One-Pass Authenticated Key Establishment Protocol on Bilinear Pairings for Wireless Sensor Networks

Manoj Ranjan Mishra1, Jayaprakash Kar2 and Banshidhar Majhi3

1 School of Computer Application
KIIT University, Bhubaneswar, India
2 Department of Information Systems
Faculty of Computing & Information Technology
King Abdulaziz University, Kingdom of Saudi Arabia
3 Department of Computer Science & Engineering
National Institute of Technology, Rourkela, India

Abstract. The article proposes one-pass authenticated key establishment protocol in random oracles for Wireless Sensor Networks. Security of the protocol relies on Computational Diffie-Hellman Problem on Bilinear Pairings. In one-pass key establishment protocol, the initiator computes a session key and a related message. The key token is to be sent to the intended receiver using receiver’s public key and sender secret key. From the received key token the receiver compute the session key, which is the same as the one computed by the sender, using sender public key and receiver’s secret key. Because of low communication overhead, the scheme is better suited for Wireless Sensor Networks (WSNs) than the traditional key establishment protocol to establish the session key between two adjacent nodes.

Keywords: Key establishment, bilinear pairings, Session key, Authentication, BDHP

1 Introduction

Pairing-Based Cryptography is an emerging area of cryptography that revolves around a particular function with interesting properties. Pairings, such as the Weil pairing, were first used in the context of cryptanalysis [1] to reduce the ECDLP into a discrete logarithm problem in the finite field. The first use of pairings in cryptography is the work of Sakai et al. [5] and Joux [21]. Both papers proposed pairings as the base for building complete cryptosystems. Since then many protocols have been proposed that use pairings as the underlying crypto primitives.

WSN systems are usually deployed in hostile environments where they encountered a wide variety of malicious attacks. Information that is the cooked data collected within the sensor network is valuable and should be kept confidential. In order to protect this transmitted information or messages between any two adjacent sensor nodes key establishment protocol and a mutual authentication are required for wireless sensor networks. Due to nature restrictions like low power, less storage space, low computation ability and short communication range of sensor nodes, most conventional protocols establish authenticated multiple keys between any two adjacent sensor nodes by adopting a key pre-distribution approach. However, these techniques have vulnerability. With rapid growth of cryptographic techniques, recent results show that Elliptic Curve Cryptography (ECC) is suitable for resource-limited WSNs. Cryptosystem based on Elliptic Curve Cryptography are especially interesting for sensor networks since they are more efficient in resource utilization than any other public key techniques [6] [20]. Cryptographic Protocol based on bilinear pairing is very interesting and emerging to modern cryptographic research community. Since the security of the protocol more stronger than others. Pairing based cryptography has also allowed many long-standing open problems to be solved in a well-designed way. The computational capability of sensor nodes are limited, so traditional public-key cryptography in which the computation of modular exponentiation is required, cannot be implemented on WSNs.

* Corresponding author: Jayaprakash Kar, e-mail: jayaprakashkar@yahoo.com
2 Preliminaries

2.1 Bilinear Pairings

A bilinear pairing is a map between two groups. The two kind of bilinear pairings are Weil and Tate pairings on elliptic curve. A bilinear pairing is defined as:
Let \( G_1 \) and \( G_2 \) be two cyclic groups of prime order \( q \) with identity element \( O \). Let \( G_T \) be a multiplicative group of order \( q \). Let \( G_1 \) and \( G_2 \) are additive group and \( G_2 \) is a multiplicative group. Let \( \hat{e} \) be a computable and non-degenerated bilinear map

\[
\hat{e} : G_1 \times G_2 \rightarrow G_T
\]

which satisfies the following properties:

- **Bilinear**: \( \hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab} \), where \( P, Q \in G_1 \) and \( a, b \in \mathbb{Z}_q^* \) and for \( P, Q, R \in G_1, \hat{e}(P + Q, R) = \hat{e}(P, R)\hat{e}(Q, R) \).
- **Non-degenerate**: If \( P \) is a generator of \( G_1 \), then \( \hat{e}(P, P) \) is generator of \( G_2 \). There exists \( P, Q \in G_1 \) such that \( \hat{e}(P, Q) \neq 1_{G_2} \)
- **Computability**: There exists an efficient algorithm to compute \( \hat{e}(P, Q) \) for all \( P, Q \in G_1 \).

