by multiplying with column from identity matrix. Column id can be extracted from sender or receiver ID.

Khan et al. proposed a symmetric key establishment scheme [20] where a symmetric matrix is used along with a generator matrix having maximum rank distance (MRD) codes. A symmetric matrix D of size $k \times k$, where k represents number of symbols over a finite field. Sensors of count n are deployed in t groups where each group contains $(k - \lambda)$ nodes with $\lambda \geq 1$. It also prepares t number of generating matrix G with a size of $k \times N$ where N represents degree of finite field. First group will get a vector g_1 of N elements from this matrix and same is the case for other $(t - 1)$ groups. After that key spaces matrix $A_i = (DG_i)^T$ is generated for $i = 1, 2, 3, \ldots, t$. Sensors of particular group are preloaded with a row from respective matrix A_i and a seed value from secret matrix G. After deployment, each sensor broadcasts its ID and a seed for the column from the matrix G. Each receiving sensor calculates the column from the provided seed value and then multiplies it with its own row preloaded from matrix A to calculate the key. Group formation improves the connectivity and reduces storage cost for matrix values and set up a link key between sender and receiver. It achieves node joining mechanism without changing existing values stored at nodes. Instead of seed, our scheme only transmits row and column indices to save communication cost. This scheme is limited for key establishment between the groups only and cross group or across the network communication will require calculating new matrices. Our scheme achieves cross WSN key establishment with very little storage cost. Chances of duplication are eliminated because in case of same timestamp, there is different nonce and cross sectional values that are used to generate a different key. Matrix of same values and size like $\alpha \times \alpha$ should be loaded in sensor located at different deployments of WSNs where value of α is fixed in all categories of networks.

The authors in [21] devised a naïve scheme for key distribution using unital design theory where a unital in geometry is represented as set consisting of $n^3 + 1$ points that are divided into subsets having a size of $n + 1$. It also imposes a condition that all pairs of distinct points of main set should be present in one subset. A unital is represented as $2 - (n^3 + 1, n + 1, 1)$ design where author used a $2 - (9, 3, 1)$ matrix with 9 points and 3 subsets with each pair of unique points existing in exactly one subset. The basic unital design and mapping to keying provide less probabilities in terms of key sharing; therefore, authors improved the unital design for achieving better network scalability and high key sharing probabilities. It achieves a better scalability and connectivity while providing secure key distribution and also reduces storage overhead. Our scheme reduces the risk of low percentage of finding common keys sharing among neighbors for establishing a secret key in larger networks. The proposed model also allows communication among different WSNs that are deployed nearby and rows and columns taken from a common matrix.

Parakh and Kak proposed a Symmetric Matrix-based Keying (SMK) scheme that preloads a sensor node with a row-column pair from X and Y matrices and $XY = K$, where K is an $N \times N$ symmetric matrix with N as network size [11]. The same row and column are loaded from two different matrices. For example, 5th row and column are loaded from X and Y matrices. For key establishment, nodes S_i and S_k exchange columns of Y and then calculate the key using (3) where Row_k is the row preloaded at S_k and Col_i is column received from S_i at S_k:

$$K_{ij} = \text{Row}_i \cdot \text{Col}_k = \text{Row}_k \cdot \text{Col}_i = K_{ji}. \quad (3)$$

Matrices X and Y are generated by taking Y as a square and nonsingular matrix and then calculate $X = K \cdot Y^{-1}$. Another approach is to calculate bivariate polynomials where size of X and Y matrices is less than K matrix as illustrated in (4) that is redrawn from scheme.

Calculation of X and Y matrices is as follows:

$$\begin{bmatrix} 1 & 3 & 1 \\ 3 & 4 & 1 \\ 9 & 9 & 1 \\ 4 & 6 & 1 \\ 5 & 1 & 1 \end{bmatrix}_X \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 5 & 3 & 2 & 4 \\ 1 & 3 & 9 & 4 & 5 \end{bmatrix}_Y = \begin{bmatrix} 5 & 8 & 8 & 0 & 7 \\ 8 & 4 & 2 & 4 & 2 \\ 8 & 2 & 1 & 9 & 6 \\ 0 & 4 & 9 & 9 & 0 \\ 7 & 2 & 6 & 0 & 3 \end{bmatrix}_K. \quad (4)$$

Authors also proposed a new Commuting Matrix-based Keying (CMK) scheme to eliminate the use of symmetric matrix K and select matrices X and Y with a condition that $XY = YX$, where matrix Y is symmetric. Each node was preloaded with randomly selected rth row of X and rth column of matrix Y. For key establishment, nodes S_i and S_k exchange columns of matrix Y. S_i calculates key by taking hash of K_{ij} concatenated with K_{ji} as illustrated in

$$\begin{aligned} K &= H\left(K_{ij} \parallel K_{ji}\right), \\ K_{ij} &= \text{Row}_i(X) \cdot \text{Col}_k(Y), \\ K_{ji} &= \text{Col}'_k(Y) \cdot \text{Col}_i(X). \end{aligned} \quad (5)$$

Quorum based key Management Scheme (QKM) [12] requires preloading a subset of key matrix in each sensor. For key establishment, sender and receiver find a common key using preloaded subset of secret matrix as shown in Figure 2 taken from scheme.

After that both the nodes generate random number and then exchange it by encrypting with common key. Shared key is obtained by calculating XOR of common key along with random values using (6). Similarly sensor node y also obtains the shared key that is used for future secure message transmission:

$$\text{SK}_{x\text{-}y} = \text{CK}_{x\text{-}y} \oplus R_x \oplus R_y. \quad (6)$$

Dai and Xu provided a key distribution scheme using LU matrix [22] where two groups of polynomials are selected

Sensor 4

$K_{1,1}$	$K_{1,2}$	$K_{1,3}$	$K_{1,4}$	$K_{1,5}$	$K_{1,6}$	$K_{1,7}$
$K_{2,1}$	$K_{2,2}$	$K_{2,3}$	$K_{2,4}$	$K_{2,5}$	$K_{2,6}$	$K_{2,7}$
$K_{3,1}$	$K_{3,2}$	$K_{3,3}$	$K_{3,4}$	$K_{3,5}$	$K_{3,6}$	$K_{3,7}$

Sensor 6

$K_{1,1}$	$K_{1,2}$	$K_{1,3}$	$K_{1,4}$	$K_{1,5}$	$K_{1,6}$	$K_{1,7}$
$K_{2,1}$	$K_{2,2}$	$K_{2,3}$	$K_{2,4}$	$K_{2,5}$	$K_{2,6}$	$K_{2,7}$
$K_{3,1}$	$K_{3,2}$	$K_{3,3}$	$K_{3,4}$	$K_{3,5}$	$K_{3,6}$	$K_{3,7}$

Figure 2: Common key calculation.

from a pool to create lower and upper triangular matrices L and U, respectively. In case of a 3×3 matrix six polynomials are selected for each L and U matrices. It imposes a condition that the product of L and U should result in a symmetric matrix K. Each sensor is preloaded with one row of L and one column of U where row and column index number should be same like ith row of L and ith column of U. After deployment, any two sensors S_i and R_k initiate the key establishment process by exchanging their rows Row_i and Row_k with each other. After that, S_i obtains the key $K_{ji} = L_{\text{Row}_j} \times U_{\text{Col}_i}$ and R_k obtains the key $K_{ij} = L_{\text{Row}_i} \times U_{\text{Col}_j}$ by multiplying the row of other sensors with their own column. The row and column are from symmetric matrix; therefore, the same key will be generated at both nodes. Our scheme also loads one row and one column but we relax the restriction that row and column index should be the same. Sensors can have randomly selected row and column indices. Moreover, we do not exchange row or column values because it increases communication overhead; instead we share indices only. In this scheme, row and column values are directly used to calculate the secret keys and compromising a row through traffic analysis attack can expose the large amount of actual keys. In our scheme, row and column values are just used to get a common value and then nonce and time stamp are also used to generate a unique key which is not entirely dependent of row and column values.

In existing schemes, communication across the WSAN is not supported due to confined set of key pools that support specific number of network size. After that, the chances of key duplication increases. Size of matrix increases for larger networks and hence becomes infeasible in terms of storage for ordinary sensor nodes. Our scheme will resolve it by using a matrix with a fix size and values like a WSN_1 with 2000 nodes and WSN_2 with 1400 nodes. The matrix size and values should remain the same across the network.

3. System Model

This section elaborates the possible key exchange scenarios. We consider WSAN model in which actors are responsible for managing sensors in their cluster. Both sensors and actors are randomly deployed and they exchange security credentials once the network is set up. Sensors are stationary while

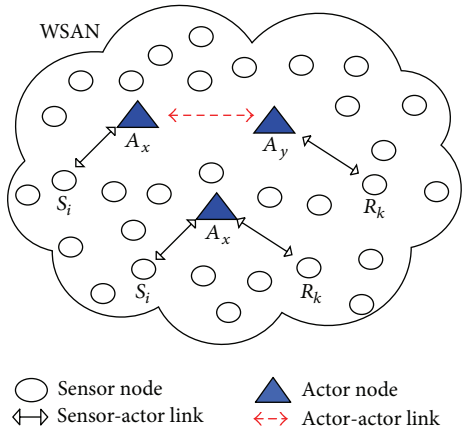


Figure 3: Intra- and intercluster communication scenarios.

