LLRA: A Lightweight Leakage-Resilient Authentication Key Exchange Scheme for Smart Meters

Sree charan Sohan (21CSB0A40)
B Bharadwaj (21CSB0B07)
Jaiditya Beeraka (21CSB0B23)

National Institute of Technology Warangal

Faculty : Dr.Bala prakash

October 21, 2024

Overview

Introduction to security challenges in smart meters

- Ensuring secure information transmission between smart meters and Neighbourhood Area Communication Network (NAN) gateways.
- Vulnerability to side-channel attacks, where attackers can extract long-term private information from non-volatile memory.
- High authentication delay and energy consumption in existing Leakage-Resilient Authentication Key Exchange (LRAKE) protocols.
- Lack of lightweight AKE protocols designed specifically to mitigate side-channel attacks on smart meters.

Problem statement

Design of a Lightweight Leakage-Resilient Authentication Key Exchange (LLRA) protocol for smart meters that can resist side-channel attacks while minimizing authentication delay and energy consumption, addressing the limitations of existing LRAKE protocols in terms of performance and security.

Threat model

- The attacker can eavesdrop and intercept messages transmitted in the bidirectional wireless communication channel between the smart meter and the NAN gateway.
- The attacker can replay messages recorded from previous sessions.
- The attacker can inject forged messages.
- The attacker can spoof identities and important parameters.
- The attacker can exploit side-channel attacks to access sensitive information in smart meters and the NAN gateway.

Proposed solution

The proposed Lightweight Leakage-Resilient Authentication Key Exchange (LLRA) scheme consists of three main phases.

- Initialization
- Login and Authentication Key Exchange
- Refreshing

Initialisation

Initialisation consists of two main phases.

- User Registration
- Certificate Authority (CA) Initialization

User Registration

- Each user (UA and SB) registers their password and identity. They randomly select a salt value (sA for UA and sB) from Zq.
- Hash values hwA and hwB are computed using the user's password and salt. The derived keys DA and DB are created from the salt and user identity.

User Registration

- **User A (UA)**:
  1. Enters a password pw_A and identity ID_A.
  2. Selects a random salt $s_A \in \mathbb{Z}_q^*$.
  3. Computes:

$$hw_A = H_1(pw_A \oplus s_A)$$

$$DA = H_2(s_A) \oplus ID_A$$

  4. Records the salt s_A.

User Registration

- **User B (SB)**:
  1. Enters a password pw_B and identity ID_B.
  2. Selects a random salt $s_B \in \mathbb{Z}_q^*$.
  3. Computes:
$$hw_B = H_1(pw_B \oplus s_B)$$
$$DB = H_2(s_B) \oplus ID_B$$
  4. Records a password file containing hw_B and DB.

CA Initialization

- CA selects random numbers $a, b \in \mathbb{Z}_q^*$.
- UA sends ID_A and s_A to CA, which checks the uniqueness of ID_A.
- If unique, CA generates a public/private key pair for UA:
  - Selects $r_A \in \mathbb{Z}_q^*$.
  - Computes:
    $$R_A = r_A \cdot G$$
  - Computes:
    $$Z_A = H_1(a, b, s_A, s_B) \oplus H_1(hw_A, R_A)$$
  - Encodes r_A into two n-dimensional vectors:
    $$\vec{r}_{AL} = (r_{AL1}, \ldots, r_{ALn})$$
    $$\vec{r}_{AR} = (r_{AR1}, \ldots, r_{ARn})$$
  - Stores $\vec{r}_{AL}$ and $\vec{r}_{AR}$ independently and destroys r_A.

CA Initialization

- The same process occurs for SB with ID_B and s_B:
- CA sends $\{r_A, R_A, Z_A, Z_B, R_B\}$ to UA and $\{r_B, R_B, Z_B, Z_A, R_A\}$ to SB.

Login Phase

At User U_A

User inputs ID (ID_A) and password (PW_A)

$h_A = H(ID_A, PW_A)$ (Hash of credentials)

User U_A sends $(ID_A, Auth)$ to server S_B

At Server S_B

$h_A^* = H(ID_A, \text{stored } PW_A)$ (Recompute hash)

If $h_A = h_A^*$, the login is successful.

S_B sends Login_Success or Login_Fail.

