Serverless lightweight mutual authentication protocol for small mobile computing devices
Collins Isdory Mtita, Maryline Laurent, Pascal Daragon

To cite this version:
Serverless Lightweight Mutual Authentication Protocol for small mobile Computing Devices

Collins MTITA\(^1\), 2, Maryline LAURENT\(^2\), Pascal DARAGON\(^1\)

\(^1\)TRAXENS S.A.S, Marseille, France
\(^2\)Institut Mines-Télécom/Télécom SudParis, CNRS UMR 5157 SAMOVAR, Évry, France
\(^{1,2}\)collins.mtita, maryline.laurent@telecom-sudparis.eu

Abstract

Small and inexpensive computing devices are becoming potential players in the Internet arena. Smart constrained devices are used for collecting or generating information which is later relayed to the designated servers. The information gathered must be adequately protected against all kinds of attacks during storage and transmission. However, most smart constrained devices have limited resources and unable to run most of the classical protocols that provide robust security.

This article presents a lightweight mutual authentication protocol for resource constrained heterogeneous computing devices with high mobility. Our solution is novel as it facilitates two resource constrained devices to autonomously perform mutual authentication and establish a session key without sharing common parameters beforehand. Our proposed lightweight protocol is symmetric and uses simple primitives such as XoR, comparison and keyed-Hashed Message Authentication Code (HMAC) during mutual authentication. Moreover, our protocol requires minimum storage for storing few keys and parameters locally. The protocol is formally verified using AVISP tool.

Keywords: Security, Low energy footprint, lightweight security, Validation, Mutual Authentication, Heterogeneous Constrained Computing Devices, Symmetric Protocol

1 Introduction

Smart constrained computing devices are slowly becoming major players in the Internet arena dominated by the era of ubiquitous computing. There is a steady emergence of platforms \([5, 18]\) that allow smart constrained computing devices to independently interconnect and interact with other parties over the Internet in ways that were not initially envisaged.

Before harnessing the full potential brought by these technologies into our daily lives we must first address core security and privacy issues pertaining to the information stored and exchanged. Technology growth forces smart constrained devices to release the information gathered on real-time basis. However, this poses new challenges \([20, 27, 31]\) because constrained devices do not have enough capabilities to provide access rights to various external entities \([5]\) without the help of a powerful central server.

Using the current centralized server models, services provided by smart devices in remote geographical locations may be regularly interrupted due to the lack of reliable connectivity between smart computing devices and centralized server. Likewise, it is challenging to grant access control when the interaction between smart constrained devices and external parties cannot be known in advance, nonetheless, appropriate access rights must be assigned to each requesting party in order to allow smooth authentication and reliable access to the information. In such cases, a reliable solution necessitates authentication to proceed without a persistent connection to the central server.

In this paper we address the issue of securing data stored or exchanged by smart computing devices in the absence of the centralized server. We propose a serverless mutual authentication protocol for mobile resource-constrained devices. Our protocol is symmetric, lightweight and uses simple primitives like keyed-Hash Message Authentication Code (HMAC), XoR and comparison operations to validate authenticity of all parties accessing data in smart devices.

The rest of this paper is organized as follows, section 2 presents a scenario pertaining to the protocol discussed in this article while section 3 presents related work. Section 4 discusses our proposed protocol and section 5 validates our protocol using AVISP tool. Protocol’s analysis is done in section 6 and section 7 concludes.

2 Serverless Protocol Scenario and Features

Imagine a scenario where tracking devices are attached to containers in transit from one geographic location to another. Tracking devices collect and log crucial information in the course of the trip but also serve as identifiers for respective containers they are attached to. On transiting custom checkpoints, controllers possessing mobile personal digital assistants (PDAs) must authenticate to tracking devices before identifying and accessing stored information, for instance container number, type of merchandise, destination e.t.c. But, PDAs must obtain prior authorization to access information in tracking devices from a remote central server, which is only accessible via the network.

One way to accomplish this task requires PDAs to establish persistent connections to the central server during the authentication process. However, in case of network unavailability, the protocol breaks down and PDAs cannot communicate with tracking devices. The journey must be delayed until connection is established. This leads to unnecessary inconveniences.

Alternatively, PDAs may connect once or twice to the central server, whenever the network is available during the day, download appropriate authentication data and store them locally. In that case, PDAs can autonomously authenticate tracking devices without the need of a persistent connection to the central server. This guarantees reliable authentication between legitimate PDAs and tracking devices, even when the central server is unavailable or incapacitated.

Our proposed scenario’s network architecture contains three types of communicating parties - Central Server (CS), Lightweight Responder (LR) and Lightweight Initiator (LI).

1. Central Server (CS): A powerful server with unlimited resources that controls access between LI and LRs and administers LRs.

2. Lightweight Initiator (LI): A terminal accessing information stored in LRs. LI has enough resources in terms of computing power and storage space.