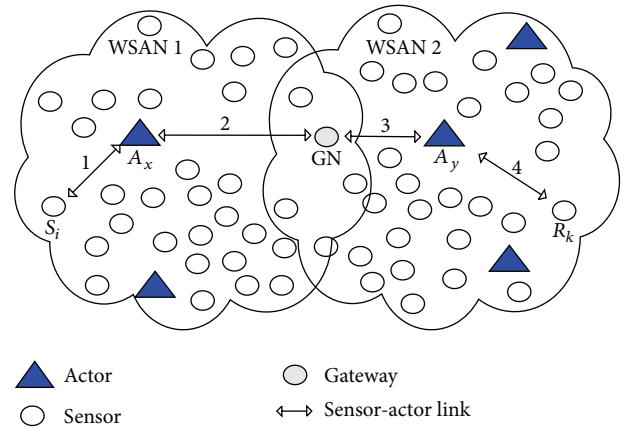


Figure 4: Inter-WSAN key establishment using gateway nodes.

actors can move on demand. Afterwards, any two nodes can distribute the key using DKEP procedure where an actor node is considered first priority to act as the intermediate node between sender and receiver. In other scenarios, ordinary sensor nodes or mixer of both is used as intermediaries between distant sender and receiver. We have considered scenarios where an actor is either stationary or mobile to collect desired data from sensor nodes. We assume that an actor contains the symmetric keys of all the sensor nodes in the network, and the sink node maintains a list of all actors and sensor nodes. It is also assumed that an intrusion detection system is available to detect the malicious nodes.

We consider four possible scenarios for message exchange between sender S_i and receiver R_k containing security parameters. In the first scenario, S_i and R_k can directly exchange messages when both are within direct communication range of each other. In second scenario, an actor A_x serves as an intermediary between S_i and R_k when they are not within range of each other as shown in Figure 3. In third scenario, both S_i and R_k belong to different clusters managed by the actors A_x and A_y, respectively, which require intercluster communication as illustrated in Figure 3.

In the fourth scenario we have considered that multiple WSANs are deployed in a region to perform activities for different categories of applications. Actors A_x and A_y are not in direct communication range and want to communicate for establishing secret key between distant nodes located in different WSAN. In this scenario, an intermediate ordinary sensor node named gateway node (GN) which is located at common boundary of both WSAN can exchange messages to a distant receiver R_k via A_y as illustrated in Figure 4. The GN receives key messages from A_x and A_y and then proceeds with key establishment. It does not use an actor-actor link during such type of communication. Node S_i transmits encrypted message to A_x that decrypts the message and then reencrypts it with a key $E_{K_{A_x\text{-GN}}}$ established between A_x and the GN. Similarly, GN decrypts and reencrypts the message to forward towards actorA_y that further transmit to receiver R_k. After exchanging security parameters, both the nodes simultaneously perform XOR to obtain the keys.

This work is also applicable to mobile environments such as mobile ad hoc networks and vehicular ad hoc networks where same sized matrix can be preloaded on all devices. Most of the modern WSN applications such as [23] have to be integrated with other networks which may require establishing keys across the network. For example, the proposed DKEP can provide such ability to establish keys in these networks.

4. Disjoint Key Establishment Protocol (DKEP)

This section elaborates our proposed DKEP protocol. Key exchange between the communicating parties including sender, receiver, actor, and intermediaries is discussed in this protocol to ensure the secure transmission. Message structure for encryption and decryption procedure is elucidated in stepwise manner along with security parameters that are part of message. Role of message authentication code (MAC) is also highlighted to ensure integrity for each message. Finally key establishment procedure is explored. Moreover, the key establishment scenarios for sensor-sensor and actor-actor are illustrated briefly to show relevant steps of DKEP in those cases. A brief description for all notations used in proposed DKEP is provided in Notations.

In DKEP, security credentials are exchanged using intermediaries to obtain actual key at sender and receiver. We have used a matrix-based approach for key establishment scheme that securely distributes the key without transmitting the actual key on the network. Key distribution begins when rows and columns are taken from $\alpha \times \alpha$ matrix where α is selected as per security requirement and much less than cluster size. Before deployment, each node is preloaded with randomly selected rows and columns. After deployment, rows and column indices and security credentials can be exchanged between the sender and receiver. After receiving the parameters, both nodes decrypt the message, check message freshness by comparing timestamps, and check message integrity by using hash values. After that, common values are located at intersection of row and column indices. Finally the key is obtained simultaneously on both nodes

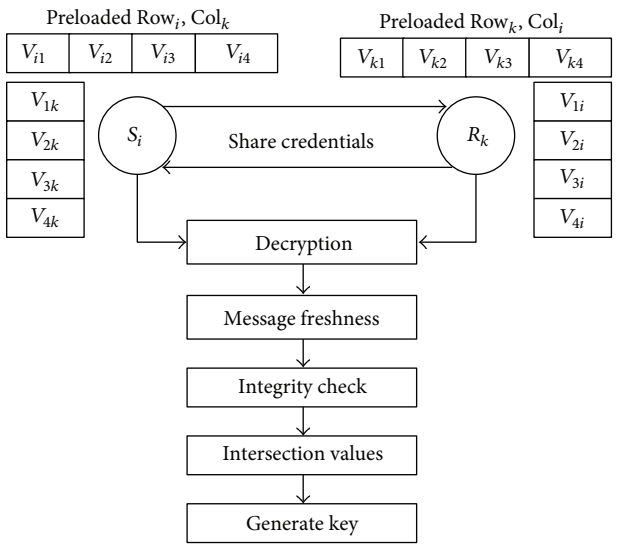


Figure 5: Matrix-based key establishment.

by taking XOR of parameters and the common values as illustrated in Figure 5 where S_i and R_k represent sender and receiver.

This section provides the detailed discussion on secure key distribution protocol for second scenario discussed earlier in Section 3. Key distribution process begins when sender node S_i encrypts message $(\text{Row}_i, \text{Col}_i, \text{ts}_i, \text{rn}_i, R_k, \text{MAC}(\text{Row}_i \parallel \text{Col}_i \parallel \text{ts}_i \parallel \text{rn}_i))$ using preestablished key $E_{K_{S_i\text{-}A_x}}$ between sender S_i and actor A_x as illustrated in (7). In the message, Row_i is row index, Col_i is column index, ts_i is timestamp, rn_i is the nonce from node S_i, and hash of values is also concatenated:

$$C_1 = E_{K_{S_i\text{-}A_x}}\left(\text{Row}_i, \text{Col}_i, \text{ts}_i, \text{rn}_i, R_k, \text{MAC}\left(\text{Row}_i \parallel \text{Col}_i \parallel \text{ts}_i \parallel \text{rn}_i\right)\right). \tag{7}$$

Sender node S_i sends the message (ID_{S_i}, C_1) to the actor node A_x. The message is decrypted by actor A_x to extract the values as illustrated in

$$\left(\text{Row}_i, \text{Col}_i, \text{ts}_i, \text{rn}_i, R_k, \text{MAC}\left(\text{Row}_i \parallel \text{Col}_i \parallel \text{ts}_i \parallel \text{rn}_i\right)\right) = D_{K_{S_i\text{-}A_x}}\left(\text{ID}_{S_i}, C_1\right). \tag{8}$$

Actor node A_x checks the message freshness by calculating difference of system's timestamp with ts_i and then comparing with threshold value. In case of successful result, actor A_x checks integrity of message by concatenating and then taking hash of values to compare it with hash value in the message. Finally the message is reencrypted using key $E_{K_{A_x\text{-}R_k}}$ preestablished between actor A_x and receiver R_k as shown in

$$C_2 = E_{K_{A_x\text{-}R_k}}\left(\text{Row}_i, \text{Col}_i, \text{ts}_i, \text{rn}_i, R_k, \text{MAC}\left(\text{Row}_i \parallel \text{Col}_i \parallel \text{ts}_i \parallel \text{rn}_i\right)\right). \tag{9}$$

Actor node A_x sends the message (ID_{A_x}, C_2) to the receiver R_k. The message is decrypted by receiver R_k to extract the values as illustrated in

$$\left(\text{Row}_i, \text{Col}_i, \text{ts}_i, \text{rn}_i, R_k, \text{MAC}\left(\text{Row}_i \parallel \text{Col}_i \parallel \text{ts}_i \parallel \text{rn}_i\right)\right) = D_{K_{A_x\text{-}R_k}}\left(\text{ID}_{A_x}, C_2\right). \tag{10}$$

Similarly receiver R_k validates the freshness and integrity of message. In case of success, R_k sends a message $(\text{Row}_k, \text{Col}_k, \text{ts}_k, \text{rn}_k, S_i, \text{MAC}(\text{Row}_k \parallel \text{Col}_k \parallel \text{ts}_k \parallel \text{rn}_k))$ after encrypting with key $E_{K_{R_k\text{-}A_x}}$ preestablished between sender R_k and actor A_x as illustrated in (11). In the message, Row_k is row index, Col_k is column index, ts_k is timestamp, rn_k is the nonce from node R_k, and hash of these values is also concatenated:

$$C_3 = E_{K_{R_k\text{-}A_x}}\left(\text{Row}_k, \text{Col}_k, \text{ts}_k, \text{rn}_k, S_i, \text{MAC}\left(\text{Row}_k \parallel \text{Col}_k \parallel \text{ts}_k \parallel \text{rn}_k\right)\right). \tag{11}$$

Receiver R_k responds to the actor A_x with a message (ID_{R_k}, C_3) for forwarding to node S_i. The message is further decrypted by the actor A_x to extract the values as illustrated in

$$\left(\text{Row}_k, \text{Col}_k, \text{ts}_k, \text{rn}_k, S_i, \text{MAC}\left(\text{Row}_k \parallel \text{Col}_k \parallel \text{ts}_k \parallel \text{rn}_k\right)\right) = D_{K_{R_k\text{-}A_x}}\left(\text{ID}_{R_k}, C_3\right). \tag{12}$$

Actor A_x checks freshness and integrity of the message. In case of success a message is sent towards S_i after encrypting it using key $E_{K_{A_x\text{-}S_i}}$ preestablished between actor A_x and S_i as shown in

$$C_4 = E_{K_{A_x\text{-}S_i}}\left(\text{Row}_k, \text{Col}_k, \text{ts}_k, \text{rn}_k, S_i, \text{MAC}\left(\text{Row}_k \parallel \text{Col}_k \parallel \text{ts}_k \parallel \text{rn}_k\right)\right). \tag{13}$$