Authentication Key Exchange Phase: At User U_A

On Input (Newsession, T_A, S_B)

$$y_A \leftarrow Z_q^*$$

$$\overrightarrow{v_A} = (y_A, r_{A1}, \ldots, r_{Al})$$

$$w_A = (1, T_A, r_{A1}, \ldots, r_{Al})^T$$

Compute:

$$u_A = v_A^T \cdot w_A$$

$$X_A = H(hw_A \cdot D_A \oplus D_B \oplus y_A)$$

$$Y_A = (u_A + y_A) \cdot G$$

$$K_A = Z_A \oplus H(h_A, R_A) \oplus X_A$$

$$\text{Auth} = \text{Enc}_{K_A}(Y_A \oplus T_A)$$

$$C_A = H(T_A, y_A, \text{Auth})$$

Send $CA = (X_A, Y_A, T_A, C_A, \text{Auth}_A)$ to S_B

Authentication Key Exchange Phase: At Server S_B (Part 1)

On Input (Newsession, T_B, U_A)

$$
\begin{aligned}
& y_B \leftarrow Z_q^* \\
& \overrightarrow{v_B} = (y_B, r_{B1}, \ldots, r_{Bl}) \\
& w_B = (1, T_B, r_{B1}, \ldots, r_{Bl})^T \\
& u_B = v_B^T \cdot w_B \quad \text{(compute inner product)} \\
& X_B = H(hw_B \cdot D_B \oplus D_B \oplus y_B) \\
& Y_B = (u_B + y_B) \cdot G
\end{aligned}
$$

If T_A is fresh:

Authentication Key Exchange Phase: At Server S_B (Part 2)

$$
\begin{aligned}
&K_A^* = Z_B \oplus H(hw_B, R_B) \oplus X_A \\
&y_A^* = \mathrm{Dec}_{K_A^*}(\mathrm{Auth}) \oplus T_A \\
&C_A^* = H(T_A, y_A^*, \mathrm{Auth}) \\
&\text{Verify } C_A^* = C_A \\
&V_B = (u_B + y_B) \cdot Y_A \\
&sk_B = X_A \oplus X_B \oplus H_3(V_B) \\
&E_B = \mathrm{Enc}_{sk_B}(T_B, hw_B, y_B, u_B, X_A, V_B, T_A) \\
&\text{Send } \langle X_B, Y_B, T_B, E_B \rangle \text{ to } U_A
\end{aligned}
$$

Key Verification Phase: At User U_A

If T_B is fresh, compute

$$V_A = (u_A + y_A) \cdot Y_B$$

$$sk_A = X_A \oplus X_B \oplus H_3(V_A)$$

Decrypt E_B to verify the session:

If valid, $X_B = X_B$ and $V_A = V_B$ are true.

Final session key: $SK_A = H(sk_A, \mathrm{sid}_A)$

Final Session Key Derivation: At Server S_B

If T_A is fresh, compute

$$\text{Verify } X_B = X_B \text{ and } V_B = V_A$$
$$\text{Final session key: } SK_B = H(sk_B, \text{sid}_B)$$

Refreshing

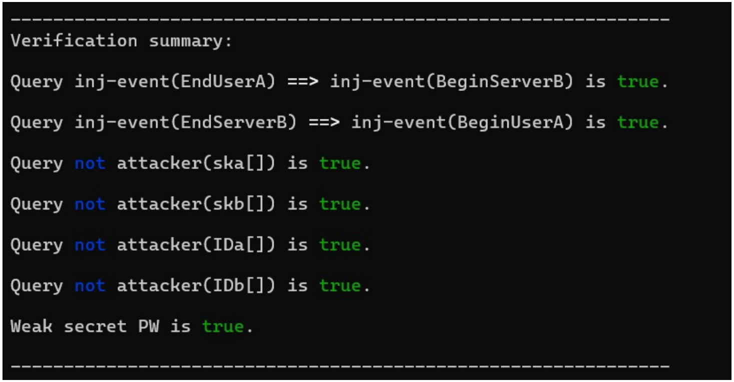
- UA and SB refresh their keys:
    - UA runs $\mathit{Refresh}_{n,1}^{\mathbb{Z}_q^*}$ on $(\vec{r}_{AL}, \vec{r}_{AR})$.
    - SB runs $\mathit{Refresh}_{n,1}^{\mathbb{Z}_q^*}$ on $(\vec{r}_{BL}, \vec{r}_{BR})$.