3. Lightweight Responder (LR): A resource constrained terminal with limited resources in terms of storage capacity, computation power and energy. LR constantly collects and stores sensitive data. Each LR has a secret key \(K_C\) provided by CS and a static timestamp \(T_c\) initialized by CS during device setup. LRs can be RFID Tags, NFC Tags or other constrained data capturing devices.

LR is the most important player in the scenario that our proposed protocol aims at protecting.

In our scenario, LRs are geographically distributed in form of clusters. A cluster is a collection of LRs within a small geographic region sharing the same secret key \(K_c\) that is used to verify authenticity of external entities interacting with LRs within a cluster. LI is an external device accessing information stored in LRs within a given cluster. Prior to accessing LRs in a cluster, LI must securely connect to CS and request authorization to access LRs within a given...
cluster. LI’s request contains its geographic position, authenticating parameters and the identifier IDL. As CS knows all legitimate LR s and LR clusters, their secret keys and respective geographic positions, it replies to LI by providing necessary parameters for accessing requested cluster of LR s. Some of these parameters are LI’s key KLI, access rights AR, Time Window W5 and list of temporary identities L1 for LR s within a cluster.

Our proposed scenario has the following basic features:

1. **Constrained resources**: LR s cannot memorize credentials and access control rights [10], [36] for each LI due to limited resources.
2. **No prior knowledge of each other**: LI and LR have no knowledge of each other’s existence prior to mutual authentication phase.
3. **Absence of Server during authentication session**: CS may not be available during mutual authentication phase due to unreliable connectivity.
4. **Scalability**: Within a cluster, LIs and LR s can freely interact.
5. **Trust relationship**: LR and LI do not have mutual trust. The trust relationship is built during authentication with the help of CS, even if it is not actively involved in the authentication process.

## 3 Related Work

Various security protocols for constrained devices have been put forward, most of which advocate the availability of a persistent connection to the central server during authentication [1], [23]. However, connection-oriented model faces major limitations [22], especially in the era of ubiquitous computing, where devices can be located anywhere without a guarantee of reliable connectivity [20]. Intermittent connectivity may render protocols unavailable during long periods of time. This is one of the major reasons behind the efforts to seek alternative and reliable solutions. This section analyzes some of the proposed serverless protocols for constrained devices.

In 2008, Tan et al. [32] first introduced the **Serverless search and authentication protocol**. Tan et al. suggested that protocol’s reliability can be guaranteed by eliminating a persistent link between an RFID reader and the backend during authentication process. Their proposal included downloading necessary information that can allow a reader to autonomously authenticate tags. However, Tan et al.’s protocol was found to be vulnerable to traceability, impersonation and privacy attacks by the authors of [33]. In 2009 Lin et al. [21] proposed a serverless RFID authentication protocol which improved the computational performance of Tan et al. protocol. However, as pointed out by authors of [19], Lin et al.’s, like with Tan et al.’s, proposed protocol performed a one-sided authentication, where the reader authenticates the tag but the tag does not authenticate the reader. Moreover, authors of [19] reveal that Lin et al.’s protocol is also vulnerable to impersonation attack.

Hoque et al. [15] proposed a serverless, untraceable authentication, and forward secure protocol for RFID tags. Hoque et al. claim that their protocol safeguards both reader and tags against common attacks without the need of backend server’s intervention. But, Deng et al. [11] found that Hoque et al.’s authentication protocol was susceptible to data desynchronization attack and proposed an improved version. Deng et al.’s proposed authentication protocol was designed to withstand data desynchronization attacks, but the authors of [29] found that Deng et al.’s protocol is still vulnerable to data desynchronization attack after two protocol runs.

The authors of [2] proposed ERAP, the **ECC based RFID Authentication Protocol**, which performs mutual authentication between the reader and the authorized RFID tags without the need of persistent connection to the backend server. This scheme was found vulnerable to denial of service attack by authors of [23]. The authors of [33] propose (HOA) **HLR Offline Authentication**, the authentication scheme suitable for low-power mobile devices based on ECC. However, this protocol requires prior knowledge of each communicating entity and too much CPU and memory resources as tags must perform ECC point multiplication and modular operations.

The idea to use timestamp during authentication of constrained devices was first introduced by Tsidik [34]. Tsidik’s idea was quite novel but curious due to the fact that most of the constrained devices do not have embedded clocks to keep track of the time. Tsidik suggested that RFID reader should periodically broadcast timestamp of its current time. A tag, within the proximity of reader, receives and compares the broadcast timestamp with a stored timestamp value. If the former is strictly greater than the latter, the tag computes a response derived from its permanent key and the new timestamp. Otherwise, the tag replies with a pseudorandom to confuse the adversary and thwart narrowing attacks. According to Tsidik [34], a narrowing attack occurs when the adversary queries a tag with a particular timestamp and then later tries to identify the same tag by querying a candidate tag with a timestamp slightly above the previous one.