Actor node A_x sends the message (ID_{A_x}, C_4) to the sender S_i. The message is decrypted by S_i to extract the values as illustrated in

$$\left(\text{Row}_k, \text{Col}_k, \text{ts}_k, \text{rn}_k, S_i, \text{MAC}\left(\text{Row}_k \parallel \text{Col}_k \parallel \text{ts}_k \parallel \text{rn}_k\right)\right) = D_{K_{A_x\text{-}S_i}}\left(\text{ID}_{A_x}, C_4\right). \tag{14}$$

Symmetric key can be obtained after exchanging row and column indices along with timestamp and nonce values. The sender S_i and receiver R_k simultaneously obtain keys by taking XOR of two nonce values $(\text{rn}_i, \text{rn}_k)$ with time stamps $(\text{ts}_i, \text{ts}_k)$ and the common value V_{RC} as illustrated in

$$K_{S_i\text{-}R_k} = \text{rn}_i \oplus \text{ts}_i \oplus V_{\text{RC}} \oplus \text{ts}_k \oplus \text{rn}_k. \tag{15}$$

In this equation, $\oplus$ represents XOR and V_{RC} is the set of values at intersection of rows and columns of two nodes, for

example, row Row_i, column Col_k, and row Row_k, column Col_i as illustrated in

$$V_{\text{RC}} = V_{\text{Row}_i\text{Col}_k} \oplus V_{\text{Row}_k\text{Col}_i}. \tag{16}$$

For example, node S_i contains 5th row and 3rd column and node R_k contains 4th row and 1st column; then $V_{\text{RC}} = V_{\text{Row}_5\text{Col}_1} \oplus V_{\text{Row}_4\text{Col}_3}$. Both the nodes contain the row and column indices of each other; therefore, V_{RC} could be calculated using (16) on both nodes by identifying the values at intersection. Same values are selected at both sender S_i and receiver R_k. Moreover, same key value is obtained at both nodes and key is never transmitted on the network as illustrated in Figure 6. In case of sensor-sensor key distribution scenario, there is no end to end key and messages are not encrypted as illustrated in

$$\begin{aligned} S_i &\longrightarrow R_k : \{\text{Row}_i, \text{Col}_i, \text{ts}_i, \text{rn}_i, R_k, \\ &\quad \text{MAC}(\text{Row}_i \parallel \text{Col}_i \parallel \text{ts}_i \parallel \text{rn}_i)\} \\ R_k &\longrightarrow S_i : \{\text{Row}_k, \text{Col}_k, \text{ts}_k, \text{rn}_k, S_i, \\ &\quad \text{MAC}(\text{Row}_k \parallel \text{Col}_k \parallel \text{ts}_k \parallel \text{rn}_k)\} \\ K_{S_i\text{-}R_k} &= \text{rn}_i \oplus \text{ts}_i \oplus V_{\text{RC}} \oplus \text{ts}_k \oplus \text{rn}_k. \end{aligned} \tag{17}$$

In case of actor-actor key distribution scenario, actors are preloaded with three row and column pairs. During key establishment between A_x and A_y, one row and one column indices are randomly selected to exchange between them. After that, key is calculated using

$$\begin{aligned} A_x &\longrightarrow A_y : \{\text{Row}_x, \text{Col}_x, \text{ts}_x, \text{rn}_x, A_y, \\ &\quad \text{MAC}(\text{Row}_x \parallel \text{Col}_x \parallel \text{ts}_x \parallel \text{rn}_x)\} \\ A_y &\longrightarrow A_x : \{\text{Row}_y, \text{Col}_y, \text{ts}_y, \text{rn}_y, A_x, \\ &\quad \text{MAC}(\text{Row}_y \parallel \text{Col}_y \parallel \text{ts}_y \parallel \text{rn}_y)\} \\ K_{A_x\text{-}A_y} &= \text{rn}_x \oplus \text{ts}_x \oplus V_{\text{RC}} \oplus \text{rn}_y \oplus \text{ts}_y. \end{aligned} \tag{18}$$

In DKEP, the communication is not stopped even if the actor is compromised or damaged. Only the routing paths that were using the actor as intermediary node are affected and an alternate node can be selected for that path by establishing on demand symmetric keys. It also reduces the memory overhead as compared to the existing cluster based schemes because each node does not require $\gamma - 1$ nodes at the beginning. The number of keys stored in the memory grows according to the on demand contact with other nodes of the cluster. This contact can be done only to route the query to neighboring nodes and then route the required data towards the sink node. If a node is compromised and its rows and columns are exposed to the attacker then attacker is not able to get those keys that were established particularly using these rows and columns. These keys were calculated by using one value from matrix and taking its XOR with the two nonce values with timestamp which are not stored in the permanent memory. It means nothing is revealed by retrieving the rows and columns. It also does not affect the keys in other nodes. It achieves much better resilience against node and traffic capturing attacks. In this way the key exposure problem at intermediate node is also resolved. It also ensures that any two distant nodes can establish keys securely and hence achieves much better connectivity. Moreover, the receiver transmits message using disjoint paths that confuse the traffic analysis attacker and strengthens the proposed scheme. If symmetric key is compromised from a node then intruder cannot recover any past keys established earlier or current keys between other nodes.

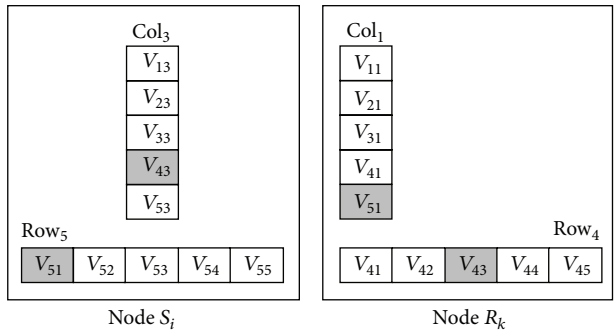


Figure 6: V_{RC} calculation at sender S_i and receiver R_k.

A possible variation in the scheme could be the use of lightweight public key cryptography like Elliptic Curve Cryptography (ECC) to secure only those key exchange messages that are shared between the ordinary nodes where actor nodes are not present nearby. In other scenarios discussed in system model, ordinary nodes encrypt the key exchange messages using preestablished secret keys between ordinary sensors and actor nodes.

5. DKEP Formal Specification

To verify correctness of DKEP, we use Rubin Logic [16] and performed stepwise formal modeling of the scheme. It verifies the proposed protocol for the standardized requirements of cryptographic functions including encryption, decryption, authentication, integrity protection, and freshness of message. Rubin Logic is equally applicable to verify the send, receive, and update operations performed during message exchange between distant nodes. This modeling technique includes the formalization steps that are similar to the flow of programming functions in real implementation of the scheme. A global set is maintained that contains information about entities, their roles, and global variables of protocol. The information saved in global set can be subdivided into observer, rule, secret, and principal sets. The schemes that provide stepwise discussion on formalization [24–26] of WSAN security protocols are illustrated in the form of case studies.

A list of notations is provided earlier in Notations to elaborate the symbols used to describe DKEP protocol. In this section, additional notations are included in Notations to further elaborate the symbols used in local set during formal specification of DKEP.

Table 1: Local set for DKEP.