We call such a bilinear map \( \hat{e} \) is an admissible bilinear pairing.

2.2 Pairing types

The properties of pairing depends on the selected groups \( G_1, G_2 \) and \( G_T \). Generally pairings is of three types depending three basic groups.

- **Type-1**: Here \( G_1 = G_2 \).
- **Type-2**: Pairing where \( G_1 \neq G_2 \), \( \exists \) an efficiently computable homomorphism \( \phi : G_2 \rightarrow G_1 \).
- **Type-3**: Pairing where \( G_1 \neq G_2 \), \( \exists \) an efficiently computable homomorphism \( \phi : G_2 \rightarrow G_1 \)

3 Security Model

In 2006 LaMacchia, Lauter & Mityagin introduced a stronger security model for AKE protocols. The model is a weak corruption model, where the adversary is allowed to reveal only the static private key of an entity through Long-Term Key Reveal. The protocol is modeled as a collection of programs running at different entities \( e_1, e_2 \ldots e_n \) with identities \( ID_1, ID_2 \ldots ID_n \). Each entity is allowed to have different instances of running the protocol, modeling the real time scenario of having multiple sessions open with different partners. The adversary \( A \) performs polynomial bounded number of queries to the oracles provided to \( A \) by \( B \). Let us assume that, \( A \) can activate each user not more than \( m \) number of times. Challenger \( B \) runs \( A \) as subroutine and selects \( ID_1, ID_2 \ldots ID_n \) with identities \( s \) is oracle \( \{\prod_{i,j}^s\} \), where \( \prod_{i,j}^s \) behaves as node \( i \) carrying out the protocol with node \( j \) for the \( s \) times. If \( B \) selects the \( s' \)th session activated at \( ID_i \) with \( ID_j \) as the test session, then \( B \) guess will be correct. \( B \) sets the public key of \( KGC \) as \( aP \). The communications network is controlled by a PPT adversary \( A \), which schedules and mediates all sessions between the entities. It is also given the power of initiating fictitious entities by obtaining private keys from the \( KGC \) for arbitrary identities. All the entities (including the honest ones) are activated by \( A \). Upon activation, the entities perform some computations as per the received communication, update their internal state and complete the session. The session identifier \( sid \) is assumed to be the concatenation of the messages exchanged between two entities along with their identities. Two sessions are said to be matching sessions if their \( sid \)'s are identical. Let \( msk \) be the master secret key used by the \( KGC \) to issue private keys to the users. \( A \) selects identities of the honest entities and let them obtain private keys for the identities from the \( KGC \). It is allowed to obtain private keys for any arbitrary identity of its choice. \( A \) run the following queries:
- **Send**: In this query, a unique message \( m \) is to be sent to \( ID_u \) that comes from \( ID_v \). The result will submit to \( A \). If \( m \) is null message (i.e., \( m = \phi \)), the queries active the entity \( ID_u \) as **initiator** otherwise its role is considered as **responder**. Thus the session \( sid \) is defined as the tuples \( <ID_u, ID_v, m, role> \), where \( role \in \{\text{initiator}, \text{responder}\} \).

- **Reveal\((sid)\)**: The session key \( sid \) generated by the challenger \( B \) in the session \( sid \) returns to \( A \).

- **Long-term Key Reveal\((ID_u)\)**: In this query, the challenger \( B \) takes \( ID_u \) and returns the **Long-term Private key**.

- **Ephemeral Key Reveal\((sid)\)**: The challenger \( B \) returns the ephemeral private key used in the incomplete session \( sid \) to \( A \). This may includes all sensitive session state information used by the entity \( ID_u \) in \( sid \). It is assumed that, when the session is completed, the session state information is erased from the memory of the entity.

