SPIKE: A Novel Session Key Management Protocol with Time-Varying Secure Cluster Formation in Wireless Sensor Networks

Sutirtha Sanyal
ssanyal77@gmail.com

Abstract

In this paper we propose Symmetric Parallel Immalleable Key Establishment (SPIKE), a novel session key distribution protocol for Wireless Sensor Networks (WSN). In the protocol, each node before being deployed sets up a secret key with a central Key Management Server (KMS). After deployment, when a node wants to initiate a secure session with another node, both nodes, in parallel, turn to KMS for mutual authentication. The symmetric authentication steps are designed carefully to thwart several types of security breaches present in WSN environment such as Man-in-the-Middle attack, Replay attack, Amplification attack, Sybil attack, Denial of Service (DoS) attack.

After verification, KMS generates and dispatches the session key to both nodes, simultaneously. Each session key is valid for a certain period. Later when a node wants to communicate with another node which is already part of some secure communication group, KMS includes the new node also in the same session. This incrementally builds up a group of nodes forming a connected component of the original network graph, all sharing the same session key.

In terms of the key storage requirement per node, exactly one session key needs to be maintained in a node at any given instance. This is independent of the network size and the number of concurrent sessions active on that node. In the server side too, session key maintenance overhead becomes minimal as the protocol assigns the same key to all the nodes belonging to a connected component.

1. Introduction

Wireless Sensor Networks (WSN) are gaining wide popularity in solving several day-to-day problems[1], [2], [3] which require monitoring in a continual time basis. Security is a pressing concern for sensor networks. In one hand secure communication among nodes is essential for data critical applications. On the other hand, sophisticated security measures are difficult to implement on these devices since they are resource constrained. Maintaining public key in each node with certificate is costly in terms of storage. Moreover, if a group of nodes communicate with each other, public key of all nodes eventually needs to be stored, in each node. Even if the public key is not used for encryption but only as a session key wrapper, each node still needs to store the certificate. In the case of symmetric encryption, maintaining pairwise session keys for all concurrent sessions is equally inefficient.

Therefore, it is beneficial to assign the duty to a central authority which naturally can generate, distribute and track sessions keys as necessary due to its high storage capacity and faster execution environment. Moreover, with the compulsory involvement of a central key issuing server, sensor nodes can be sure of each other's identity. This in turn can prevent several identity impersonation attacks. We name this server as Key Management Server (KMS). Before deployment, each sensor node is required to set up a secret key and a program which controls the key schedule, with the KMS.

This paper makes the following contributions:

- We propose a novel protocol, Symmetric Parallel Immalleable Key Establishment (SPIKE) to establish session keys in Wireless Sensor Networks. The proposal is symmetric because two communicating nodes, in each step during the authentication phase, exchange messages of same length with the KMS. The proposal also carries out the verification and the key delivery in parallel.
- The proposal requires storage of only *one* session key per node irrespective of the number of concurrent sessions active on that node and the total network size. The key distribution ensures the same session key in all the nodes forming a connected component[1] of the original network. If we represent the network as a graph where nodes represent vertices and a bi-directional channel is represented by an edge between vertices, then the protocol will assign a single session key to each connected component of this graph.
- SPIKE provides protection against several attacks present in WSN environment, viz. Man-in-the-Middle attack, Replay attack, Sybil attack, Amplification attack, Denial of Service (DoS) attack.

1. In this article we will use terms "cluster" and "connected component" interchangeably.

2. Related Work

Key management in wireless sensor networks has been an area of extensive research. Blundo et al. proposed [4] a conference key distribution scheme based on evaluation of symmetric polynomials. Staddon et al. proposed self-healing key distribution [5] where it distributes two sets of polynomials which can be used to recover lost session keys. There are other schemes where a central server broadcast messages to all the nodes and then nodes reconstruct the common key themselves[6], [7]. Some rely on Weil, Tate-Lichtenbaum or Ate pairing based methods[8], [9], [10] while others rely on Diffie-Hellman[11], [12].

Datagram Transport Layer Security (DTLS)[13] is a well known scheme suitable for authenticating storage and bandwidth efficient nodes to each other by exchanging public keys with certificates. This protocol is designed particularly for the UDP style of transport layer. There are other techniques which implement secure encryption and decryption using certificate-less public key[14], [8]. In [8] the central server keeps a master public key ($[s]P$, $s \in \mathbb{Z}$ the master secret, $P \in E(K)$, a known point in an elliptic curve) and generates private key for each node according to its id, I, $[s]I$ where I gets mapped to a point in E. To establish a session key, a node first picks a $r \in \mathbb{Z}$ and calculates the pairing $e([s]P, I)^r$ and forwards $[r]P$ to the other node. It computes $e([r]P, [s]I)$. Due to bilinearity both evaluate same in K^*. Thus the certification (existence of $[s]I$) is implicit. The drawback of the scheme is a different session key needs to be established for each concurrent session.

3. Architectural Background and Protocol Setup

3.1. Architectural Background

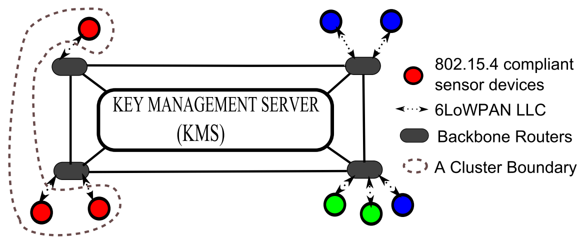


Figure 1. A Representative WSN Architecture

In this section we will describe the underlying architecture of the network and conventions we will follow in this paper to define cryptographic message formats.

The WSN architecture is shown in Figure 1. In this architecture, 802.15.4 compliant sensor nodes are deployed in a dispersed area. These nodes are connected to several edge routers which are deployed in a similar manner along with the nodes (not shown separately in the figure). These edge routers are further connected to backbone routers which finally connect to the KMS.

KMS distributes keys based on requests from nodes. The key distribution protocol ensures that each connected set of nodes receives the same session key. If nodes are outside of all existing communicating groups then KMS generates a new session key. This situation is highlighted in the figure where the color indicates the session key. KMS is protected by a known, certified public key. No certificate thus needs to be exchanged when a node communicates with the KMS using this public key.