(1) Sender (S_i)	(3) Receiver (R_k)
$\text{POSS}(S_i) = \{\text{ID}_{S_i}, K_{S_i\text{-}A_x}, \text{RowVal}_i, \text{ColVal}_i\}$	$\text{POSS}(R_k) = \{\text{ID}_{R_k}, K_{A_x\text{-}S_i}, \text{RowVal}_k, \text{ColVal}_k\}$
$\text{BEL}(S_i) = \{\#(\text{ID}_{S_i}), \#(K_{S_i\text{-}A_x}), \#(\text{RowVal}_i), \#(\text{ColVal}_i)\}$	$\text{BEL}(R_k) = \{\#(\text{ID}_{R_k}), \#(K_{A_x\text{-}S_i}), \#(\text{RowVal}_k), \#(\text{ColVal}_k)\}$
$\text{BL}(S_i) = \text{Hash}(h(.); \text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i) \rightarrow H_{S_{i\text{RC}}}$	$\text{BL}(R_k) = \text{Hash}(h(.); \text{Row}_k, \text{Col}_k, \text{rn}_k, \text{ts}_k) \rightarrow H_{R_{k\text{RC}}}$
$\text{Concat}(\text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i, R_k, H_{S_{i\text{RC}}}) \rightarrow P_{S_{i\text{RC}}}$	$\text{Concat}(\text{Row}_k, \text{Col}_k, \text{rn}_k, \text{ts}_k, S_i, H_{R_{k\text{RC}}}) \rightarrow P_{R_{k\text{RC}}}$
$\text{Encrypt}(\{P_{S_{i\text{RC}}}\}K_{S_i\text{-}A_x}) \rightarrow C_1$	$\text{Encrypt}(\{P_{R_{k\text{RC}}}\}K_{R_k\text{-}A_x}) \rightarrow C_2$
$\text{Send}(A_x, \text{Concat}\{\text{ID}_{S_i}, C_1\}) \rightarrow M_1$	$\text{Send}(A_x, \text{Concat}\{\text{ID}_{R_k}, C_2\}) \rightarrow M_2$
$\text{Update}(\text{rn}_i, \text{ts}_i)$	$\text{Update}(\text{rn}_k, \text{ts}_k)$
$\text{Receive}(A_x, \{\text{ID}_{A_x}, C_4\})$	$\text{Receive}(A_x, \{\text{ID}_{A_x}, C_3\})$
$\text{Split}(\{\text{ID}_{A_x}, C_4\})$	$\text{Split}(\{\text{ID}_{A_x}, C_3\})$
$\text{Decrypt}(\{C_4\}K_{A_x\text{-}S_i}) \rightarrow P_{R_{k\text{RC}}}$	$\text{Decrypt}(\{C_3\}K_{A_x\text{-}R_k}) \rightarrow P_{S_{i\text{RC}}}$
$\text{Split}(P_{R_{k\text{RC}}}) \rightarrow (\text{Row}_k, \text{Col}_k, \text{rn}_k, \text{ts}_k, S_i, H_{R_{k\text{RC}}})$	$\text{Split}(P_{S_{i\text{RC}}}) \rightarrow (\text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i, R_k, H_{S_{i\text{RC}}})$
$\text{Freshness}((\text{ts}_{S_i} - \text{ts}_k) \geq \Delta t) \rightarrow \text{Abort}$	$\text{Freshness}((\text{ts}_{R_k} - \text{ts}_i) \geq \Delta t) \rightarrow \text{Abort}$
$\text{MAC}(\text{Concat}(\text{Row}_k, \text{Col}_k, \text{rn}_k, \text{ts}_k)) \rightarrow H^+$	$\text{MAC}(\text{Concat}(\text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i)) \rightarrow H^\wedge$
$\text{Check}(H_{R_{k\text{RC}}}, H^+) \rightarrow \text{Abort}$	$\text{Check}(H_{S_{i\text{RC}}}, H^\wedge)$
$\text{XOR}(V_{\text{Row}_i\,\text{Col}_k}, V_{\text{Row}_k\,\text{Col}_i}) \rightarrow V_{\text{RC}}$	$\text{XOR}(V_{\text{Row}_i\,\text{Col}_k}, V_{\text{Row}_k\,\text{Col}_i}) \rightarrow V_{\text{RC}}$
$\text{XOR}(\text{rn}_i, \text{ts}_i, V_{\text{RC}}, \text{rn}_k, \text{ts}_k) \rightarrow K_{S_i\text{-}R_k}$	$\text{XOR}(\text{rn}_i, \text{ts}_i, V_{\text{RC}}, \text{rn}_k, \text{ts}_k) \rightarrow K_{S_i\text{-}R_k}$
$\text{Hash}(h(.); \text{rn}_k, \text{KeySuccess}) \rightarrow H_{S_i}$	$\text{Hash}(h(.); \text{rn}_i, \text{KeySuccess}) \rightarrow H_{R_k}$
$\text{Concat}(\text{rn}_k, \text{KeySuccess}, H_{S_i}) \rightarrow P_{S_i}$	$\text{Concat}(\text{rn}_i, \text{KeySuccess}) \rightarrow P_{R_k}$
$\text{Encrypt}(\{P_{S_i}\}K_{S_i\text{-}R_k}) \rightarrow C_6$	$\text{Encrypt}(\{P_{R_k}\}K_{S_i\text{-}R_k}) \rightarrow C_5$
$\text{Send}(R_k, \text{Concat}\{\text{ID}_{S_i}, C_6\}) \rightarrow M_6$	$\text{Send}(S_i, \text{Concat}\{\text{ID}_{R_k}, C_5\}) \rightarrow M_5$
$\text{Update}(M_{\text{ID}})$	$\text{Update}(M_{\text{ID}})$
$\text{Receive}(R_k, \{\text{ID}_{R_k}, C_5\})$	$\text{Receive}(S_i, \{\text{ID}_{S_i}, C_6\})$
$\text{Split}(\{\text{ID}_{R_k}, C_5\})$	$\text{Split}(\{\text{ID}_{S_i}, C_6\})$
$\text{Decrypt}(\{C_5\}K_{S_i\text{-}R_k}) \rightarrow P_{R_k}$	$\text{Decrypt}(\{C_6\}K_{S_i\text{-}R_k}) \rightarrow P_{S_i}$
$\text{Split}(P_{R_k}) \rightarrow (\text{rn}_k, \text{KeySuccess}, H_{R_k})$	$\text{Split}(P_{S_i}) \rightarrow (\text{rn}_k, \text{KeySuccess}, H_{S_i})$
$\text{Freshness}((\text{ts}_{S_i} - \text{ts}_k) \geq \Delta t) \rightarrow \text{Abort}$	$\text{Freshness}((\text{ts}_{R_k} - \text{ts}_i) \geq \Delta t) \rightarrow \text{Abort}$
$\text{Check}(\text{MAC}(\text{Concat}(\text{rn}_i, \text{KeySuccess})), H_{R_k})$	$\text{Check}(\text{MAC}(\text{Concat}(\text{rn}_k, \text{KeySuccess})), H_{S_i})$
$\text{Check}(\text{rn}_i, \text{KeySuccess}) \rightarrow \text{Update}(K_{S_i\text{-}R_k})$	$\text{Check}(\text{rn}_k, \text{KeySuccess}) \rightarrow \text{Update}(K_{S_i\text{-}R_k})$
(2) Actor (A_x)	
$\text{POSS}(A_x) = \{\text{ID}_{A_x}, K_{A_x\text{-}S_i}, K_{A_x\text{-}R_k}\}$	$\text{Receive}(R_k, \{\text{ID}_{R_k}, C_2\})$
$\text{BEL}(A_x) = \{\#(\text{ID}_{A_x}), \#(K_{A_x\text{-}S_i}), \#(K_{A_x\text{-}R_k})\}$	$\text{Split}(\{\text{ID}_{R_k}, C_2\})$
$\text{BL}(A_x) = \text{Receive}(S_i, \{\text{ID}_{S_i}, C_1\})$	$\text{Decrypt}(\{C_2\}K_{A_x\text{-}R_k}) \rightarrow P_{R_{k\text{RC}}}$
$\text{Split}(\{\text{ID}_{S_i}, C_1\})$	$\text{Split}(P_{R_{k\text{RC}}}) \rightarrow (\text{Row}_k, \text{Col}_k, \text{rn}_k, \text{ts}_k, S_i, H_{R_{k\text{RC}}})$
$\text{Decrypt}(\{C_1\}K_{A_x\text{-}S_i}) \rightarrow P_{S_{i\text{RC}}}$	$\text{Freshness}((\text{ts}_{A_x} - \text{ts}_k) \geq \Delta t) \rightarrow \text{Abort}$
$\text{Split}(P_{S_{i\text{RC}}}) \rightarrow (\text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i, R_k, H_{S_{i\text{RC}}})$	$\text{MAC}(\text{Concat}(\text{Row}_k, \text{Col}_k, \text{rn}_k, \text{ts}_k)) \rightarrow H^\sim$
$\text{Freshness}((\text{ts}_{A_x} - \text{ts}_i) \geq \Delta t) \rightarrow \text{Abort}$	$\text{Check}(H_{R_{k\text{RC}}}, H^\sim) \rightarrow \text{Abort}$
$\text{MAC}(\text{Concat}(\text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i)) \rightarrow H^*$	$\text{Encrypt}(\{P_{R_{k\text{RC}}}\}K_{A_x\text{-}S_i}) \rightarrow C_4$
$\text{Check}(H_{S_{i\text{RC}}}, H^*) \rightarrow \text{Abort}$	$\text{Send}(S_i, \text{Concat}\{\text{ID}_{A_x}, C_4\}) \rightarrow M_4$
$\text{Encrypt}(\{P_{S_{i\text{RC}}}\}K_{A_x\text{-}R_k}) \rightarrow C_3$	$\text{Update}(M_{\text{ID}})$
$\text{Send}(R_k, \text{Concat}\{\text{ID}_{A_x}, C_3\}) \rightarrow M_3$	
$\text{Update}(M_{\text{ID}})$	

Each entity maintains a local set that is subdivided into possession set POSS( ), belief set BEL( ), seen set, and behavior List BL( ) where detailed discussion is provided in [16, 27]. Local set for DKEP scenario is illustrated in Table 1 that includes the procedural steps performed at sender, receiver, and the actor individually. In this scenario, $\text{POSS}(S_i)$ and $\text{BEL}(S_i)$ sets explore the storage requirements during and after execution of protocol steps at sender node S_i. Computational and communication oriented details of the DKEP are enumerated in $\text{BL}(S_i)$. All these steps are performed individually for each entity including S_i, R_k, and A_x. It also highlights the parameters and operations that are performed during protocol implementation of the scheme using C language for adding it in NS-2 protocols.

5.1. DKEP Analysis and Verification. In this section, DKEP is analyzed for intracluster key distribution scenario where both the distant sensor nodes are in communication range of one actor. In this scenario, initially sender S_i transmits message M_1 to actor node A_x. After the send operation, system calls the update procedure to refresh the rn_i, ts_i values in the observer list as shown below:

(i) $\text{Hash}(h(\cdot); \text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i) \rightarrow H_{S_{i\text{RC}}}$.

(ii) $\text{Concat}(\text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i, R_k, H_{S_{i\text{RC}}}) \rightarrow P_{S_{i\text{RC}}}$.

(iii) $\text{Encrypt}(\{P_{S_{i\text{RC}}}\}K_{S_i\text{-}A_x}) \rightarrow C_1$.

(iv) $\text{Send}(A_x, \text{Concat}\{\text{ID}_{S_i}, C_1\}) \rightarrow M_1$.

(v) $\text{Update}(\text{rn}_i, \text{ts}_i)$.

All the parameters including message M_1, cipher text C_1, hash $H_{S_{i\text{RC}}}$, concatenated string $P_{S_{i\text{RC}}}$, nonce and timestamp values rn_i, ts_i, and row and column indices $\text{Row}_i, \text{Col}_i$ are saved in possession set at S_i as illustrated below. Belief set also appends the message M_1 in the list:

$$\text{POSS}(S_i) = \{\text{ID}_{S_i}, K_{S_i\text{-}A_x}, P_{S_{i\text{RC}}}, \text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i, H_{S_{i\text{RC}}}, C_1, M_1, R_k\}.$$

$$\text{BEL}(S_i) = \{\#(\text{ID}_{S_i}), \#(K_{S_i\text{-}A_x}), \#(M_1)\}.$$

Similarly R_k also transmits the message containing row and column indices along with timestamp and nonce values:

(i) $\text{Hash}(h(\cdot); \text{Row}_k, \text{Col}_k, \text{rn}_k, \text{ts}_k) \rightarrow H_{R_{k\text{RC}}}$.

(ii) $\text{Concat}(\text{Row}_k, \text{Col}_k, \text{rn}_k, \text{ts}_k, S_i, H_{R_{k\text{RC}}}) \rightarrow P_{R_{k\text{RC}}}$.