- **Extract\((ID_u)\)**: The challenger \( B \) takes \( ID_u \) returns the private key to \( A \).

The adversary \( A \) issues a **Test** query to a clean session \( sid \) as follows:

1. **Test\((sid)\)**: Here the adversary \( A \) begins by running the oracle \( \prod_{i,v}^{\phi} \) and asks a single new query **Test** on the session \( sid \). To answer the query the oracle flips a fair coin \( b \in \{0,1\} \) and returns the session key in the session \( sid \) to \( A \) if \( b = 0 \) or a random value if \( b = 1 \). Finally it terminates the game if he guesses the correct value \( b' \) which is different from the random value. The adversary \( A \) wins the game if \( b' = b \) and selected test session is clean.

2. **Clean\((sid)\)**: Let \( sid \) be a session completed at the entity \( ID_u \) and let \( sid' \) be the matching session at \( ID_v \). The long-term secret keys \( ID_u \) and \( ID_v \) are denoted by \( SK_u \) and \( SK_v \) respectively. Let \( E_{SK_u} \) and \( E_{SK_v} \) be the ephemeral secret keys used in \( sid \) and \( sid' \) respectively. A session \( sid' \) is said not to be clean if one of the following conditions holds:
   - \( ID_u \) or \( ID_v \) is an adversary controlled party (whose private keys are obtained by **Extract** queries).
   - \( A \) reveals the master secret key \( msk \) of the KGC.
   - \( A \) reveals the session key held in \( sid \) or \( sid' \) (if there exists such a \( sid' \)).
   - A matching session \( sid' \) exists and \( A \) reveals \( SK_u \) and \( E_{SK_u} \) or \( SK_v \) and \( E_{SK_v} \).
   - A matching session \( sid \) does not exist and \( A \) reveals \( SK_u \) and \( E_{SK_u} \) or \( SK_v \).

In all other cases the session is said to be **clean**.

**Definition 1.** The advantage of the adversary \( A \) in the experiment of the protocol \( \pi \) is defined as:

\[
\text{Adv}_\pi(A) = \frac{\text{Pr}[\text{Suc}]}{\frac{1}{2}}
\]

An one-pass authenticated protocol \( \pi \) is said to be secure under the defined model, if there does not exist PPT adversary that has no negligible advantage to distinguish the real session key from a random bit string. i.e

\[
\text{Adv}_\pi(A) = \epsilon(k) \leq k^{-c} \quad \forall \quad k > k_c, c > 0
\]

### 4 Proposed One-Pass Protocol

Consider two arbitrary nodes \( i \) and \( j \) would like to share session keys to establish secure communication. Node \( i \) has computed long-term private and public key as \( S_i = s \cdot Q_i \). Similarly node \( j \) has computed his long-term private and public key as \( S_j = \lambda_j \cdot Q_j \). The protocol comprises the following three polynomial time solvable (PPT) algorithms.

- **Setup** : Given security parameters \( k \), the **KGC** chooses groups \( G \) and \( G_T \) of prime order \( q \). A generator \( P \) of \( G \), a bilinear map \( e : G \times G \rightarrow G_T \) and collision resistant hash function \( H_1 : \{0,1\}^n \rightarrow G \), \( H_2 : G \times \{0,1\}^n \rightarrow \mathbb{F}_q^* \), \( H_3 : G \rightarrow \mathbb{F}_q^* \). A key derivation function \( F : G_T \rightarrow \{0,1\}^k \). Where \( |k| \) is the length of the key strings. It chooses a master-key \( s \in \mathbb{F}_q^* \) and computes \( P_{pub} = sp \). The KGC publishes the system public parameters \( \text{params} = \langle G, G_T, e, q, p, P_{pub}, H_1, H_2, H_3 \rangle \).