Table 1
NOTATIONS USED

Notation	***Meaning***
$\{X\}K$	Message X encrypted with key K.
MAC$\{X\}K'$	MAC of Message X with key K'.
C=$\{X\}K\|\|$MAC$\{C\}K'$	C is the encrypted X with K. Then MAC is computed for C with K' and affixed.
ID_i	Device ID of node i.
K_p	Public key of a public-private key pair of KMS.
K_i^n	Pre-shared secret key to be used by node i, for $n-th$ access.
$\mathbb{P}_i$	The pre-installed program used by node i and KMS, for generating the key-schedule K_i^n.
K_{SS}	Session key.
F	Finite field of operations.
N_i	Nonce from node i to the KMS. It is concatenation of a random elliptic curve over F and a random point on it.
T_{SS}	Validity Period of the Session Key.

3.2. Protocol Setup

Every sensor node which wishes to transmit and receive data securely must first register to KMS. The registration process seeds a node i with the pair $K_i^1, \mathbb{P}_i$ in a secure manner[15]. K_i^1 is the initial pre-shared secret key. $\mathbb{P}_i$ is a program which takes the value of a key and produces another key of same length. Each time, after an authentication phase (successful or not), the current secret key is replaced by the new key generated by $\mathbb{P}_i$, i.e., $K_i^n = \mathbb{P}_i(K_i^{n-1})$. There are several choices for implementing this program. The key expansion routine of AES[16] can be used. Alternatively, initial key could be inferred as a point of large order in an elliptic curve and subsequent keys can be multiples of that point. The algorithm used in the program need not be known by the node. KMS uses the same program $\mathbb{P}_i$ to derive the same key. All computations are carried out in a finite field F. Characteristic and modulus polynomial of the field are public. The abbreviations used are summarized in table I.

3.3. Attacker Model

Adversaries could attempt to impersonate other nodes by forging its identity. If impersonation is successful, the malicious node acts as the man-in-the-middle. It can either be a passive eavesdropper or actively suppress or modify part or all the messages. It may try to launch replay attacks by replaying old messages. It may set up amplification attack by manipulating the source ip address to point to a victim node because source address of node is not checked in UDP. It may attempt denial of service attack by flooding KMS with either bogus or pre-captured messages of valid format. As we will show in section 5, SPIKE is resilient to such attacks. Moreover, the pre-shared key changes according to the program $\mathbb{P}_i$ installed during registration. This gives protection against birthday type of attacks where adversary launches differential cryptanalysis after obtaining several cipher texts encrypted with the same key.

4. Proposed Key Management Protocol

4.1. Session Key Setup

4.1.1. Two nodes without keys. In this section we discuss the protocol operation when both nodes do not have keys initially. The negotiation process is described in the Figure 2(a). First the initiator (Node A) contacts other node (Node B) as shown in the figure. This initiation message is not encrypted and contains its id. If B accepts the request then the procedure moves to the next stage from where active involvement of the Key Management Server (KMS) will be required in all the steps until the delivery of the session key.

As shown in the figure, upon receipt of the acceptance from B, A contacts KMS with an encrypted message (message 3). It contains a nonce (N_A) encrypted with the pre-shared secret password K_A^n. The n denotes the $n-th$ authentication request by A. As described in section 3.2, after an authentication process is over (in A after receiving and in KMS after sending the message 9 or if any one timeouts), K_A^n is modified according to the specific program $\mathbb{P}_A$. N_A is concatenation of a random elliptic curve and a random point on it. Apart from the nonce, A supplies its ID_A and the other peer's id (ID_B) in the request. There is a third field which indicates whether A already has any established session key or not. If it is 0 then the node does not have any session key. If it is set to 1 (as will be the case in the next scenario), then it signifies that the node currently has a valid session key. This entire message is encrypted with the KMS's public key. Also a message authentication code (MAC) is computed for this encrypted message with K_A^n.

Upon receiving this message, KMS cannot immediately check the integrity of the message since it does not know the id of A yet. Therefore, it first decrypts the message with its private key. After that it can retrieve the id of the sender node (ID_A) and the id of the other node (ID_B). Now it can retrieve the secret key corresponding to A and continue to decrypt the next part of the message which is encrypted with K_A^n to recover N_A.

After KMS receives a similar request from the other node (B), it randomly selects an isomorphism $\mathcal{I}$ on the curve provided in N_A and form another nonce denoted as N'_A in Figure 2(a). KMS also sets an additional bit, after the N'_A, to indicate to A that in the next message it will receive the session key if it correctly sends back the recomputed nonce (N''_A) in message 7. If this bit is unset, as could be the case in the following scenarios, then it means the addressed node already has a valid key which it should continue to use and there will be no further communication from KMS. Next, A must retrieve the isomorphism from the nonce KMS supplied in message 5 and use it to compute another curve isomorphic to what was given in message 3. This curve and the corresponding point are returned in message 7. The nonce computation steps are described in detail in the following paragraph.

Using Random Isomorphisms for Nonce Generation. A chooses an elliptic curve E_A over a finite field F and a point P_A on it and encodes it as a nonce N_A in message 3. KMS selects an isomorphism $\mathcal{I} = \{u, r, s, t\} (u \neq 0)$ to get equation of another curve E'_A. The map is

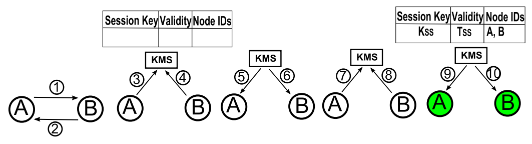
$$\phi(\mathcal{I}) : E_A \to E'_A, \quad (x, y) \mapsto (u^2(x+t), u^3(y + sx + r))$$

To get the image of P_A on E'_A, P'_A, KMS uses the push-forwarding map (same as the inverse map which converts equation of E'_A to E_A)[17].

$$\mathcal{I}_*\{u', r', s', t'\} = \{u^{-1}, (st - r)u^3, -su, -tu^2\}$$

Corresponding $\phi(\mathcal{I}_*)$ sends a point $P_A \in E_A$ to its image $P'_A \in E'_A$. In message 5 KMS sends the $\{\mathcal{I}, P'_A\}$ as nonce N'_A. After receiving 5, A gets back $\mathcal{I}$ to compute E'_A and applies $\phi(\mathcal{I})$ again on E'_A to get E''_A. It also computes $\mathcal{I}_*$ and get the image of P'_A on E''_A, P''_A. In message 7 it sends back $\{E''_A, P''_A\}$ as N''_A.