Security Analysis

Formal Verification Using ProVerif Tool:

- The LLRA protocol is formally verified using the Proverif tool to ensure its security claims.
- Security Properties Verified:
    - Authentication: Both parties verify each other's identity.
    - Confidentiality: Protection of sensitive keys and identifiers from unauthorized access.
    - Weak Secret Resistance: Security measures in place to protect user passwords from being easily guessed.
- Result: No attacks were detected during the verification process, confirming the robustness of the LLRA protocol.

Verified output using Proverif



```
--------------------------------------------------------------
Verification summary:

Query inj-event(EndUserA) ==> inj-event(BeginServerB) is true.

Query inj-event(EndServerB) ==> inj-event(BeginUserA) is true.

Query not attacker(ska[]) is true.

Query not attacker(skb[]) is true.

Query not attacker(IDa[]) is true.

Query not attacker(IDb[]) is true.

Weak secret PW is true.

--------------------------------------------------------------
```

Security Game: Queries (Part 1)

In the security game, a probabilistic polynomial-time (PPT) adversary A interacts with the instance oracles by making the following queries:

- **Execute($_i^s$, $_j^n$):** This oracle returns the transcript of all messages sent over the network to A.
- **Send($_i^s$, $_j^n$, msg, f):** This query returns the next message with the leakage boundary $f(r)$. $fLi\, r\pi_i^L \leq rL$ and $fRi\, r\pi_i^R \leq rR$.
- **Reveal($_i^s$):** This query returns a session key $\pi_i^s.sk$.

Security Game: Queries (Part 2)

- **Corrupt($_i{}^s$):** This query returns the long-term secrets of the instance π_i^s to A.
- **Test($_i{}^s$):** This query returns null if π_i^s does not have session key freshness (The BPR security model gives the definition of session key freshness). Only a fresh instance oracle can be Test-queried, which avoids the adversary trivially winning the security game.
- Otherwise, an unbiased coin b is tossed. This query returns $\pi_i^s.sk$ if $b = 0$ or a random value with the same length of sk if $b = 1$. If A's output b' is equal to b, then we say that A wins the security game.

Security Analysis

Theorem (Security of LLRA)

Theorem 1. *Suppose there is a PPT adversary A against the LLRA protocol. If $H_1(\cdot)$, $H_2(\cdot)$, and $H_3(\cdot)$ are modeled as random oracles, and the ECDLP and ECCDHP are hard under the subgroup generated by G on $E(\mathbb{F}_p)$, then the LLRA protocol is LLRA-secure.*
For any AKE adversary A against LLRA that runs in time at most t, involves at most n_p honest parties and activates at most k sessions, we show that there exist an ECCDH solver S and an ECDLP solver T, such that:

Security Analysis

Theorem (Security of LLRA)

$$Adv_{LLRA}^{A}(k) \leq 2n_p \cdot \epsilon' + \frac{(n_e + n_s)^2}{n_p} + \frac{n_h^2}{2^l} + 2\left(\frac{n_s + n_h}{|X|} + \frac{n_e + n_h}{|X|}\right) + \frac{4(n_s - 1)}{|X|} + 2(n_s + n_c)Adv_{ECDLP}^{A}(T) + 2n_c Adv_{ECCDH}^{A}(S)$$

where S runs in time $O(tk)$ and T runs in time $O(t)$. n_s, n_c, n_e, and n_h denote the number of Send, Corrupt, Execute, and H-hash oracles queried by A, respectively. The values l and $|X|$ denote the bit length of H's outputs and the cardinality of the dictionary X, respectively.

Formal Proof

Game G0

This game corresponds to a real attack scenario. Based on the security definition, we obtain:

$$\mathrm{Adv}_{\mathrm{LLRA}}^{A}(k) = 2\Pr[b' = b] - 1$$

where $\Pr[b' = b]$ denotes the probability that the adversary guesses the correct bit b.