- **Extract** : **KGC** takes the input the identities \( ID_i \) and \( ID_j \) of node \( i \) and \( j \) respectively and runs the algorithm and Computes the private keys as \( S_i = s \cdot Q_i \) and \( S_j = s \cdot Q_j \) of the node \( i \) and \( j \) respectively. (Where Public key of node \( i \) and \( j \) are \( Q_i \) and \( Q_j \) respectively. \( Q_i = H_1(ID_i) \) and \( Q_j = H_1(ID_j) \).
Protocol: The protocol involves by considering two arbitrary nodes $i$ and $j$. Follows the following steps

- Node $i$ chooses randomly $\lambda \in \mathbb{Z}_q^*$ and computes $U = \lambda \cdot Q_i$.
- Computes $\mathcal{H}_2(U, ID_i || ID_j) = \Gamma$ and $h = \mathcal{H}_3(\Gamma)$.
- Computes $SK_{ij} = \hat{e}((\lambda + h)S_i, Q_j)$
- Computes $V = \lambda \cdot \Gamma$.
- Sends the pair of elements $(U, V)$ to Node $j$.
- After received, Node $j$ computes $\Gamma = \mathcal{H}_2(U, ID_i || ID_j)$, $h = \mathcal{H}_3(\Gamma)$ and $W = s \cdot \Gamma$.
- Verify $\hat{e}(U, W) = \hat{e}(S_i, V)$ and computes $SK_{ji} = \hat{e}(U + hQ_i, S_j)$

4.1 Proof of Correctness

\[ \hat{e}((\lambda + h)S_i, Q_j) = \hat{e}(S_i, Q_j)^{(\lambda + h)} = \hat{e}(Q_i, Q_j)^{(\lambda + h)} \]
\[ \hat{e}(U + hQ_i, S_j) = \hat{e}(\lambda Q_i + hQ_i, S_j) = \hat{e}(Q_i, S_j)^{(\lambda + h)} = \hat{e}(Q_i, Q_j)^{(\lambda + h)} \]

Consider the verification equation $\hat{e}(U, W) = \hat{e}(S_i, V)$ and check the equality as

\[ \hat{e}(U, W) = \hat{e}(\lambda Q_i, sQ) = \hat{e}(Q_i, Q)^{\lambda s} \]
\[ \hat{e}(S_i, V) = \hat{e}(Q_i, Q)^{\lambda s} \]

1. Setup: Given security parameters $k$, the KGC chooses groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order $q$.
   A generator $P$ of $G$, a bilinear map $\hat{e} : G \times G \rightarrow G_T$ and collision resistant hash function $\mathcal{H}_1 : \{0, 1\}^* \rightarrow G$, $\mathcal{H}_2 : G \times \{0, 1\}^* \rightarrow \mathbb{F}_q^*$, $\mathcal{H}_3 : G \rightarrow \mathbb{F}_q^*$.
   It chooses a master-key $s \in \mathbb{F}_q^*$ and computes $P_{pub} = sP$.
   The KGC publishes the system public parameters $params = \langle G, G_T, \hat{e}, q, p, P_{pub}, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3 \rangle$.

2. Extract: KGC takes the input the identities $ID_i$ and $ID_j$ of node $i$ and $j$ respectively and runs the algorithm and Computes the private keys as $S_i = s \cdot Q_i$ and $S_j = s \cdot Q_j$ of the node $i$ and $j$ respectively. $Q_i = \mathcal{H}_1(ID_i)$ and $Q_j = \mathcal{H}_1(ID_j)$.
   (Where Public key of node $i$ and $j$ are $Q_i$ and $Q_j$ respectively.