As an example lets consider an exchange over F_{509^2} with modulus $x^2+508x+2$ and generator denoted by a. The equation for the curve is given by a 5-tuple, $\{a_1, a_2, a_3, a_4, a_6\}$ ($y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$). A selects $E_A = \{64a + 248, 150a + 358, 221a + 373, 478a + 139, 170a + 471\}$ and the point $P_A = \{141a + 148, 334a + 147\}$ and encodes these in 126 bits as N_A in message 3. KMS selects $\mathcal{I} = \{355a + 18, 465a + 372, 301a + 501, 73a + 199\}$ to get $E'_A \cong E_A$, $\{a+45, 134a+302, 341a+452, 225a+447, 4a+392\}$. The image of P_A in E'_A is P'_A, $\{150a+268, 465a+505\}$ obtained via the push-forwarding map $\mathcal{I}_* = \{250a + 275, 306a + 350, 2a + 74, 195a + 443\}$. A checks $\phi(\mathcal{I}_*) : (P_A) \mapsto (P'_A)$. KMS sends $\mathcal{I}$ and P'_A in message 5 as N'_A. In message 7, A forms N''_A by applying ϕ again on E'_A to get E''_A, $\{161a + 152, 93a + 205, 210a + 342, 408a + 100, 43a + 6\}$

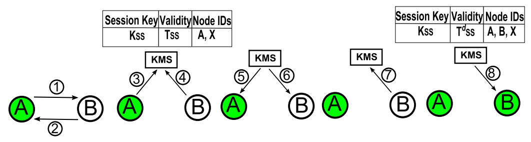


Session Key	Validity	Node IDs

Session Key	Validity	Node IDs
Kss	Tss	A, B

① ID_A ② Accept, ID_B
③ $C=\{ID_A, ID_B, 0, \{N_A\}K_A^n\}K_P \,||\, MAC\{C\}K_A^n$
④ $C=\{ID_B, ID_A, 0, \{N_B\}K_B^n\}K_P \,||\, MAC\{C\}K_B^n$
⑤ $C=\{N'_A, 1\}K_A^n \,||\, MAC\{C\}N_A$
⑥ $C=\{N'_B, 1\}K_B^n \,||\, MAC\{C\}N_B$
⑦ $C=\{ID_A, ID_B, \{N''_A\}K_A^n\}K_P \,||\, MAC\{C\}K_A^n$
⑧ $C=\{ID_B, ID_A, \{N''_B\}K_B^n\}K_P \,||\, MAC\{C\}K_B^n$
⑨ $C=\{ID_B, K_{SS}, T_{SS}, N_A\}K_A^n \,||\, MAC\{C\}K_{SS}$
⑩ $C=\{ID_A, K_{SS}, T_{SS}, N_B\}K_B^n \,||\, MAC\{C\}K_{SS}$

(a) Two Nodes. Both without Keys

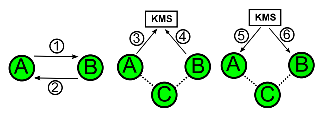


Session Key	Validity	Node IDs
Kss	Tss	A, X

Session Key	Validity	Node IDs
Kss	T^dss	A, B, X

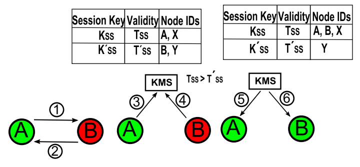
① ID_A ② Accept, ID_B
③ $C=\{ID_A, ID_B, 1, \{N_A\}K_A^n\}K_P \,||\, MAC\{C\}K_A^n$
④ $C=\{ID_B, ID_A, 0, \{N_B\}K_B^n\}K_P \,||\, MAC\{C\}K_B^n$
⑤ $C=\{N_A, 0\}K_A^n \,||\, MAC\{C\}N_A$
⑥ $C=\{N'_B, 1\}K_B^n \,||\, MAC\{C\}N_B$
⑦ $C=\{ID_B, ID_A, \{N''_B\}K_B^n\}K_P \,||\, MAC\{C\}K_B^n$
⑧ $C=\{ID_A, K_{SS}, T^d_{SS}, N_B\}K_B^n \,||\, MAC\{C\}K_{SS}$

(b) Two Nodes. One without Key



① ID_A ② Accept, ID_B
③ $C=\{ID_A, ID_B, 1, \{N_A\}K_A^n\}K_P \,||\, MAC\{C\}K_A^n$
④ $C=\{ID_B, ID_A, 1, \{N_B\}K_B^n\}K_P \,||\, MAC\{C\}K_B^n$
⑤ $C=\{N_A, 0\}K_A^n \,||\, MAC\{C\}N_A$
⑥ $C=\{N_B, 0\}K_B^n \,||\, MAC\{C\}N_B$

(c) Two Nodes with Same Key



Session Key	Validity	Node IDs
Kss	Tss	A, X
K´ss	T´ss	B, Y

Session Key	Validity	Node IDs
Kss	Tss	A, B, X
K´ss	T´ss	Y

① ID_A ② Accept, ID_B
③ $C=\{ID_A, ID_B, 1, \{N_A\}K_A^n\}K_P \,||\, MAC\{C\}K_A^n$
④ $C=\{ID_B, ID_A, 1, \{N_B\}K_B^n\}K_P \,||\, MAC\{C\}K_B^n$
⑤ $C=\{N_A, 0\}K_A^n \,||\, MAC\{C\}N_A$
⑥ $C=\{ID_A, K_{SS}, T_{SS}, N_B\}K_B^n \,||\, MAC\{C\}K_{SS}$

(d) Two Nodes with Different Keys

Figure 2. Different Scenarios in Establishing a Session Key

and $\mathcal{I}_*$ to get $P''_A = \{449a + 185, 271a + 183\} \in E''_A$. As evident, E_A, E'_A, E''_A are isomorphic because they have same $j-invariant$, $114a + 198$.

Meanwhile, after the acknowledgement, the other node B follows a similar process. It sends an encrypted message to KMS with A's identity in it. If KMS receives one of the messages from either A or B, it waits for a specific time interval for the other. If any of them gets lost, KMS will time out after that period and in turn nodes will time out without any response from the KMS. If messages are successfully delivered then both A (B) and KMS are sure about each others identity. Therefore KMS can now deliver the session key to A and B separately. It also embeds the nonce which was first given by A (and B) in the message 3 (and 4). It includes the id of the other node as well. This is because, in the meantime, A or B could have received concurrent authentication requests from other nodes and hence have sent corresponding requests to the KMS. Even though the key (K_{SS}) will be same for all those sessions, A must know to whom it can send the next message encrypted with the session key.