(iii) $\text{Encrypt}(\{P_{R_{k\text{RC}}}\}K_{R_k\text{-}A_x}) \rightarrow \text{C}_2$.

(iv) $\text{Send}(A_x, \text{Concat}\{\text{ID}_{R_k}, C_2\}) \rightarrow M_2$.

(v) $\text{Update}(\text{rn}_k, \text{ts}_k)$.

Similarly the related parameters are saved in possession set at receiver node R_k as illustrated below:

$$\text{POSS}(S_i) = \{\text{ID}_{S_i}, K_{S_i\text{-}A_x}, \text{Row}_k, \text{Col}_k, \text{rn}_k, \text{ts}_k, H_{R_{k\text{RC}}}, P_{R_{k\text{RC}}}, C_2, M_2, S_i\}.$$

$$\text{BEL}(S_i) = \{\#(\text{ID}_{S_i}), \#(K_{S_i\text{-}A_x}), \#(M_2)\}.$$

After the exchange of messages between sender and receiver, each sensor node splits the message to extract the cipher text. After that, it is decrypted using the shared key between actor and the sensor node S_i to retrieve concatenated string $P_{S_{i\text{RC}}}$ that is further split to extract $\text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i, R_k, H_{S_{i\text{RC}}}$ values. For freshness evaluation of the message, initially timestamp ts_{A_x} is calculated at actor A_x and then received time stamp value ts_i is subtracted from it. A message can be aborted when timestamp difference is larger than the threshold time Δt. Otherwise, node R_k calculates the MAC of $\text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i$ and compares it with hash $H_{S_{i\text{RC}}}$ to ensure integrity protection. Actor node proceeds with next steps by checking that both the hash values are equal otherwise message is discarded. Similar steps are followed by A_x to receive message M_2 from node R_k with different parameters:

$$\text{POSS}(A_x) = \{\text{ID}_{A_x}, K_{A_x\text{-}S_i}, K_{A_x\text{-}R_k}\}.$$

$$\text{BEL}(A_x) = \{\#(\text{ID}_{A_x}), \#(K_{A_x\text{-}S_i}), \#(K_{A_x\text{-}R_k})\}.$$

$\text{BL}(A_x) =$

(i) $\text{Receive}(S_i, \{\text{ID}_{S_i}, C_1\})$,

(ii) $\text{Split}(\{\text{ID}_{S_i}, C_1\})$,

(iii) $\text{Decrypt}(\{C_1\}K_{A_x\text{-}S_i}) \rightarrow P_{S_{i\text{RC}}}$,

(iv) $\text{Split}(P_{S_{i\text{RC}}}) \rightarrow (\text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i, R_k, H_{S_{i\text{RC}}})$,

(v) $\text{Freshness}((\text{ts}_{A_x} - \text{ts}_i) \geq \Delta t) \rightarrow \text{Abort}$,

(vi) $\text{MAC}(\text{Concat}(\text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i)) \rightarrow H^*$,

(vii) $\text{Check}(H_{S_{i\text{RC}}}, H^*) \rightarrow \text{Abort}$.

Actor node A_x further prepares message by encrypting received concatenated string $P_{S_{i\text{RC}}}$ using preestablished key $K_{A_x\text{-}R_k}$ between A_x and receiver node. The encrypted message is transmitted towards R_k and then temporary values $(P_{S_{i\text{RC}}}, \text{rn}_i, \text{ts}_i, R_k, C_3, M_3, H^*)$ used during calculations are removed from memory using Forget procedure:

(viii) $\text{Encrypt}(\{P_{S_{i\text{RC}}}\}K_{A_x\text{-}R_k}) \rightarrow C_3$.

(ix) $\text{Send}(R_k, \text{Concat}\{\text{ID}_{A_x}, C_3\}) \rightarrow M_3$.

(x) $\text{Update}(M_{\text{ID}})$.

(xi) $\text{Forget}(P_{S_{i\text{RC}}}, \text{rn}_i, \text{ts}_i, R_k, C_3, M_3, H^*)$.

Actor node A_x maintains a list of variables and parameters in possession set to perform calculations during key establishment:

$$\text{POSS}(A_x) = \{\text{ID}_{A_x}, K_{A_x\text{-}S_i}, K_{A_x\text{-}R_k}, P_{S_{i\text{RC}}}, R_k, \text{rn}_i, \text{ts}_i, C_3, M_3, H^*\}.$$

$$\text{BEL}(A_x) = \{\#(\text{ID}_{A_x}), \#(K_{A_x\text{-}S_i}), \#(K_{A_x\text{-}R_k})\}.$$

Actor node A_x performs the same operation for message received from R_k to forward towards sensor node S_i with different parameters. The receiver R_k receives the message from A_x and then splits the message to get C_3 which is decrypted to extract the concatenated string. It is further tokenized to get the security parameters including $\text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i, R_k, H_{S_{i\text{RC}}}$. Freshness of message is checked and then comparison of the hash values is performed for integrity protection:

(i) $\text{Receive}(A_x, \{\text{ID}_{A_x}, C_3\})$.

(ii) $\text{Split}(\{\text{ID}_{A_x}, C_3\})$.

(iii) $\text{Decrypt}(\{C_3\}K_{A_x\text{-}R_k}) \rightarrow P_{S_{i\text{RC}}}$.

(iv) $\text{Split}(P_{S_{i\text{RC}}}) \rightarrow (\text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i, R_k, H_{S_{i\text{RC}}})$.

(v) $\text{Freshness}((\text{ts}_{R_k} - \text{ts}_i) \geq \Delta t) \rightarrow \text{Abort MAC}(\text{Concat}(\text{Row}_i, \text{Col}_i, \text{rn}_i, \text{ts}_i)) \rightarrow H^{\wedge}$.

(vi) $\text{Check}(H_{S_{i\text{RC}}}, H^{\wedge})$.

Finally the symmetric key is obtained by calculating XOR of time stamps and nonce values exchanged between sender and receiver. It also includes the value V_{RC} which is obtained by taking XOR of common value at the intersection of rows and columns of sender and receiver as shown below. After that a key success message is exchanged between sender and receiver to confirm the secret key establishment as per the

following steps. At the end, Message ID is updated in the belief set and Forget operation is executed to remove the out of scope values from the memory of R_k:

(vii) $\text{XOR}(V_{\text{Row}_i\text{Col}_k}, V_{\text{Row}_k\text{Col}_i}) \rightarrow V_{\text{RC}}$.

(viii) $\text{XOR}(\text{rn}_i, \text{ts}_i, V_{\text{RC}}, \text{rn}_k, \text{ts}_k) \rightarrow K_{S_i\text{-}R_k}$.

(ix) $\text{Hash}(h(\cdot); \text{rn}_i, \text{KeySuccess}) \rightarrow H_{R_k}$.

(x) $\text{Concat}(\text{rn}_i, \text{KeySuccess}) \rightarrow P_{R_k}$.

(xi) $\text{Encrypt}(\{P_{R_k}\}K_{S_i\text{-}R_k}) \rightarrow C_5$.

(xii) $\text{Send}(S_i, \text{Concat}\{\text{ID}_{R_k}, C_5\}) \rightarrow M_5$.

(xiii) $\text{Update}(M_{\text{ID}})$.

(xiv) $\text{Forget}(\text{Row}_i, \text{Col}_i, \text{Row}_k, \text{Col}_k, H_{S_{i\text{RC}}}, P_{S_{i\text{RC}}}, \text{rn}_k, \text{ts}_k, C_3, M_3, H^{\wedge}, V_{\text{RC}}, H_{R_k}, P_{R_k}, C_5, M_5)$.

Possession set contains the variables and parameters used during calculations regarding key distribution and message extraction as shown below:

$$\text{POSS}(R_k) = \{\text{ID}_{R_k}, K_{R_k\text{-}A_x}, \text{Row}_i, \text{Col}_i, \text{Row}_k, \text{Col}_k, H_{S_{i\text{RC}}}, P_{S_{i\text{RC}}}, \text{rn}_k, \text{ts}_k, C_3, M_3, H^{\wedge}, V_{\text{RC}}, H_{R_k}, P_{R_k}, C_5, M_5, K_{S_i\text{-}R_k}\}.$$

$$\text{BEL}(R_k) = \#(\text{ID}_{R_k}), \#(K_{R_k\text{-}A_x}), \#(K_{S_i\text{-}R_k}).$$

After execution of all steps illustrated above, the newly established key $K_{S_i\text{-}R_k}$ is stored in the belief set. Unnecessary and out of scope values are removed from the possession set and hence the memory of sender node S_i and receiver node R_k. For this purpose, it calls a function $\text{Forget}(\text{Row}_i, \text{Col}_i, \text{Row}_k, \text{Col}_k, H_{S_{i\text{RC}}}, P_{S_{i\text{RC}}}, \text{rn}_k, \text{ts}_k, C_3, M_3, H^{\wedge}, V_{\text{RC}}, H_{R_k}, P_{R_k}, C_5, M_5)$ after successful exchange of key success message between sender and receiver. The sender node S_i receives the cipher text C_5 for the key success message M_5 encrypted with currently established key $K_{S_i\text{-}R_k}$. Node S_i decrypts the cipher text C_5 to get $(\text{rn}_i, \text{KeySuccess})$ and verify the nonce value rn_i as shown in the following steps. Node S_i calls the Forget procedure to clear the out of scope variables from possession set and memory as well:

(i) $\text{Receive}(R_k, \{\text{ID}_{R_k}, C_5\})$.

(ii) $\text{Split}(\{\text{ID}_{R_k}, C_5\})$.

(iii) $\text{Decrypt}(\{C_5\}K_{S_i\text{-}R_k}) \rightarrow P_{R_k}$.

(iv) $\text{Split}(P_{R_k}) \rightarrow (\text{rn}_k, \text{KeySuccess}, H_{R_k})$.