3. Protocol: The protocol involves by considering two arbitrary nodes $i$ and $j$

<table>
<thead>
<tr>
<th>Node $i$</th>
<th>Node $j$</th>
</tr>
</thead>
</table>
| Chooses randomly $\lambda \in \mathbb{Z}_q^*$ and computes $U = \lambda \cdot Q_i$.
| Computes $\mathcal{H}_2(U, ID_i || ID_j) = \Gamma$ and $h = \mathcal{H}_3(\Gamma)$.
| $SK_{ij} = \hat{e}((\lambda + h)S_i, Q_j)$
| Computes $V = \lambda \cdot \Gamma$.
| $(U, V)$ |
| Computes $\Gamma = \mathcal{H}_2(U, ID_i || ID_j)$, $h = \mathcal{H}_3(Q)$
| Computes $W = s \cdot \Gamma$
| and Verify $\hat{e}(U, W) = \hat{e}(S_i, V)$.
| Computes $SK_{ji} = \hat{e}(U + hQ_i, S_j)$ |

Fig. 1. Proposed One-Pass Protocol

5 Security Analysis

This section describes the security analysis and prove that the scheme is provably secure win random oracle model.

Theorem 1 In the random oracle model, the proposed one-pass protocol is secure, if the adversary $A$ can computes and distinguish the real session key from a given random oracle during the game with a non-negligible advantage and run Keygen queries, Send, Reveal, Extract, Ephemeral Key Reveal and Test queries; then there exists a challenger $B$ that can solve an instances of Bilinear Diffie-Hellman problem with a non-negligible advantage.
Proof:

- Setup: The Challenger $B$ receives a random instance $(P, aP, bP, cP)$ of the Bilinear Diffie-Hellman problem. His goal is to compute $\hat{e}(P, P)^{abc}$. $B$ will run $A$ as a subroutine and act as $A$’s challenger in the game. $B$ needs to maintain lists $L_1, L_2$ and $L_3$ that are initially empty and are used to keep track of answers to queries asked by $A$ to oracles $H_1, H_2$ and $H_3$ respectively. Consider there are $n$ number of users with identities $ID_1, ID_2, \ldots ID_n$ are participating in the protocol. Let us assume that, $A$ can activate each user not more than $m$ number of times. $B$ selects $ID_i, ID_j \in \{ID_1, ID_2, \ldots ID_n\}$ and $s \in \{1, 2, \ldots m\}$ i.e oracle $\{\prod_{i,j}\}$, where $\prod_{i,j}$ behaves as node $i$ carrying out the protocol with node $j$ for the $s$ times. If $B$ selects the $s^{th}$ session activated at $ID_j$ with $ID_i$ as the test session, then $B$ guess will be correct. $B$ sets the public key of $KGC$ as $aP$. $B$ runs the queries of $A$ and answers the followings:

Oracle Simulation:

1. $H_1$-Oracle: At the beginning of the game, For $H_1$-queries the list $L_1$ is empty. $B$ gives $A$ the system parameters with $P_{pub} = cP$ ($c$ is unknown to $B$ and plays the role of the $KGC$’s master-key). Then $B$ chooses two distinct random numbers $i, j \in \{1, \ldots q_{h}\}$. $A$ asks a polynomial bounded number of $H_1$ requests on identities of his choice. At the $i^{th}$ $H_1$ request, $B$ answers by $H_1(ID_i) = aP$. At the $j^{th}$, he answers by $H_1(ID_j) = bP$. Since $aP$ and $bP$ belong to a random instance of the BDH problem, $A$’s view will not be modified by these changes. Hence, the private keys $S_{ID_i}$ and $S_{ID_j}$ (which are not computable by $B$) are respectively $acP$ and $bcP$. Thus the solution $\hat{e}(P, P)^{abc}$ of the BDH problem is given by $\hat{e}(Q_{ID_i}, S_{ID_j}) = \hat{e}(S_{ID_i}, Q_{ID_j})$. For requests $H_1(ID_k)$ with $k \neq i, j$, $B$ chooses $b_k \leftarrow_R \mathbb{F}_q^*$, adds the pair $(ID_k, b_k)$ in list $L_1$ and answers $H_1(ID_k) = b_kP$. Further on input $ID_1 \in \{0, 1\}$, $B$ first checks the $L_1$-list $<ID_i, X_i, q_i, x_i>$, then selects new random $\gamma_i \leftarrow_R \mathbb{F}_q^*$, sets $X_i = b_i \cdot P, q_i = \gamma_i, \gamma_i$, add this tuple $<ID_i, X_i, q_i, x_i, \gamma_i>$ to the $L_0$-list and returns $q_i$. Otherwise, $B$ selects a new random $\gamma_i \leftarrow_R \mathbb{F}_q^*$, sets $X_i = x_i \cdot P, q_i = \gamma_i$, add this tuple $<ID_i, X_i, q_i, x_i, \gamma_i>$ to the $L_1$-list and returns $q_i$. $B$ starts with empty list $L_1$. It takes the input $ID_u$ and checks if there is entry for it in $L_1$. If exists, then returns the stored $Q_u$ to $A$. Otherwise, it chooses a value $\psi_u \in \mathbb{F}_q^*$ randomly and computes $\psi_u \cdot P$. It returns it to $A$. The entry $<ID_u, \psi_u, Q_u>$ includes in the list $L_1$. If $u = j$, the value $bP$ from its input is returned and entry $<ID_u, 1, bP>$ is added to the list.