Along with the session key KMS also sets the validity period of the key. The time is determined according to the local clock of KMS. If nodes are operating with different clocks and knowing the difference with respect to the KMS clock, nodes should re-adjust the validity period. This is to ensure that the key expires at about the same time in KMS and in two nodes. KMS records the newly generated key and validity period in an internal master table. After the end of this period this session key will be deemed expired at the KMS. Node should delete an expired key. Otherwise, KMS will detect the discrepancy if it indicates in message 3 that it has a valid key and none found in the master table. Also as the session begins, both nodes record it since all further communication between them will be carried out under this session key without any intervention from the KMS.

4.1.2. Two nodes with only one having an established key. Another scenario (Figure 2(b)) is when one node has an established key and the other does not. In this scenario the goal of the protocol is to transfer the established key to the other node with the timer value adjusted accordingly. The node itself cannot transfer the key to the other node since it does not know its secret password and the other node also cannot use a session key directly given by a node which it does not trust. Therefore a similar procedure is initiated. However there is one difference. The difference is the node which has a valid key will indicate so in its message 3 to the KMS (Figure 2(b)). Upon receiving message 3, KMS learns that A already has a valid session key. Therefore it performs the lookup with the given node id A and retrieves the key K_{SS}. After checking the validity of the key (K_{SS}), KMS bypasses the verification procedures with A and sends it the response for no change in the session key (0 in message 5). However, verification for B proceeds as usual. After the verification KMS will simply transfer the same key to B. However, the validity period of the key needs to be recalculated for B. KMS subtracts from the current time the time K_{SS} was first issued and decrements the original validity period (T_{SS}) by this difference. B receives the key with the new validity period (T^{d}_{SS}).

4.1.3. Two nodes both having same established key. It is also possible that two non-communicating nodes are connected with the same node. In this scenario (Figure 2(c)) both nodes have the same key although they are not aware of that. It is required that nodes contact KMS in a similar manner as described in previous sections. KMS detects that both entities are currently in possession of the same valid key. Therefore it simply responds to them to continue with their existing keys.

4.1.4. Two nodes having different established keys. Finally we handle the case when two nodes both having valid but different session keys, want to communicate. Upon receiving request from both of them, KMS can check the validity period of two keys. It then selects the key which has longer validity period. In the Figure 2(d), node A has the key K_{SS} which was issued later than the key of node B ($K^{'}_{SS}$). Hence KMS will ask node B to replace its key to K_{SS}. This replacement is done without consulting other nodes within the cluster which were talking to B with $K^{'}_{SS}$. Therefore when any one of those nodes attempts to contact B, the scenario again reduces to Figure 2(d). In this way the newly generated key will iteratively propagate through the network replacing keys issued before, if required.

4.2. Key Management Unit Operations

The KMS follows the state machine as shown in the Figure 3. Initially it is in the idle state till it receives connection from any node. Once contacted by a node it spawns a new thread for handling the subsequent steps.

If KMS can successfully decode the incoming message using its private key and then using the pre-shared secret key of the node (as described in section 4.1.1), it moves to the new state where it waits for the connection from the second node. If this connection times out, KMS goes back to the idle state. Meanwhile it can receive connection from a node other than the one it is waiting for. In that case it spawns another thread while this thread still remains at state 1. The new thread handles the incoming messages in a similar fashion (not shown in the figure).

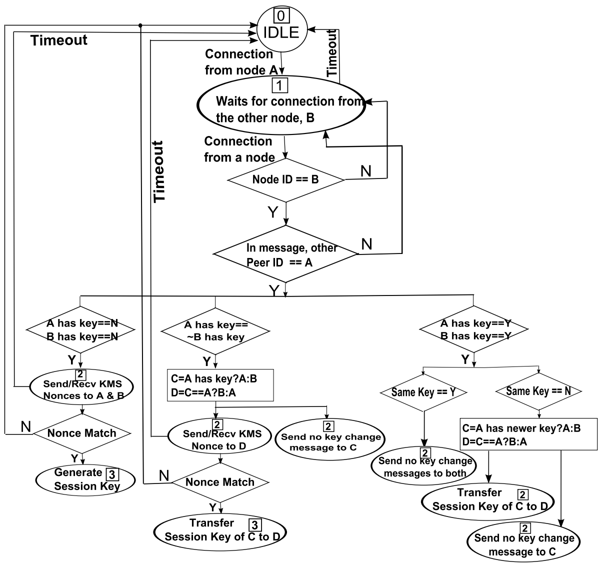

0
IDLE
Timeout
Connection from node A
Timeout
1
Waits for connection from the other node, B
Connection from a node
Timeout
Node ID == B
N
Y
In message, other Peer ID == A
N
Y
A has key==N B has key==N
Y
2
Send/Recv KMS Nonces to A & B
N
Nonce Match
Y
Generate Session Key
3
A has key== ~B has key
Y
C=A has key?A:B D=C==A?B:A
2
Send/Recv KMS Nonce to D
2
Send no key change message to C
N
Nonce Match
Y
Transfer Session Key of C to D
3
A has key==Y B has key==Y
Y
Same Key == Y
Same Key == N
C=A has newer key?A:B D=C==A?B:A
2
Send no key change messages to both
Transfer Session Key of C to D
2
2
Send no key change message to C

Figure 3. Key Negotiation Steps in KMS

However if the incoming request is from node B (the node for which this thread was waiting), then it moves to state 2. In state 2, KMS chooses two random isomorphisms and applies them on nonces it received in previous two messages and sends the result back to A and B. After that KMS waits for the node A and B to send back recomputed nonces.

If nodes are able to pass the nonce verification, then KMS moves to the final state where it generates the session key and sends it to both nodes. Also note that, KMS does not distinguish between two communicating nodes and interact with both of them in parallel. Therefore, in Figure 3, if the message from B arrives earlier than A, then also the protocol remains same.