Formal Proof

Game G1

This game simulates the hash oracle H. The execution of the Reveal, Send, Corrupt, and Test queries in this game is equivalent to executing an actual attack. Thus, we have:

$$\Pr[\xi_1] = \Pr[\xi_0]$$

Formal Proof

Game G2

In this game, a collision occurs based on the birthday attack. In the content simulation, the probability of collisions is at most:

$$\binom{n_e + n_s}{2} \cdot \frac{1}{n_p}$$

In the hash oracle simulation, the probability of collisions is at most:

$$\binom{n_h}{2} \cdot \frac{1}{2^l}$$

Thus, we have:

$$\begin{aligned} |\Pr[\xi_2] - \Pr[\xi_1]| &\leq \binom{n_e + n_s}{2} \cdot \frac{1}{n_p} + \binom{n_h}{2} \cdot \frac{1}{2^l} \\ &= \frac{(n_e + n_s)(n_e + n_s - 1)}{2n_p} + \frac{n_h(n_h - 1)}{2^{l+1}} \\ &\leq \frac{(n_e + n_s)^2}{2n_p} + \frac{n_h^2}{2^{l+1}} \end{aligned}$$

Formal Proof

Game G3

In this game, we analyze the leakage-resilience property in Definition 6.
We set $m = 1$ and $n > 20$; this setting guarantees that $n > 20 \cdot m$, $n \geq \frac{m}{3}$, and $n > 16$.
The leakage-resilient storage $\Lambda_{n,1}^{Z^*}$ is $(2\lambda, \epsilon)$-secure leakage-resilient, and the refreshing protocol $\mathsf{Refresh}_{n,1}^{Z^*}$ is (l, λ, ϵ'), where $l \in \mathbb{N}$, ϵ and ϵ' are negligible, and $\lambda = (0.15n \log q, 0.15n \log q)$.
Thus, we have:

$$|\Pr[\xi_3] - \Pr[\xi_2]| \leq n_p \cdot \epsilon'$$

Formal Proof

Game G4

In this game, A replaces Y_A with the random value x_{Y_A}. In order to distinguish x_{Y_A} from Y_A, A queries n_s times with a probability of $\mathrm{Adv}^{A}_{\mathrm{ECDLP}}(T)$.
Then, we have:

$$|\Pr[\xi_4] - \Pr[\xi_3]| \leq n_s \cdot \mathrm{Adv}^{A}_{\mathrm{ECDLP}}(T)$$

Formal Proof

Game G5

In this game, the ciphertext E_B is replaced with the encryption output of hw'_B. If A can distinguish Game G5 from Game G4, then the semantic security of authenticated encryption will be compromised in at most $n_e + n_h$ instances, each with a probability of $\frac{1}{|X|}$.
Thus, we have:

$$|\Pr[\xi_5] - \Pr[\xi_4]| \leq \frac{n_e + n_h}{|X|}$$

Formal Proof

Game G6

In this game, the ciphertext E_A is replaced with the encryption result of hw_A. If E_A is a valid ciphertext, the session key is set to be identical to that of π_j^n; otherwise, the session key is uniformly chosen from the elements in the dictionary of size $|X|$.
In total, there are $n_s + n_h$ events, each with a probability of $\frac{1}{|X|}$.
Therefore, we have:

$$|\Pr[\xi_6] - \Pr[\xi_5]| \leq \frac{n_s + n_h}{|X|}$$

Formal Proof

Game G7 - Case 1

In this case, A makes a Corrupt(π_i^s) query but no Send($\pi_j^n, \pi_i^s, \cdot, \cdot$) query.
A can obtain identity information by issuing n_c Corrupt queries on π_i^s.
Meanwhile, A finds it difficult to retrieve y_A from X_A and Y_A.
Thus, if A can correctly compute the values k_A with advantage $\mathrm{Adv}_{\mathrm{ECDLP}}^{A}(T) + \mathrm{Adv}_{\mathrm{ECCDH}}^{A}(S)$, A can forge E_A with the same advantage.