2. $H_2$-Oracle: On input $(U, ID_i||ID_j)$, $B$ first checks the $L_2$-List, whether the tuple $<ID_i, ID_j, U, Q>$ in the $L_2$-List, $B$ returns $\Gamma$, otherwise $B$ chooses a new random $\Gamma \in \mathbb{G}_5$, adds $\Gamma$ to the $L_2$-list and return $\Gamma$.

3. $H_3$-Oracle: On input the element $\Gamma$, $B$ first checks the $L_3$-List, whether the tuple $<\Gamma, h>$ in the $L_3$-List, $B$ returns $h$, otherwise $B$ chooses a new random $h \leftarrow_R \mathbb{F}_q^*$, includes $h$ to the $L_3$-list and return $h$.

4. Keygen-Oracle: When $A$ makes a Keygen query with $ID_i$, as the input, $B$ checks the $L_1$-List to verify whether or not there is an entry for $ID_i$. If the $L_1$-List does not contain an entry for $ID_i$, return $\bot$. Otherwise, if $ID_i = ID_B$, $B$ recovers the tuple $<ID_i, X_i, q_i, x_i>$ from the $L_0$-List and returns $<X_i, q_i, x_i, \lambda \gamma_i \cdot (cP)>$, if $ID_i \neq \{ID_1\}_{i=1..n}$, $B$ recovers the tuple $<ID_i, X_i, q_i, x_i>$ from the $L_1$-List and returns $<X_i, q_i, S_{ID_i}, d_i>$, where $S_{ID_i} = x_i(aP) = aX_i$ and $d_i \leftarrow_R \mathbb{F}_q^*$ is randomly selected.

5. Send-Oracle: $A$ makes a Send$(ID_u, ID_v, m)$ query base on the observation to the session sid:
   (a) The node $u$ with $ID_u$ originates $m = \mu$. If $u = 1, v = j$ with $s^{th}$ session between them, $B$ randomly selects $\gamma \in \mathbb{F}_q^*$ and returns $X_u = \lambda \gamma \cdot (cP)$. If $u = i$, $B$ chooses $\eta_i, h_{j,t} \in \mathbb{F}_q^*$ computes $X_{j,t} = \eta_i P - h_{j,t}Q_j$ and returns $h_{j,t}$ by running $H_2$ corresponds to the query $(X_{j,t}, ID_j||ID_u)$, $B$ stores $\eta_i$ in the list $L_2$ for session sid. Similarly for other session between $ID_i$ and $ID_j$, $B$ chooses (b) The role of $ID_i$ is responder ($m \neq \mu$). $B$ accepts the session and marks it as completed.

6. Reveal$(s)$-Query: Let the $s$-session between the node $ID_i$ and $ID_j$, $B$ aborts the simulation. Let the session is at $ID_j$, $h_{j,t} = H_2(X_{j,t}, ID_j||ID_u)$. $B$ retrieves
\[ \eta_t \text{ and returns } F(\hat{e}(\eta t(aP, Q))) = \hat{e}(\eta t P - h_jtQ_j + h_jtQ_j, Q_j)^a = \hat{e}(X_jt + h_jtQ_j, Q_j)^a. \] In other cases the session key with its knowledge of static and ephemeral private keys.