(v) $\text{Freshness}((\text{ts}_{S_i} - \text{ts}_k) \geq \Delta t) \rightarrow \text{Abort Check}(\text{MAC}(\text{Concat}(\text{rn}_i, \text{KeySuccess})), H_{R_k})$.

(vi) $\text{Check}(\text{rn}_i, \text{KeySuccess}) \rightarrow \text{Update}(K_{S_i\text{-}R_k})$.

(vii) $\text{Forget}(\text{rn}_k, \text{ts}_k, C_5, P_{R_k}, H_{R_k})$.

Similarly, the receiver node R_k also receives the cipher text C_6 for the key success message M_6 and performs all the steps illustrated above.

6. Results and Analysis

The proposed model is simulated for a clustered WSAN using NS-2 where an actor is the cluster head. Running times for critical computations are calculated using C++ language with ns-2 and C# in visual studio 2013 as well. In WSAN, nodes and actors are randomly deployed in a region of 1200 × 1200 meters where nodes were static and actors can move within the network field. Cluster sizes are varied from 50 to 500 nodes and 2 to 20 actors were deployed with an increase of one actor per 25 nodes. Queue type is assigned as Queue/DropTail/PriQue as shown in Table 2.

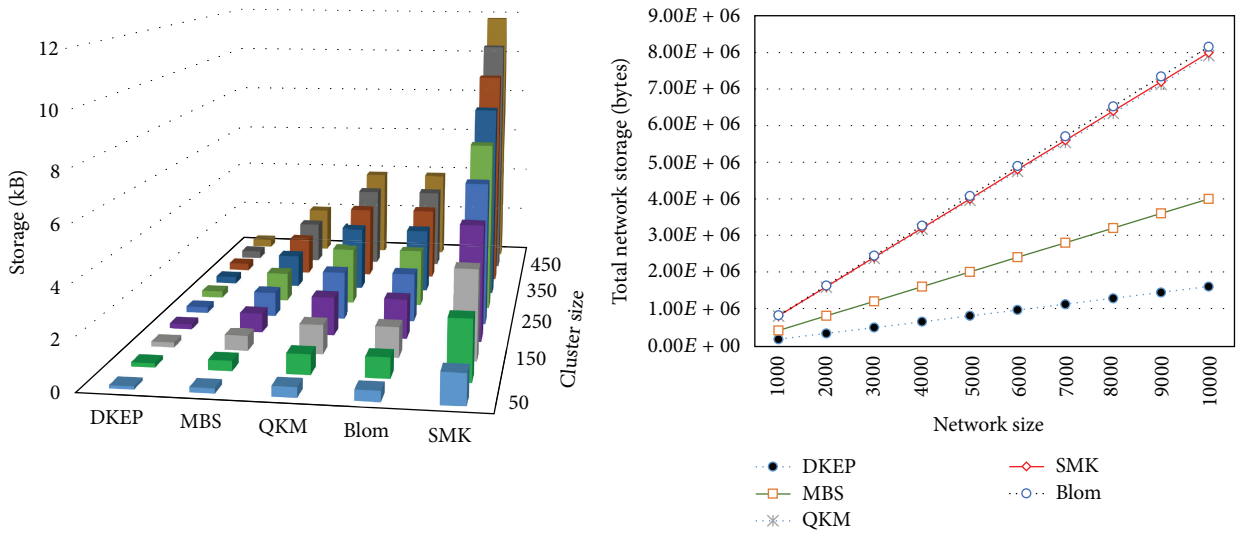
Table 2: Simulation parameters.

Simulation setup	
Parameters	Values
Network field	1200 × 1200 meters
Initial energy at node	1000 J
Tx power at node	0.819 J
Receiving power	0.049 J
Queue type	Queue/DropTail/PriQue
Max packet in queue	50
Routing protocol	DSDV
Agent trace	ON
Router trace	ON
Cluster size	50–500 nodes
Network size	1000–10000
Neighbor count	5–25
Associations count	1–5
Intermediaries count	3,4,5
Captured gateway	1–4
Row column compromised	1 + 1

DKEP scheme is evaluated to measure the resilience, storage, and communication overhead during key establishment for intra- and intercluster scenarios.

6.1. Storage Overhead. Each sensor node is preloaded with matrix elements including row, column, or both. These elements are preloaded for future calculation for key establishment after deployment. In the following section, we have analyzed the storage required for preloading these matrix elements only. Figure 7(a) elucidates the storage per node in bytes for different cluster sizes and fixes network size of 1000 nodes. Different relevant existing schemes need to store key credentials like Blom's scheme [9] that requires $2(\lambda+1)$ values of one row and column. If each value in matrix is S bytes which is considered 8 bytes in this scenario, then $2(\lambda + 1) \times S$ bytes whereas MBS [10] requires $(\lambda + 1) \times S$ bytes that is reduced due to the fact that one column is calculated from identity matrix.

QKM scheme [12] requires $(\gamma - 1) \times S$ bytes where γ represents cluster size and SMK scheme [11] reserves $3(\gamma) \times S$ bytes. Proposed DKEP scheme requires $2 \times \sqrt{\gamma} \times S$ bytes for storing row-column pair. Results prove the dominance of DKEP as compared to preliminaries. Figure 7(b) elucidates the storage per whole network in existing and proposed schemes for different network sizes with same cluster size equal to 100 nodes. Individual node requires needs to store $(\lambda + 1) \times S$ bytes for preloaded key material in MBS [10] and network wide storage cost is $(\lambda + 1) \times S \times N$ bytes, where N



(a) (b)

FIGURE 7: Storage per node is shown in (a) and network wide storage is shown in (b).

represents network size. Similarly the network wide cost of QKM scheme [12] is $(\gamma - 1) \times S \times N$ bytes, SMK scheme [11] reserves $2(\lambda + 1) \times S \times N$ bytes, and proposed DKEP scheme requires $2 \times \sqrt{\gamma} \times S \times N$ bytes. Results prove that proposed DKEP scheme is much more scalable and can be applied to large networks.

Gateway node stores two rows and columns in existing schemes whereas DKEP needs to load only one pair. DKEP requires to load row and column pair from same matrix in all clusters because key is not fully dependent on matrix values and cannot be calculated mathematically even intruder obtains all rows and columns. These stored matrix elements are also refreshed after regular interval to keep the network more resilient against node capturing attacks.

6.2. Communication Overhead. Source and destination nodes exchange messages during key establishment within the cluster. Figure 8(a) illustrates number of bytes transmitted during message exchange for keying between neighboring nodes. Cluster size is equal to 100 nodes and $\lambda = 50$ and number of neighbors are varied from 5 to 25. In QKM scheme [12], two nodes transmit two messages of size S for exchanging their IDs and two more messages of size T for exchanging random values that costs $((2 \times S) + (2 \times T))$ bits. SMK scheme [11] requires to transmit one column of matrix Y from each node that costs total $(2 \times \text{Col}_{\text{Size}} \times \text{EC}_{\text{size}})$ bits where EC_{size} is size of each element in column. MBS [10] requires transmitting one row and indices from each node that exchanges total $(2 \times \text{Row}_{\text{Size}} \times \text{ER}_{\text{size}}) + 4$ bits, where ER_{size} represents size of row elements. Blom's scheme [9] requires transmitting one row and one column that transmits $(2 \times (\text{Row}_{\text{size}} + \text{Col}_{\text{size}}) \times \text{EM}_{\text{size}})$ bits where EM_{size} represents size of matrix elements which is considered 8 bits per value.

DKEP requires $(2 \times P_{\text{size}})$ bits where P_{size} is sum of 16-bit node ID, 16-bit nonce, 16-bit timestamp, 8-bit row, and column index each. Communication cost increases rapidly in SMK, MBS, and Blom's scheme with the increase in cluster size and hence number of neighboring nodes. QKM is suitable in terms of communication cost but it is not scalable due to storage overhead. Results ensure the applicability of DKEP for large cluster sizes and hence larger networks.

During intercluster path key establishment, messages are exchanged between sender S_i and receiver R_k across the cluster where actors A_x and A_y are also involved in this process along with gateway node GN. In this scenario, we have ignored the messages transmission cost between actors and considered the ordinary member nodes that transmit only 4 messages during S_i to A_x, GN to A_y, S_i back to A_y, and from node GN back to A_x transmission. Figure 8(b) elucidates message transmission cost for managing t associations across the cluster where t represents the number of path keys established with neighboring cluster. QKM scheme [12] requires $((4 \times S) + (4 \times T))$ bits, SMK scheme [11] requires $(4 \times \text{Col}_{\text{Size}} \times \text{EC}_{\text{size}})$ bits, MBS [10] requires $((4 \times \text{Row}_{\text{Size}} \times \text{ER}_{\text{size}}) + 4)$ bits, and proposed DKEP requires $(4 \times P_{\text{size}})$ bits for message exchange.

Figure 9 elucidates the cost for establishing the path key using intermediaries in the following communication scenarios including intracluster, intercluster, and inter-WSAN with gateway node. Communication cost $\text{Comm}_{\text{cost}}$ is calculated using (19) where message size Msg_{size} is calculated in such a way that Blom's scheme [9] requires $((\text{Row}_{\text{size}} + \text{Col}_{\text{size}}) \times \text{EM}_{\text{size}})$ bits, QKM scheme [12] requires $(S + T)$ bits, SMK scheme [11] requires $(\text{Col}_{\text{Size}} \times \text{EC}_{\text{size}})$ bits, MBS [10] requires $((\text{Row}_{\text{Size}} \times \text{ER}_{\text{size}}) + 4)$ bits, and DKEP requires P_{size} bits:

$$\text{Comm}_{\text{cost}} = \text{Interm}_{\text{count}} * \text{Msg}_{\text{size}} * \text{MsgCount}_{\text{node}}. \quad (19)$$

The number of intermediate nodes $\text{Interm}_{\text{count}}$ during path key is 3, 4, and 5 for intracluster, intercluster, and inter-WSAN with gateway node based communication scenarios. Each intermediate node sends total 2 messages that is represented as $\text{MsgCount}_{\text{node}}$.