There could be concurrent authentication requests from disjoint pairs of nodes or pairs of nodes with one node in common. Assume that node B_1 has started the authentication steps with node A (without key). However, before it is over, other nodes $B_2, B_3, \ldots, B_k$ also communicate with A. Some of them may have a valid key. KMS will spawn a separate thread for each of these connections. Let the thread (A, B_1) finishes first and delivers the session key to A and B_1. Later when the thread (A, B_2) finishes, it finds out, from the master table, that A has obtained a valid session key. Therefore it transfers that key to B_2 and sends a no key change message to A. In another scenario, assume both B_1 and B_2 have keys with B_2 having the newer key. Here,

A will first get the key of B_1. Later the thread (A, B_2) will deliver the key of B_2 to A and also pull it in the same cluster as B_2. It is necessary to lock the master table before consulting and updating it since separate threads may attempt to alter the status of the same node. An alternative approach to handle these types of concurrency issues is to use new technologies such as Transactional Memory (TM)[18], [19]. Using TM, the reading and updation of the master table should be wrapped within a single transaction.

Pre-shared secret key between a node A and KMS changes after completion of an authentication request. Now, if a node allows concurrent authentication requests (not to be confused with concurrent sessions which start after successful authentications), it must also be able to store those many pre-shared secret keys. This is because messages pertaining to previous requests will be encrypted with the corresponding pre-shared secret key. For example, in the previous case A must be able to store k number of pre-shared secret keys (the number of session keys needed to be stored in a node is always 1). Let A, B_1 request is the $n{-}th$ request from A to KMS. Now, messages encrypted with K_A^{n+1} (for A, B_2 request) may arrive earlier than messages encrypted with K_A^n. In that case A should attempt to decrypt it with both K_A^{n+1} and K_A^n and check the MAC. In KMS also same requirement applies. If KMS sees an authentication request from a node while it is carrying out one for the same node, it needs to save the current pre-shared secret key, and continue to decrypt the new request with the next pre-shared secret key of that node.

4.3. Duration of the Authentication Steps

For encryption, messages are partitioned into basic blocks. Each block is then encrypted with symmetric encryption key such as in message 5, 9 or with public key of the KMS as in messages 3, 7 (Figure 2(a)). For symmetric encryption with block ciphers[16], output of individual encryption block can be chained together starting with a proper initialization vector (IV) [20]. We assume the basic block size is M bits, for both symmetric and asymmetric encryption. Ciphered outputs are of same length. Symmetric keys (K_A^n, K_{SS}) and nonce (N_A, N_B) are of the length K bits. Because, we reuse nonce as a symmetric key in message 5 (and 6) in order to have separate keys for encryption and MAC computation. Node ids are I bits long and session validity period occupies T bits. Due to the symmetric nature of the protocol, both nodes exchange messages of same length to the KMS.

Length of message 3 (and 4) is

$$L_a = \left\lceil \frac{2I + 1 + \lceil \frac{K}{M} \rceil M}{M} \right\rceil M + M \tag{1}$$

The last addition is due to the MAC. We compute MAC in CBC manner[21]. Therefore the MAC size is also M bits.

Similarly length of message 5 (and 6) is

$$L_b = \left\lceil \frac{K + 1}{M} \right\rceil M + M \tag{2}$$

Length of message 7 (and 8) is

$$L_c = \left\lceil \frac{2I + \lceil \frac{K}{M} \rceil M}{M} \right\rceil M + M \tag{3}$$

Finally the messages with the session key, 9 and 10 are of length

$$L_d = \left\lceil \frac{2K + I + T}{M} \right\rceil M + M \tag{4}$$

Theorem 1. *$L_b \le L_c \le L_a \le L_d$, for $1 \le I \le T$, $1 < K \nmid I$ and $1 \le M \le 2K$.*

Proof: For positive x, a if $2x \ge a$, $x \le \lceil \frac{x}{a} \rceil a \le 2x$.

First three in-equalities are clear as L_b does not contain any I whereas L_c and L_a have two with an additional bit in the latter. For $L_d \ge L_a$, we only prove the case $T = I$ (then $T > I$ follows automatically).

Let $\alpha = \frac{(2I+1)}{M}$ and $\rho = \frac{K}{M}$.

Then $L_d = \lceil \alpha - \frac{1}{M} + 2\rho \rceil M + M$ and $L_a = \lceil \alpha + \lceil \rho \rceil \rceil M + M$.

Now, $\alpha + \lceil \rho \rceil - \lfloor \alpha + \lceil \rho \rceil \rfloor = \frac{r}{M} \ldots (i)$, where $2I + 1 \equiv r$ (mod M).

Therefore, for $2\rho > \lceil \rho \rceil$, $\alpha - \frac{1}{M} + 2\rho$ will either fall in the same interval with $\alpha + \lceil \rho \rceil$ or above. Consequently we have $\lceil \alpha - \frac{1}{M} + 2\rho \rceil \ge \lceil \alpha + \lceil \rho \rceil \rceil$, completing the proof.

In the case when $2\rho = \lceil \rho \rceil$ we must show that $r \ne 1$ in (i). If $2\rho = \lceil \rho \rceil$ then $M = 2K$. Since $K \nmid I$, $2I + 1 \not\equiv 1$ (mod $2K$). □

4.4. Commencing the Session with the Key

From message 9 (Figure 2(a)) node A learns the key K_{SS} and the peer against which the key is to be used (node B). However, there could be several scenarios if one or both of these messages get lost (message 9 or 10) as shown in Algorithm 1 and Algorithm 2.

Since node A initiated the connection, after sending message 2 node B would be expecting the next message from A to be encrypted. Meanwhile, it may happen that the message 9 gets lost but message 10 is through. In that case node A will timeout. This timeout period (denoted as $Timeout1$) is equal to sum of lengths of message 7 and response 9 divided by the data rate ($D_{A \leftrightarrow KMS}$). If A timeouts here then it resends its request and the situation becomes similar to Figure 2(b).