Formal Proof

Game G7 - Case 2

In this case, A makes a Corrupt(π_i^s) query and a Send$(\pi_j^n, \pi_i^s, \cdot, \cdot)$ query. A obtains identity information of π_i^s by issuing a Corrupt query. A can set x_{y_A} and x_{u_A} and request a valid message msg$'_2$ using the Send$(\pi_j^n, \pi_i^s, \text{msg}'_1, \cdot)$ query.
Then A computes the value $x_{sk_A} = X_A \oplus X_B \oplus H_3(Y_B \cdot (x_{u_A} + x_{y_A}))$. If A executes $n_s - 1$ times Send queries to guess $ID_{\pi_j^n}$, the probability that A produces a valid E_A from x_{sk_A} is bounded by $\frac{1}{|X|}$.
Therefore, we have:

$$|\Pr[\xi_7] - \Pr[\xi_6]| \leq (n_s - 1) \cdot \frac{1}{|X|} + n_c \cdot \left(\text{Adv}^{A}_{\text{ECDLP}}(T) + \text{Adv}^{A}_{\text{ECCDH}}(S)\right)$$

Formal Proof

Game G8

Similar to Case 2 of Game G7, we can derive:

$$|\Pr[\xi_8] - \Pr[\xi_7]| \leq (n_s - 1) \cdot \frac{1}{|X|}$$

In total, there are $n_s + n_h$ events, each with a probability of $\frac{1}{|X|}$.

Formal Proof

Game G9

This game serves as a bridging step where the advantage of A in guessing b is completely eliminated. Therefore, we obtain:

$$\Pr[\xi_9] = \Pr[\xi_8] = \frac{1}{2}.$$

Thus, we have:

$$\begin{aligned}\mathrm{Adv}^{A}_{\mathrm{LLR}}(k) \leq 2n_p \cdot \epsilon' + \frac{(n_e + n_s)^2}{2n_p} + \frac{n_h^2}{2^l} \\ + 2\left(\frac{n_s + n_h}{|X|} + \frac{n_e + n_h}{|X|}\right) + 4\frac{(n_s - 1)}{|X|} \\ + 2(n_s + n_c)\mathrm{Adv}^{A}_{\mathrm{ECDLP}}(T) + 2n_c\mathrm{Adv}^{A}_{\mathrm{ECCDH}}(S).\end{aligned}$$

CLR-eGNY Logic Analysis

According to definitions in Dziembowski et al. [31], we create a new GNY expression.

Leakage-Resilient Storage Scheme

Based on previous studies, $P \mid\equiv \Lambda(r)$ (P believes, or is entitled to believe, that private key r is leakage-resilient) can be generalized as follows:

$$\Lambda_{n,m}\mathbb{Z}_q^* = (\mathrm{Encode}_{n,m}\mathbb{Z}_q^*, \mathrm{Decode}_{n,m}\mathbb{Z}_q^*)$$

is a $(2\lambda_\Lambda, \epsilon)$-secure leakage-resilient storage scheme, and

$$\mathrm{Refresh}_{n,m}\mathbb{Z}_q^*$$

CLR-eGNY Logic Analysis

Leakage-Resilient Storage Scheme

There is a leakage function $f = (f_1, f_2)$, where:

$$f_1(r_L) = f_2(r_R) = 0.15n \log q \text{ bits}$$

P leaks a maximum of $f_1(r_L)$ and $f_2(r_R)$ bits of the long-term private key.

$P \ni \text{Encode}_{n,m}\mathbb{Z}_q^*(r),\ P \mid\sim (f_1(r_L), f_2(r_R)),\ P \ni SK,\ P \ni \text{Refresh}_{n,m}\mathbb{Z}_q^*(r_L,$

Finally, P believes the private key r is leakage-resilient:

$$P \mid\equiv \Lambda(r)$$

Formal Proof - LRJ1

Leakage-Resilient Jurisdiction Rule (LRJ1)

Suppose that for principal P, all of the following conditions hold:

1. P believes that Q is an authority on some statement $C(r_P, r_Q)$;
2. P believes that Q believes in $C(r_P, r_Q)$;
3. P believes that r_P is leakage-resilient;
4. Q believes that r_Q is leakage-resilient.