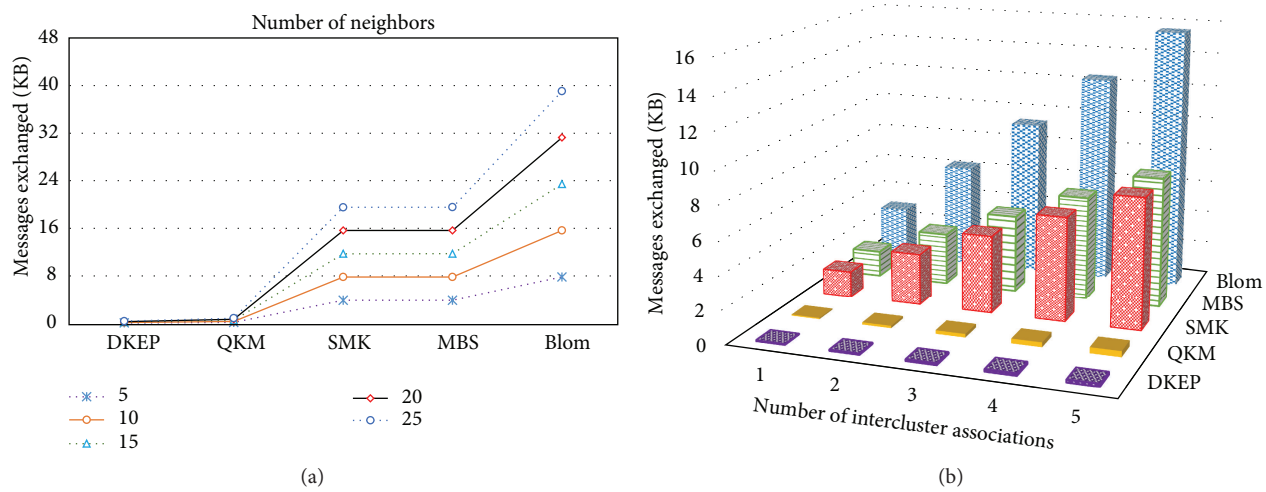


Figure 8: Communication cost for intracluster keying is shown in (a) and intercluster scenario is shown in (b).

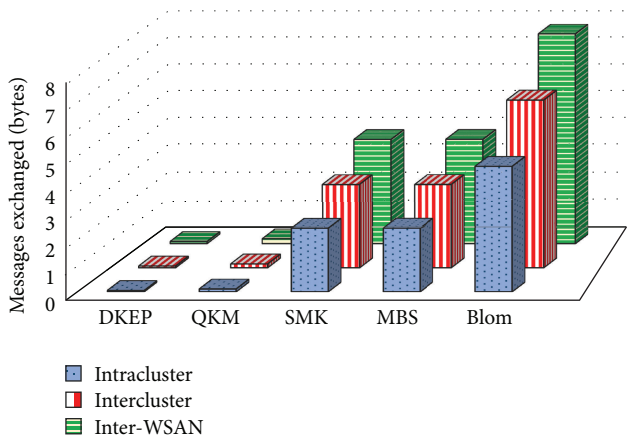


Figure 9: Communication cost for path key distribution.

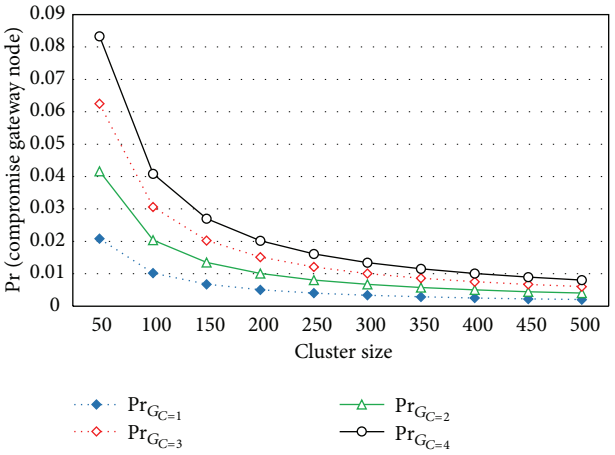


Figure 10: Probability of compromised gateway node.

6.3. Resilience. During inter-WSAN communication, two ordinary nodes from different regions communicate using two actors and gateway node. Probability $\Pr_{\text{Gateway\_Node}}$ to calculate the existence of an uncompromised gateway node in the path can be calculated using (20) where γ is the cluster size and ψ is number of nodes captured by an attacker in the network:

$$\Pr_{\text{Gateway\_Node}} = \frac{\binom{\gamma-3}{\psi}}{\binom{\gamma-2}{\psi}}. \tag{20}$$

Total uncompromised nodes $\gamma - 2$ represent that 2 ordinary nodes are excluded from compromised nodes in network whereas $\gamma - 3$ represents the further exclusion of one gateway node. The probability $\Pr_{\text{G\_Compromised}}$ that a selected gateway node is captured at either sender or receiver side can be calculated using

$$\Pr_{\text{G\_Compromised}} = 1 - \Pr_{\text{Gateway\_Node}} = \frac{\psi}{\gamma - 2}. \tag{21}$$

Figure 10 elucidates the probability of compromising a gateway node during intercluster key establishment scenario for different cluster sizes. It considers scenario for calculating the probability $\Pr_{G_{C=1}}$, $\Pr_{G_{C=2}}$ up to 4 nodes compromised where cluster size varies from 100 nodes to 450 nodes. By compromising 3 nodes in a cluster size of 250 nodes, there is 1 percent chance that gateway node is compromised.

If an ordinary node is compromised then a single row and column pair are revealed to adversary. Probability that a particular row and column are compromised is calculated using (22) where ω rows and columns are compromised, respectively,

$$\Pr_{\text{RC\_Compromised}} = 1 - \frac{\binom{M-1}{\omega-1}}{\binom{M}{\omega}} = \frac{\omega}{M}. \tag{22}$$

A column and row pair is loaded in a node; therefore, compromising an ordinary node reveals one row and column pair in Blom's scheme [9], QKM scheme [12], MBS [10], and proposed DKEP scheme. In case of SMK scheme [11],

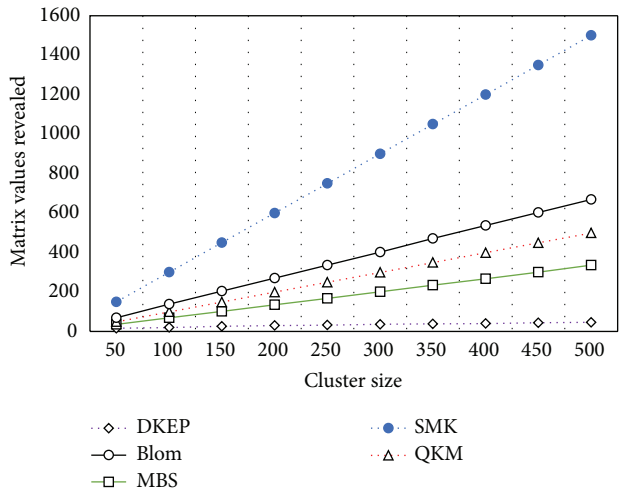


Figure 11: Matrix values revealed on node compromise.

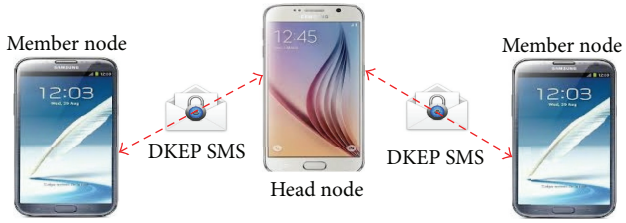


Figure 12: Client-server communication using DKEP.

one row and two columns are revealed. Figure 11 illustrates the exposure of number of matrix values when a node is compromised by an attacker. Network size is equal to 1000 nodes whereas cluster size varies from 100 to 500 and $\lambda = 60$. Results prove the strong resilience of DKEP as compared to existing schemes. It is much secure than these results because in existing schemes exposure of rows and columns results in key exposure as keys are calculated by multiplying row and column values. In DKEP scheme exposure of row and column does not expose the key because values at particular index of row and column are taken as one part of the key. Row and column values are also refreshed after regular intervals to further protect against node capture attacks.

7. DKEP Based Security in Mobile Applications

Proposed scheme is equally applicable in mobile applications for providing security in mobile ad hoc networks. Client and server applications are developed to operate in a peer to peer and cluster based scenario as illustrated in Figure 12. These applications are developed for the proof of the concept and verification of protocol steps in a practical scenario.

Client and server nodes are preloaded with one row and column from $\alpha \times \alpha$ matrix. Member nodes and server mobile exchange row and column indices along with timestamp and nonce values. Keys are established by taking XOR of these security credentials according to DKEP specifications. Established keys and related details are saved in an XML file in flash memory of android mobile. It was tested by establishing keys with four mobiles and then sending encrypted SMS. Application at receiver side reads cipher message and decrypts it using preestablished key. It was also tested for chatting application between two android mobiles. Both client and server application are successfully performing the intended functionalities. DKEP is equally applicable in securing MANET based application scenarios.

7.1. Key Generation Phase. If two nodes do not contain a preestablished key then following steps are followed to establish a secret key between two mobiles. Established keys are saved in an XML and can be retrieved for encrypting or decrypting the SMS for secure communication between mobile nodes. Specifications of DKEP are adopted to establish keys.