7. Extract/Long-Term Key Reveal (sid)-Query: \( B \) first checks to see if there is an entry corresponding to \( ID_u \) in \( L_1 \). On no match, it makes a \( H_1 \) query with the input \( ID_u \). It retrieves the value \( \psi_u \) from \( L_1 \) and returns \( \psi_u(xP) \). Note that an Extract or Long-Term Key Reveal query on the input \( ID_j \) does not have to be simulated by \( B \). On such a query \( B \) aborts its execution.

8. Ephemeral Key reveal-Query: If this is the \( s^\text{th} \) session between \( ID_i \) and \( ID_j \), \( B \) outputs fail. For all the sessions at \( ID_j \), the Ephemeral Key Reveal is not handled. In all other cases, \( B \) returns the corresponding ephemeral private key it has chosen while answering the Send queries.

9. Test (sid)-Query: If this is not the \( s^\text{th} \) session between \( ID_i \) and \( ID_j \), \( B \) outputs fail. \( sid \) is the anticipated session but if it is not clean then \( B \) aborts its simulation. Otherwise, \( B \) has to return the session key held in the session \( sid \) or a random value from the session key distribution after tossing a coin. However, as shown below \( B \) cannot compute the real session key, which would be of form:

\[
SK = \hat{e}(Z_{i,s} + h_{i,s}Q_i, Q_j)^a
= \hat{e}(\lambda_i\gamma(cP) + h_{i,s}\lambda_iP, bP)^a
= \hat{e}(\gamma(c + h_{i,s})\lambda_iP, bP)^a
\]

It is required to solve BDH problem with instances \(<aP,bP,c\lambda_iP>\).

**Table 1. Security Comparison**

<table>
<thead>
<tr>
<th>Scheme</th>
<th>IKA</th>
<th>PFS</th>
<th>KKS</th>
<th>UKS</th>
<th>KcI</th>
<th>LoI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yuan Wang (Protocol-I)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Yuan Wang (Protocol-II)</td>
<td>√</td>
<td>×</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Gorantla et al. [4]</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Konstantinos Chalkias et al. [14]</td>
<td>√</td>
<td>×</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Proposed Protocol</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
</tbody>
</table>

* IKA: Implicit Key Authentication, PFS: Perfect forward secrecy, KKS: Known Key Secrecy, UKS: Unknown Key Share, KcI: Key-compromise Impersonation, LoI: Loss of Information

**Table 2. Comparison of Computational Cost**

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pairing Mul(G)</td>
<td>Pairing Mul(G)</td>
</tr>
<tr>
<td>Yuan Wang (Protocol-I) [15]</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Yuan Wang (Protocol-II) [15]</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Gorantla et al. [4]</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Proposed Protocol</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

6 Implementation issues

We can follow the similar technique used in [13] to implement the protocol in the single-hop setting in which each sensor node can establish the session key. We assume that the system public parameters and the master secret key are generated by the base station and embedded on each sensor node during the deployment. We assume that the base station is
powerful enough to perform computationally intensive cryptographic operations like pairing, hashing etc., and the sensor nodes, on the other hand, have limited resources in terms of computation, memory and battery power. We can use the sensor nodes MicaZ 3 in the implementation. The nodes are developed by Crossbow Technology. Its RF transceiver complies with IEEE 802.15.4/ZigBee, and the 8-bit microcontroller is Atmel ATmega128L, a major energy consumer. We used a PC of latest configuration as a base station. The program can be developed in nesC, C and Java. The base operating system for the MicaZ platform is TinyOS 2.0.

Let us consider n no of sensor nodes as \( s_n \) with identity \( ID \). The system parameters \( (G_1, G_2, q, \hat{e}, P, P_{pub}, H_0, H_1) \) is generated by the base station and all parameters will be embedded on each sensor node. One of the node will computes \( (U, V) \) and sends to its adjacent nodes. After receiving, the nodes will pass through the verification equation described in the protocol and computes the session key.