Next assume message 9 is delivered but message 10 is lost. In this case node A will be able to send a message to B encrypted with K_{SS} but B would not be able to decrypt it. Here node B will remain silent forcing node A to time out and restart the procedure. This timeout period in A is different from $Timeout1$ as this is equal to the time to send one encrypted test message to B and get the correct reply. This is denoted as $Timeout2$ in Algorithm 1. It can

Algorithm 1 Session Initiation at the Requester Node (A in Figure 2(a))

```
$t \leftarrow Timeout1 (= (L_c + L_d)/D_{A \leftrightarrow KMS})$;
$Success \leftarrow 0$;
$Send\ Message\ 7\ to\ KMS$;
while $t > 0$ do
  if $Valid\ Response\ from\ KMS\ in\ Message\ 9$; then
    $Success \leftarrow 1$;
    $break$;
  end if
  $t--$;
end while
if $Success == 1$ then
  $Success \leftarrow 0$;
  $t \leftarrow Timeout2$;
  $Send\ First\ Message\ to\ B\ encrypted\ with\ K_{SS}$;
  while $t > 0$ do
    if $Valid\ Response\ from\ B$ then
      $Success \leftarrow 1$;
      $break$;
    end if
    $t--$;
  end while
end if
if $Success == 0$ then
  $Resend\ Message\ 1$;
end if
```

Algorithm 2 Session Initiation at the Responder Node (B in Figure 2(a))

```
$t \leftarrow Timeout1 (= (L_c + L_d)/D_{B \leftrightarrow KMS})$;
$Success \leftarrow 0$;
$Received \leftarrow 0$;
$Send\ Message\ 8\ to\ KMS$;
while $t > 0$ do
  if $Message\ from\ A$ then
    $Received \leftarrow 1$;
  end if
  if $Valid\ Response\ from\ KMS\ in\ Message\ 10$ then
    $Success \leftarrow 1$;
    $break$;
  end if
  $t--$;
end while
if $Received == 1\ \&\&\ Success == 1$ then
  $Send\ Reply\ to\ A$;
else if $Success == 1$ then
  $t \leftarrow Timeout2/2$;
  while $t > 0$ do
    if $Message\ from\ A$ then
      $Received \leftarrow 1$;
      $Send\ Reply\ to\ A$;
      $break$;
    end if
    $t--$;
  end while
else
  $End\ Connection\ to\ A$;
end if
```

happen that A receives the key and send the test message which B receives while message 10 is still in flight. B can reply back immediately after obtaining the key. If B has obtained the key and yet to get the test message from A then it starts its own timeout counter which is half of the $Timeout2$. Because, A must wait till B gives correct reply to its test message whereas B waits only till it receives the test message from A. If both messages 9 and 10 are lost then the situation becomes identical with Figure 2(a).

4.5. Session Expiry

After a node receives the key and its validity period from the KMS, it starts tracking the session by decrementing the validity period at each clock tick. Once the validity period is over, the key is deleted and the node becomes keyless. Clock running in an individual node ideally should be in sync with the KMS[22]. This is required to ensure that a session expires at about the same time, both in a node and in the KMS. Once the KMS sent out the key it immediately starts tracking the validity period of the key against its clock. A node i will receive the key after a delay. After receiving the message, containing the key, i checks the packet timestamp T_i, key validity period T_{SS} and its current time C_i which leads by δ_i (can be negative) from the KMS. Then, the validity period of the key in i must be modified to $T_{SS_{new}} = T_{SS} - C_i + T_i + \delta_i$.

KMS can revoke operation of a malicious or compromised node. To do so it needs to simply delete the secret key corresponding to that node from its internal record. It is not needed to broadcast a system wide message regarding a revocation. Thereafter the revoked node will not be able to initiate or accept a session with any other node, because revoked node will not be able to encrypt message 3 correctly (Figure 2(a)).

5. Security Analysis

5.1. Man-in-the-Middle Attack

In this type of attack the adversary inserts itself in the communication route between two nodes. Following that, it attempts to impersonate either one or both nodes thereby controlling either half or full exchange without the knowledge of the other party. Even if we employ a different public-private key pair for every node, it is always possible to establish connection with one node and then impersonate that node while communicating with others. In SPIKE this attack is not possible because before two nodes start secure communication, it is required that they obtain keys from the central key management unit KMS. Now, one malicious node E can attempt to attack as follows: it supplies id of other node A to node B but to KMS it supplies its true identity, E. KMS would not be able to detect the fraud just from message 3 since E has supplied its true identity. However, after getting this message, KMS would wait for the second message from node B. That message would contain the id of node A instead of E. Therefore KMS will be able to detect the discrepancy after receiving the message 4 and wouldn't advance its state.

5.2. Replay Attack

For replay type of attacks an attacker stores a message which it captures during a session and attempts to replay the message as is in a later session. It can not modify the message as it does not have the decryption key. The standard procedure to protect against such attacks is to use a stateless cookie or what is known as a cryptographic nonce. A nonce is valid only during a short period while initiating the session. Apart from the protection against replay attack, nonce can also be used to test the liveliness of the opponent. We use nonce to protect against replay attacks and also to ensure that the KMS and the communicating nodes are computationally active. In message 3 (Figure 2(a)) when a node sends the nonce encrypted with its secret key, it can happen that it is a replay of some old message by a malicious node impersonating A. However it will be noticed by node A because it will get an unwarranted message 5 from KMS corresponding to an old nonce. Node A also needs to prove to KMS that it has the pre-shared secret key. It was not automatically proved by message 3 since that could be a replay to start with. To do so, node A decrypts the message 5, retrieve the nonce supplied by the KMS, determines the isomorphism that KMS has applied on the nonce (details are in section 4.1.1) it has previously given in 3, applies it again and sends result back (message 7). Since this isomorphism is chosen at random at KMS, if the size of F (table I) is p^n, then the probability that the KMS chooses the same isomorphism $\{u, r, s, t\}(u \neq 0)$ is $(p^{3n}(p^n - 1))^{-1}$. For the example in section 4.1.1, the probability is 2x10^{-22}. Therefore it is unlikely that an adversary will be able to replay even if it had captured both message 3 and 7.

5.3. Sybil Attack and Amplification Attack

In Sybil attack[23] one legitimate node attempts to play identity of several other nodes other than its true identity. Again this type of attack is not possible here since KMS will not recognize any node failing to provide matching id and corresponding secret password in message 3.

Amplification attack is particularly important in UDP style of protocols. Any adversary can modify the source address of a UDP packet to point to a victim. In return the victim node unintendedly receives a long message containing all the keys. This type of attack is not possible here since right at message 3 KMS can detect the fraud by observing the key mismatch with the supplied id of the victim.