Sender Side

(1) Getting Mobile Number from User.
(2) Checking the Mobile Number from XML File.
  (a) If Mobile number already in File
    "Continue Messaging".
  (b) else
    Call DKEP_KeyInit and DKEP_KeyGen functions.
  (c) end if
(3) Procedure DKEP_KeyInit(){
(4) Randomly Generate Rows and Column Indices.
(5) Concatenate Magic Word (MW), indices, time stamp and nonce.
(6) Send SMS.
(7) Saving values in XML File.
(8) }
(9) Procedure DKEP_KeyGen(){
(10) Obtain Magic Word (MW), indices, time stamp and nonce from receiver
(11) Take XOR of these values using DKEP to obtain Key
(12) Save Key in XML file.
(13) }

Receiver Side

(1) Receive SMS.
(2) If first 12 bits equals Magic Word then
(3) Split the concatenated string
(4) Save received values in XML File
(5) Call DKEP_KeyInit function.
(6) Call DKEP_KeyGen function.
(7) else
(8) Discard Message
(9) end if

7.2. Secure Messaging between Cell Phones. A mobile user can communicate securely with mobile phones that are registered earlier through key generation phase. To validate steps of DKEP, user can select the secure option to transmit the message in a cipher text form. SMS is sent towards receiver by concatenating with a magic word where receiver first checks magic word to avoid unnecessary decryption efforts. Plain text message is displayed to user after successful decryption using preestablished key saved in XML file.

Sender Side

(1) Getting Mobile Number and Message of Receiver.

(2) If Mobile number not exist in XML file.

  (a) Start Key Initialization and Generation Process

(3) else

  (a) Encrypt message using pre-established key.
  (b) Converting them in to byte Array.

(4) Concatenate the Magic Word "***DKEP***" with cipher text.

(5) Send SMS

Receiver Side

(1) Receive SMS and split the concatenated message.

(2) If first 10 characters of Message equals "***DKEP***".

  (a) Decrypt the message
  (b) Displaying Original Message.

(3) else

  (a) Displaying Original Message in list.

8. Conclusion

Secure key distribution is the essential requirement of industrial, medical, and military applications of WSN and WSAN for providing secure messaging between sensor nodes. During key distribution between two nodes across the clusters, an intermediate gateway node that has established keys in both clusters across WSAN plays a vital role for message exchange. A compromised gateway node can extract the key before the link establishment because sender and receiver have no prior end to end key to encrypt messages. In this scheme, indices for row and column are exchanged between the sender and receiver nodes and values at intersection of row and column index are used to calculate the key on both nodes. It can establish keys with neighboring nodes, actors, and sink and even across different WSN. Simulation results prove the dominance of DKEP as compared to existing schemes. DKEP reduces communication overhead as compared to existing schemes and achieves resilience against node compromising attacks because key is never transmitted across the nodes. The proposed scheme is verified using Rubin Logic and results are validated with simulations using NS-2. DKEP is also tested for secure messaging between android mobiles in a group-based scenario.

Notations

Notations for DKEP

S_i:	Sender node
R_k:	Receiver node
$K_{S_i\text{-}R_k}$:	Symmetric key between sender and receiver
rn_i, rn_k:	Random nonce values from nodes S_i and R_k
ts_i, ts_k:	Time stamp from nodes S_i and R_k
Row_i:	Row sent by node S_i
Col_i:	Column sent by S_i
V_{RC}:	Values at intersection of row and column
$V_{\text{Row}_i\text{Col}_k}$:	Values at intersection of row Row_i from node S_i and column Col_k from node R_k
$V_{\text{Col}_k\text{Row}_i}$:	Values at intersection of row Row_k from node R_k and column Col_i from node S_i
A_x, A_y:	Actor nodes.

Notations of Local Set for DKEP

RowVal_i, ColVal_i RowVal_k, ColVal_k:	Row and column values stored at sender S_i and receiver R_k
$P_{S_i}\text{RC}$, $P_{R_k}\text{RC}$:	Concatenated string at sender S_i and receiver R_k
M_1–M_6:	Messages 1 to 6
C_1–C_6:	Cipher texts of M_1 to M_6
$H_{S_i}\text{RC}$, $H_{R_k}\text{RC}$:	Hash at S_i and R_k for nonce, time stamp, row, and column indices
H^*, $H^{\sim}$:	Hash calculated at actor A_x for messages received from S_i and R_k
H^+, $H^{\wedge}$:	Hash of messages exchanged between S_i and R_k via actor A_x
H_{S_i}, H_{R_k}:	Hash calculated at S_i and R_k for reply
P_{S_i}, P_{R_k}:	Concatenated value at S_i and R_k for reply.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work is supported by the Deanship of Scientific Research at King Saud University through Research Group no. RG # 1435-051.

References

[1] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[2] I. F. Akyildiz and I. H. Kasimoglu, "Wireless sensor and actor networks: research challenges," *Ad Hoc Networks*, vol. 2, no. 4, pp. 351–367, 2004.

[3] H. Salarian, K.-W. Chin, and F. Naghdy, "Coordination in wireless sensor–actuator networks: a survey," *Journal of Parallel and Distributed Computing*, vol. 72, no. 7, pp. 856–867, 2012.

[4] M. Imran, M. A. Alnuem, W. Alsalih, and M. Younis, "A novel wireless sensor and actor network framework for autonomous monitoring and maintenance of lifeline infrastructures," in *Proceedings of the IEEE International Conference on Communications (ICC '12)*, pp. 6484–6488, Ottawa, Canada, June 2012.

[5] J. Sen, "A survey on wireless sensor network security," *International Journal of Communication Networks and Information Security*, vol. 1, no. 2, pp. 59–78, 2009.

[6] Y. Zhou, Y. Fang, and Y. Zhang, "Securing wireless sensor networks: a survey," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 3, pp. 6–28, 2008.

[7] P. Sundaram, M. Dharshini, and T. Gnanasekaran, "The unceasing detection of adjoining nodes, its connectivity, weakness impact on wireless sensor networks and actor networks," *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 3, no. 1, pp. 1346–1362, 2014.

[8] J. Du, E. Kranakis, and A. Nayak, "Distributed key establishment in disruption tolerant location based social wireless sensor and actor network," in *Proceedings of the 9th Annual Communication Networks and Services Research Conference (CNSR '11)*, pp. 109–116, IEEE, Ottawa, Canada, May 2011.

[9] R. Blom, "An optimal class of symmetric key generation systems," in *Proceedings of the EUROCRYPT 84 Workshop on Advances in Cryptology: Theory and Application of Cryptographic Techniques*, pp. 335–338, 1985.

[10] S. Sukumar, "Computational analysis of modified Blom's scheme," in *Proceedings of the Computing Research Repository (CoRR '13)*, 2013.

[11] A. Parakh and S. Kak, "Matrix based key agreement algorithms for sensor networks," in *Proceedings of the 5th IEEE International Conference on Advanced Networks and Telecommunication Systems (ANTS '11)*, December 2011.

[12] L.-C. Wuu, C.-H. Hung, and C.-M. Chang, "Quorum-based key management scheme in wireless sensor networks," in *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication (ICUIMC '12)*, February 2012.

[13] J. Qadir and O. Hasan, "Applying formal methods to networking: theory, techniques, and applications," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 1, pp. 256–291, 2015.

[14] M. Burrows, M. Abad, and M. Needham, "A logic of authentication," *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 426, no. 1871, pp. 233–271, 1989.

[15] A. Datta, A. Derek, J. C. Mitchell, and A. Roy, "Protocol composition logic (PCL)," in *Electronic Notes in Theoretical Computer Science*, vol. 172, pp. 311–358, 2007.

[16] A. Rubin and P. Honeyman, "Nonmonotonic cryptographic protocols," in *Proceedings of the Computer Security Foundations Workshop (CSFW '94)*, pp. 100–116, Franconia, NH, USA, 1994.

[17] Y. Xiao, V. K. Rayi, B. Sun, X. Du, F. Hu, and M. Galloway, "A survey of key management schemes in wireless sensor networks," *Computer Communications*, vol. 30, no. 11-12, pp. 2314–2341, 2007.

[18] M. A. Simplício Jr., P. S. L. M. Barreto, C. B. Margi, and T. C. M. B. Carvalho, "A survey on key management mechanisms for distributed Wireless Sensor Networks," *Computer Networks*, vol. 54, no. 15, pp. 2591–2612, 2010.

[19] W. H. Press, B. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Vandermonde Matrices and Toeplitz Matrices. 2.8 in Numerical Recipes in FORTRAN: The Art of Scientific Computing*, Cambridge University Press, Cambridge, UK, 1992.

[20] E. Khan, E. Gabidulin, B. Honary, and H. Ahmed, "Matrix-based memory efficient symmetric key generation and predistribution scheme for wireless sensor networks," *IET Wireless Sensor Systems*, vol. 2, no. 2, pp. 108–114, 2012.

[21] W. Bechkit, Y. Challal, A. Bouabdallah, and V. Tarokh, "A highly scalable key pre-distribution scheme for wireless sensor networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 2, pp. 948–959, 2013.

[22] H. Dai and H. Xu, "Key predistribution approach in wireless sensor networks using LU matrix," *IEEE Sensors Journal*, vol. 10, no. 8, pp. 1399–1409, 2010.

[23] J. Wan, J. Liu, Z. Shao, A. V. Vasilakos, M. Imran, and K. Zhou, "Mobile crowd sensing for traffic prediction in internet of vehicles," *Sensors*, vol. 16, no. 1, article 88, 2016.

[24] A. Derhab, A. Bouras, M. R. Senouci, and M. Imran, "Fortifying intrusion detection systems in dynamic Ad Hoc and wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2014, Article ID 608162, 15 pages, 2014.

[25] M. Imran and N. A. Zafar, "Formal specification and validation of a hybrid connectivity restoration algorithm for wireless sensor and actor networks," *Sensors*, vol. 12, no. 9, pp. 11754–11781, 2012.

[26] A. Ghafoor, M. Sher, M. Imran, and A. Derhab, "Secure key distribution using fragmentation and assimilation in wireless sensor and actor networks," *International Journal of Distributed Sensor Networks*, vol. 2015, Article ID 542856, 13 pages, 2015.

[27] P. Kumar, A. J. Choudhury, M. Sain, S.-G. Lee, and H.-J. Lee, "RUASN: a robust user authentication framework for wireless sensor networks," *Sensors*, vol. 11, no. 5, pp. 5020–5046, 2011.