6.1 Pairing Algorithms

The most efficiently computable pairings are Weil and Tate pairing on elliptic curve. Tate pairing is more efficient than the Weil pairing. We consider \textbf{Type-1} pairing on super singular elliptic curve in our proposed protocol. Tate pairing is the bilinear mapping \( \hat{e}(P, Q) \), where \( P \) and \( Q \) are two arbitrary linearly independent points on an elliptic curve \( E(F_{q}^{k}) \), evaluates as an element of extension field \( F_{q^k} \). If \( P \) is of prime order \( r \), then the pairing is evaluated as an order of \( r \). We can apply algorithm-1 to compute the Tate pairing for implementation in more efficient way in term of memory space, bandwidth and processing speed.

We consider the following elements for implementation of pairing

- As we have discussed \textbf{Type-1} pairings is more suitable on super singular curves, these curves can be divided into three sub-classes curves over binary fields as \( q = 2^m \) with \( k = 4 \), curves over field of large prime characteristics \( 3 \) \( i.e \ q = 3^m \) with \( k = 6 \) and curves over field of large prime characteristics \( q = p, p > 3 \) with \( k = 2 \). The most suitable curve for implementation on 8-bit processor is curves over filed of prime characteristics is \( k = 4 \).
- The binary field \( F_{2^{71}} \) can be chosen to achieve the security.
- Super singular curve is

\[
y^2 + y = x^3 + x
\]  

The number of points on the curves is \( 2^{71} + 2^{136} + 1 = 487805.r \). Where \( r \) is a large prime.

The following table summarizes the cost of \( \eta_T(P, Q) \) on \( y^2 + y = x^3 + x \).

<table>
<thead>
<tr>
<th>Execution</th>
<th>( Mul )</th>
<th>( Sqr )</th>
<th>( Sqrt )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main loop</td>
<td>1904</td>
<td>544</td>
<td>544</td>
</tr>
<tr>
<td>Final loop</td>
<td>114</td>
<td>139</td>
<td>0</td>
</tr>
</tbody>
</table>

7 Conclusion

Here we have proposed a novel construction of one-pass key establishment protocol for WSNs which have the memory space required for each node is fixed. Also here the sensor node can establish secure communications with other adjacent nodes. The protocol is secure against perfect forward key secrecy and modification attack. Security of the proposed protocol relays on Bilinear Diffie-Hellman Problem. It achieves the security goals \textbf{Known Key secrecy}, \textbf{perfect forward secrecy}, \textbf{unknown key share}, \textbf{key compromise impersonation} and \textbf{key control}.
Algorithm 1: Computation of $\eta_T(P, Q)$ on $y^2 + y = x^3 + x + b$ curve over $\mathbb{F}_{2^m}$

Input $P, Q$
Output $\eta_T(P, Q)$

1. $P \leftarrow (x_P, y_P), Q = (x_Q, y_Q)$
2. $\theta \leftarrow x_P + 1$
3. $\sigma_1 \leftarrow \theta \cdot (x_P + x_Q + 1) + y_P + y_Q + 1 + (\theta + x_Q)\cdot u + v$
4. for $i = 1$ to $(m + 1)/2$
5. \hspace{1em} $\theta \leftarrow x_P, x_P \leftarrow \sqrt{x_P}, y_P \leftarrow \sqrt{y_P}$
6. \hspace{1em} $\sigma_2 = \theta \cdot (x_P + x_Q) + y_P + y_Q + x_P + (\theta + x_Q)\cdot u + v$
7. \hspace{1em} $\sigma_1 \leftarrow \sigma_1 \cdot \sigma_2$
8. $x_Q \leftarrow x_Q^2, y_Q \leftarrow y_Q^2$
9. end for
10. return $\sigma_1^{(2^m - 1)(2^m - 2(m + 1)/2 + 1)}$

References