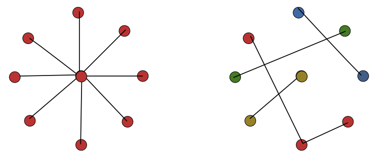
5.4. Denial of Service (DoS) Attack

Adversaries can attempt Denial of Service (DoS) attack on KMS by flooding it with messages of type 3 (Figure 2(a)). If adversary sends a random message, of course, the integrity check of the message will fail and gets discarded without further action. However, adversary may attempt to replay previously captured message 3 of valid format. The protection of replay attack requires checking of re-computed nonce in message 7. This would require KMS to perform one futile transmission (message 5). Although this could be permissible if the purpose is solely guaranteeing the freshness of the message, this wastage is not acceptable in the case of DoS type of attack since the attacker is flooding. There are two simple ways to handle such situations. If KMS receives any message 3 (with 0 in the third field) from a node which already has a valid key according to its master table, it may ignore the request. Alternatively, once a node contacts KMS for authentication, its pre-shared secret key is overwritten (both in the node and KMS) for the next authentication request. The attacker which is replaying an old message would not be able to encrypt the message with the new pre-shared key and fail integrity check.

6. Simulation and Results

6.1. Simulation Setup

We have implemented a client and a server program which are installed in each sensor node. Client process contacts to other nodes using a port and the server process listens for client requests using a different port. KMS process runs in a separate workstation with high-storage capacity. Using a publicly known port KMS listens for authentication requests from nodes. We have implemented the client, server and KMS in C using standard socket apis. We have used a data rate of 250 Kbits/s for both node to node and node to KMS link. We have used I = 40 bits, K = 128 bits, T = 40 bits and M = 128 bits as described in section 4.3. Therefore, the exchange of Figure 2(a) takes about $7ms$, considering overhead bits.



(a) Star Topology (b) Maximum Session Key Topology

Figure 4. Representative Network Topologies for Minimum and Maximum Number of Session Keys

In our experiment, we have examined two topologies (Figure 4) and one topology transformation. Arrival time of each node is simulated deterministically and spaced 1 millisecond apart. Session key is valid for 1 simulated second.

a) Star Topology- In this case, each node communicates with a central node. Obviously, in this case KMS needs to store only one session key as the number of connected component is 1. Let $n \geq 2$ be the number of nodes. Then, of all the graphs which can be formed out of n vertices ($2^{^nC_2}$), the number of graphs G_1 where the entire graph is connected, has the lower bound $G_1 \geq \prod_{i=1}^{i=n-1}(2^i - 1)$. This can be proved easily using induction on n. Therefore, the probability that KMS needs to store only 1 session key, $P_1(n) \geq \frac{\sum_{r=2}^{r=n}({}^nC_r \prod_{i=1}^{i=r-1}(2^i-1))}{2^{^nC_2}}$. Also, $\lim_{n\to\infty} P_1(n) \geq 0.289$. This nonzero probability comes from the last term, $\prod_{i=1}^{i=n-1}(1 - \frac{1}{2^i})$ which is the euler function related with the euler's pentagonal number theorem, evaluated at $\frac{1}{2}$.

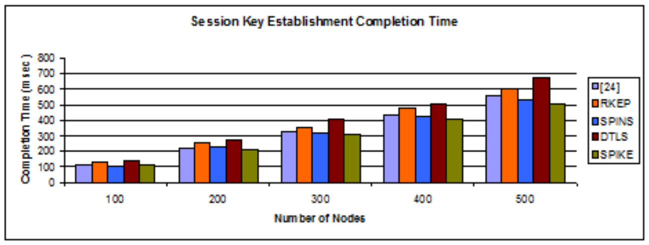
b) $\lfloor \frac{n}{2} \rfloor$-Session Key Topology- Here, each node communicates with only an another node. Thus the network graph is a set of connected components of cardinality 2 (and one component of cardinality 3 or a silent node if n is odd). This gives the upper bound on the key storage requirement in KMS. The probability of this case is $\frac{(n)!}{(\frac{n}{2})!2^{\frac{n^2}{2}}}$, if n is even and $\frac{(n)!(2n+1)}{3(\frac{n-1}{2})!2^{\frac{n^2-1}{2}}}$, if n is odd.

c) Next we transform topology from Figure 4(a) to Figure 4(b). After expiry of the session (in all the nodes after 1 second), a node i selects a different node j and after that it neither contacts nor accepts connection from any other node. The node j also does the same.

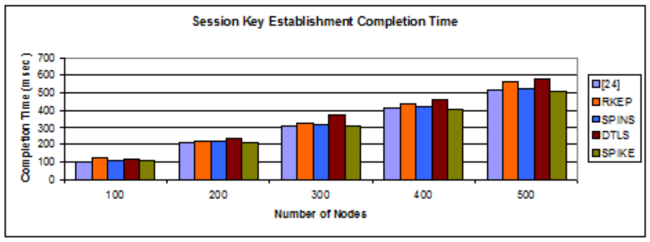
6.2. Results

We compare SPIKE with the following protocols: **1**) [24] **2**) RKEP[25] **3**) SPINS [26] **4**) DTLS [13]. In SPIKE, maximum number of session key stored in a node is 1, at any given instance, in all cases. For all other protocols this is $n - 1$ which occurs for topologies such as Figure 4(a). Further we compare Session Key Establishment Completion Time-This is defined as the difference of time when the last arrived node gets the session key to the arrival time of the first two communicating nodes. For c) we calculate the end time as the time when transformation from a) to b) gets completed. The results are shown in Figure 5.

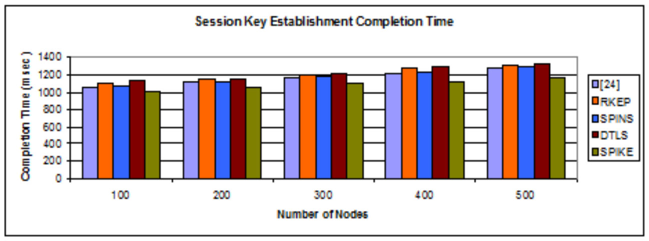
In Figure 5(a), SPIKE speeds-up (on average by 10% over [24]) the key negotiation process because of its support of simultaneous authentication requests from the same node. In this case requests from nodes will arrive to the central node of the star topology in an interval when it is still completing the authentication steps with other nodes. In Figure 5(b), SPIKE and [24] perform similar since pairs are disjoint. DTLS[13] takes longer completion time because of long certificate (1671 bytes) exchange.