Then P ought to believe in $C(r_P, r_Q)$ as well:

$$P \mid\equiv Q \mid\Longrightarrow C(r_P, r_Q),\ P \mid\equiv Q \mid\equiv C(r_P, r_Q),$$

$$\text{P} \ \mid\equiv \Lambda(r_P),\ Q \mid\equiv \Lambda(r_Q)\ \Rightarrow\ P \mid\equiv C(r_P, r_Q)$$

Formal Proof - LRJ2

Leakage-Resilient Jurisdiction Rule (LRJ2)

Suppose that for principal P, all of the following conditions hold:

1. P believes that Q is honest and competent;
2. P receives a message $X \Rightarrow C(r_P, r_Q)$ which P believes Q conveyed;
3. P believes that X is fresh;
4. P believes that r_P is leakage-resilient;
5. Q believes that r_Q is leakage-resilient.

Then P ought to believe that Q really believes $C(r_P, r_Q)$:

$$P \mid\equiv Q \mid\Longrightarrow Q \mid\equiv *,\ P \mid\equiv Q \mid\sim (X \Rightarrow C(r_P, r_Q)),$$

$$\text{P} \ \mid\equiv \#X,\ P \mid\equiv \Lambda(r_P),\ Q \mid\equiv \Lambda(r_Q)\ \Rightarrow P \mid\equiv Q \mid\equiv C(r_P, r_Q)$$

Formal Proof - LRJ3

Leakage-Resilient Jurisdiction Rule (LRJ3)

Suppose that for principal P, all of the following conditions hold:

1. P believes that Q is honest and competent;
2. P believes that Q believes that Q believes in $C(r_P, r_Q)$;
3. P believes that r_P is leakage-resilient;
4. Q believes that r_Q is leakage-resilient.

Then P ought to believe that Q really believes $C(r_P, r_Q)$:

$$P \mid\equiv Q \mid\Longrightarrow Q \mid\equiv *,\ P \mid\equiv Q \mid\equiv Q \mid\equiv C(r_P, r_Q),$$

$$\text{P} \ \mid\equiv \Lambda(r_P),\ Q \mid\equiv \Lambda(r_Q)\ \Rightarrow P \mid\equiv Q \mid\equiv C(r_P, r_Q)$$

Goals of the Protocol

The shared session keys between UA and UB shall achieve the following goals:

Formal Goals

1. **Goal 1:** $UA \mid\equiv \#SK$
   UA believes that the session key SK is fresh.
2. **Goal 2:** $UA \mid\equiv \varphi SK$
   UA believes that UB possesses the session key.
3. **Goal 3:** $UA \mid\equiv UB \ni SK$
   UA believes that UB believes in the session key.
4. **Goal 4:** $UB \mid\equiv \#SK$
   UB believes that the session key SK is fresh.
5. **Goal 5:** $UB \mid\equiv \varphi SK$
   UB believes that UA possesses the session key.
6. **Goal 6:** $UB \mid\equiv UA \ni SK$
   UB believes that UA believes in the session key.

Attacks mitigated

- **Perfect Forward Secrecy:** If UA's long-term private values are compromised, the adversary M cannot compute sk_A or sk_B without knowing the ephemeral keys y_A or y_B.
- **Resistance to Off-line Password Guessing:** The equation

$$X_A = H_2(hw_A \oplus DA \oplus u_A \oplus y_A)$$

  protects hw_A from being guessed by adversary A.
- **Resistance to Replay Attacks:** Fresh timestamps TA and TB ensure immunity to replay attacks.

Attacks mitigated

- **Mutual Authentication:** SB extracts y_A^*, computes C_A^*, and proves legitimacy to UA, who verifies $X_A^* = X_A$ and $V_B^* = V_A$ before responding.
- **Resistance to Key Compromise Impersonation:** Even if UA's hw_A and DA are compromised, A cannot impersonate SB due to the unknown y_B.

Attacks mitigated

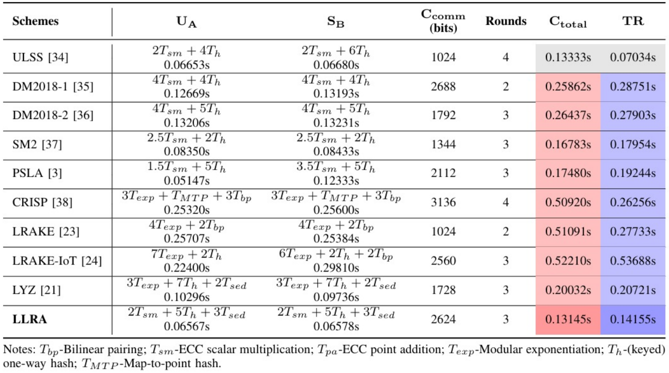
- **Resistance to Man-in-the-Middle Attacks:** A cannot pass the verification checks:

$$C_A^*? = CA, \quad X_A^*? = X_A, \quad V_B^*? = V_A.$$

- **Resistance to Impersonation Attacks:** A must obtain long-term secrets to pass verification, making impersonation difficult.
- **Resistance to DoS Attacks:** Verification of timestamps TA and TB prevents message processing if verification fails.

Performance Analysis

Table VII: Computational and Communication Costs.



Schemes	$\mathrm{U_A}$	$\mathrm{S_B}$	$\mathrm{C_{comm}}$ **(bits)**	**Rounds**	$\mathrm{C_{total}}$	**TR**
ULSS [34]	$2T_{sm}+4T_h$ 0.06653s	$2T_{sm}+6T_h$ 0.06680s	1024	4	0.13333s	0.07034s
DM2018-1 [35]	$4T_{sm}+4T_h$ 0.12669s	$4T_{sm}+4T_h$ 0.13193s	2688	2	0.25862s	0.28751s
DM2018-2 [36]	$4T_{sm}+5T_h$ 0.13206s	$4T_{sm}+5T_h$ 0.13231s	1792	3	0.26437s	0.27903s
SM2 [37]	$2.5T_{sm}+2T_h$ 0.08350s	$2.5T_{sm}+2T_h$ 0.08433s	1344	3	0.16783s	0.17954s
PSLA [3]	$1.5T_{sm}+5T_h$ 0.05147s	$3.5T_{sm}+5T_h$ 0.12333s	2112	3	0.17480s	0.19244s
CRISP [38]	$3T_{exp}+T_{MTP}+3T_{bp}$ 0.25320s	$3T_{exp}+T_{MTP}+3T_{bp}$ 0.25600s	3136	4	0.50920s	0.26256s
LRAKE [23]	$4T_{exp}+2T_{bp}$ 0.25707s	$4T_{exp}+2T_{bp}$ 0.25384s	1024	2	0.51091s	0.27733s
LRAKE-IoT [24]	$7T_{exp}+2T_h$ 0.22400s	$6T_{exp}+2T_h+2T_{bp}$ 0.29810s	2560	3	0.52210s	0.53688s
LYZ [21]	$3T_{exp}+7T_h+2T_{sed}$ 0.10296s	$3T_{exp}+7T_h+2T_{sed}$ 0.09736s	1728	3	0.20032s	0.20721s
LLRA	$2T_{sm}+5T_h+3T_{sed}$ 0.06567s	$2T_{sm}+5T_h+3T_{sed}$ 0.06578s	2624	3	0.13145s	0.14155s

Notes: T_{bp}-Bilinear pairing; T_{sm}-ECC scalar multiplication; T_{pa}-ECC point addition; T_{exp}-Modular exponentiation; T_h-(keyed) one-way hash; T_{MTP}-Map-to-point hash.

Performance Analysis

- The experimental results clearly demonstrated that LLRA can effectively reduce the computational time and running time. Compared to benchmark protocols, the running time of LLRA was decreased by at least 14% . When the computing capabilities of both parties are comparable, the total computa tional time of LLRA was decreased by at least 33% , making it more efficient and more suitable for smart meters.

Conclusion

- The Lightweight Leakage-Resilient Authentication Key Exchange (LLRA) scheme addresses long-term private key leakage due to side-channel attacks in smart meters.
- The protocol's security is validated through formal verification using the ProVerif tool, ensuring the reliability of its cryptographic claims.

Drawbacks and Possible Improvements

- **Drawback**: High authentication delay due to complex cryptographic operations.
  Possible Improvement: Session Key Reuse: Allow session key reuse for a defined period, thereby minimizing the number of key exchanges needed for subsequent authentications..
- **Drawback**: Increased energy consumption during authentication due to resource-intensive operations.
  Possible Improvement: Asynchronous Communication: Utilize asynchronous communication techniques that allow smart meters to sleep during idle times, thereby conserving energy..

Thank You