(a) Completion Time for Star Topology



(b) Completion Time for Maximum Session Key Topology



(c) Completion Time for Topology Transformation

Figure 5. Session Key Establishment Completion Time for Different Configurations

7. Conclusion

In this paper we have proposed Symmetric Parallel Immalleable Key Establishment (SPIKE), a novel session key distribution technique for Wireless Sensor Networks. It carries out the authentication in a symmetric and parallel manner. This speeds-up the authentication phase. SPIKE is minimal in terms of key storage. It requires only one session key to be stored in any node irrespective of the network size and the number of concurrent sessions the node is participating into. The proposal incorporates a central Key Management Server (KMS) which distributes session keys in a manner to ensure that nodes belonging to a connected component of the original network, share the same session key. Thereafter KMS generates or migrates session keys as old keys expire, new node initiates session or existing nodes start communicating with different sets of nodes, transforming the network topology. The protocol ensures protection against several security threats, viz. man-in-the-middle attack, record and replay kind of attacks, sybil attack, amplification attack and also denial of service attack.

Acknowledgments

Part of this work was done while the author was in Centre for Wireless Communications (CWC), University of Oulu, Finland and financially supported by University of Oulu, Finland, from 19-08-2011 to 31-08-2012. The author is solely responsible for the contents of the paper.

References

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

[2] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," *Communications of the ACM*, vol. 47, no. 6, pp. 53–57, 2004.

[3] J. Nogueira, H. Wong, A. Loureiro, C. Bekara, M. Laurent-Makanvicius, A. da Silva, S. de Oliverira, and F. Texiseira, "Wireless sensor network security," *Wireless and Mobile Network Security : Security Basics, Security in On-the-shelf and Emerging Technologies*, pp. 565–611, 2010.

[4] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, "Perfectly-secure key distribution for dynamic conferences," in *Advances in cryptology-CRYPTO 1992*. Springer, 1993, pp. 471–486.

[5] J. Staddon, S. Miner, M. Franklin, D. Balfanz, M. Malkin, and D. Dean, "Self-healing key distribution with revocation," in *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*. IEEE, 2002, pp. 241–257.

[6] Y. Zhou and Y. Fang, "A scalable key agreement scheme for large scale networks," in *Networking, Sensing and Control, 2006. ICNSC'06. Proceedings of the 2006 IEEE International Conference on*. IEEE, 2006, pp. 631–636.

[7] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," in *Advances in Cryptology-EUROCRYPT'94*. Springer, 1995, pp. 275–286.

[8] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Advances in Cryptology-CRYPTO 2001*. Springer, 2001, pp. 213–229.

[9] D. Freeman, "Constructing pairing-friendly elliptic curves with embedding degree 10," *Algorithmic number theory*, pp. 452–465, 2006.

[10] F. Hess, N. P. Smart, and F. Vercauteren, "The eta pairing revisited," *Information Theory, IEEE Transactions on*, vol. 52, no. 10, pp. 4595–4602, 2006.

[11] E. Bresson, O. Chevassut, and D. Pointcheval, "Dynamic group diffie-hellman key exchange under standard assumptions," in *Advances in Cryptology-EUROCRYPT 2002*. Springer, 2002, pp. 321–336.

[12] M. Abdalla, E. Bresson, O. Chevassut, and D. Pointcheval, "Password-based group key exchange in a constant number of rounds," *Public Key Cryptography-PKC 2006*, pp. 427–442, 2006.

[13] N. Modadugu and E. Rescorla, "The design and implementation of datagram tls," in *Proceedings of ISOC NDSS*, 2004.

[14] S. Al-Riyami and K. Paterson, "Certificateless public key cryptography," *Advances in Cryptology-ASIACRYPT 2003*, pp. 452–473, 2003.

[15] C. Kuo, M. Luk, R. Negi, and A. Perrig, "Message-in-a-bottle: User-friendly and secure key deployment for sensor nodes," in *Proceedings of the 5th international conference on Embedded networked sensor systems*. ACM, 2007, pp. 233–246.

[16] J. Daemen and V. Rijmen, *The design of Rijndael: AES–the advanced encryption standard*. Springer Verlag, 2002.

[17] J. H. Silverman, *The Arithmetic of Elliptic Curves*. Springer, 2008, vol. 2nd Edition.

[18] S. Sanyal and S. Roy, "QuickTM: A Hardware Solution to a High Performance Unbounded Transactional Memory," in *Proc. of the 12th IEEE International Conference on High Performance Computing and Communications (HPCC-10)*, 2010, pp. 62–71.

[19] S. Sanyal, S. Roy, A. Cristal, O. S. Unsal, and M. Valero, "Dynamically Filtering Thread-Local Variables in Lazy-Lazy Hardware Transactional Memory," in *Proc. of the 11th IEEE International Conference on High Performance Computing and Communications (HPCC-09)*, 2009, pp. 171–179.

[20] M. Liskov, R. Rivest, and D. Wagner, "Tweakable block ciphers," *Advances in Cryptology-CRYPTO 2002*, pp. 31–46, 2002.

[21] D. Whiting, R. Housley, and N. Ferguson, "Rfc 3610, counter with cbc-mac (ccm)," *Internet Engineering Task Force*, 2003.

[22] T. Kunz and E. McKnight-MacNeil, "Implementing clock synchronization in wsn: Cs-mns vs. ftsp," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*. IEEE, 2011, pp. 157–164.

[23] H. Yu, M. Kaminsky, P. Gibbons, and A. Flaxman, "Sybilguard: defending against sybil attacks via social networks," in *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4. ACM, 2006, pp. 267–278.

[24] N. T. Canh, P. Truc, T. H. Hai, Y.-K. Lee, S. Lee *et al.*, "Enhanced group-based key management scheme for wireless sensor networks using deployment knowledge," in *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*. IEEE, 2009, pp. 1–5.

[25] A. Wacker, M. Knoll, T. Heiber, and K. Rothermel, "A new approach for establishing pairwise keys for securing wireless sensor networks," in *Proceedings of the 3rd international conference on Embedded networked sensor systems*. ACM, 2005, pp. 27–38.

[26] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. Culler, "Spins: Security protocols for sensor networks," *Wireless networks*, vol. 8, no. 5, pp. 521–534, 2002.