However, Tsidik’s idea is vulnerable to Denial of Service (DoS) attacks. An adversary can easily desynchronize the tag by sending the timestamp value that is ahead of time. This idea was later improved by authors of [8] by moving the attack from the resource constrained tag to the powerful backend server. The improvement aimed at thwarting DoS attacks against the tags but it also resulted to an exhaustive search to the backend server.

## 4 Serverless Mutual Authentication Protocol

Our protocol leverages on the power of CS’s knowledge on LR clusters with their respective credentials to facilitate authentication, even though CS does not actively participate during mutual authentication phase. To further facilitate authentication between two constrained devices, our protocol uses timestamp as one of the parameters during authentication between LI and LR.

### 4.1 Security and Privacy Requirements

Our proposed authentication protocol must fulfill the following requirements.

1. **Mutual Authentication**: Our protocol must perform mutual authentication between LR and LI prior to data exchange session in order to thwart impersonation attacks.
2. **Key Exchange**: Our protocol must securely establish a common key between LR and LI to be used during data exchange session.
3. **Freshness**: Messages exchanged during mutual authentication session must be fresh. Our protocol uses timestamp and random values to enforce freshness.

### 4.2 Privacy and Security threat models

Several attacks may be launched against our protocol. We propose privacy and security games to model possible threats and demonstrate how resilient our protocol can be against attacks.

**Game 1**: \( \beta \) masquerades as LI

- **Phase 1.1**: \( \beta \) eavesdrops several exchanges between one or more LR and various LIs.
- **Phase 1.2**: \( \beta \) sends message \( b_1 \) and then message \( b_3 \) to LR. \( \beta \) wins the game if he can reply LR with a valid message \( b_3 \).

**Game 2**: LR\(_i\)s

- **Phase 2.1**: \( \beta \) colludes with a legitimate device LI and listens to exchanges between LI and responder LR\(_i\) and then between LI and LR\(_j\).
- **Phase 2.2**: Challenger selects LR\(_i\), \( i \in \{1,2\} \), \( \beta \) listens to exchanges between LI and LR\(_i\), and \( \beta \) sends a guess \( i \) value to the challenger.

**Game 3**: \( \beta \) depletes LR\(_i\)’s resources

- **Phase 3.1**: \( \beta \) eavesdrops messages \( b_1 \) between LIs and LRs.
- **Phase 3.2**: \( \beta \) sends forgery \( b_1 \) messages to a targeted LR within a cluster, \( \beta \) wins the game if he can successfully deplete LR’s battery within 12 hours (This corresponds to an overnight attack).
4.3 Assumptions

1. All LR}s running our protocol are capable of performing simple primitives such as Keyed-Hash Message Authentication Code (HMAC), comparison and XOR. In this article, HMAC is based on SHA1 (Secure Hash Algorithm 1) [13] and its output is truncated to the 128 bits (energy saving). However, our protocol can work with any HMAC.

2. Pseudo Random Number Generator (PRNG) and HMAC are robust.

3. CS and LI share secret parameters used to launch a secure channel, e.g., via secure protocol https, for exchanging secret information.

4. CS shares a secret key $K_C$ with each LR. $K_C$ is common among all legitimate LR}s within a specific cluster.

5. LI’s periodic key $K_I$ is used to solicit LR}s within a specific cluster sharing the same key $K_C$, provided Time Window $W_S$ is still valid.

4.4 Protocol Notations

Table 1 presents notations used in the protocol. Access Rights (AR) is a code for access levels and rights that LI has pertaining to the data stored in LR. In our protocol, AR is represented in form of a code, like Unix file permissions, with Read, Write and Execute options. $H_K$ is a secret parameter that LI uses to securely pass initial parameters to LR{s. LR’s static timestamp $T_{LC}$ is initialized by a default timestamp value $T_{init}$ during initial configuration by CS. Time Window $W_S = [T_0·[T_{Z}]]$ is a 64 bit parameter made from two timestamps, 32 bits start timestamp $T_0$ and 32 bits end timestamp $T_Z$ parameters used to show $K_I$’s validity.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Symbol</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>LI’s system Time</td>
<td>$T_{LI}$</td>
<td>32</td>
</tr>
<tr>
<td>LR’s stored timestamp</td>
<td>$T_{C}$</td>
<td>32</td>
</tr>
<tr>
<td>Start Time Window</td>
<td>$T_0$</td>
<td>32</td>
</tr>
<tr>
<td>End Time Window</td>
<td>$T_{Z}$</td>
<td>32</td>
</tr>
<tr>
<td>Time Window</td>
<td>$W_S$</td>
<td>64</td>
</tr>
<tr>
<td>LI’s identifier</td>
<td>$ID_L$</td>
<td>128</td>
</tr>
<tr>
<td>LI’s key</td>
<td>$K_L$</td>
<td>128</td>
</tr>
<tr>
<td>LR’s identifier</td>
<td>$ID_L$</td>
<td>128</td>
</tr>
<tr>
<td>LR’s cluster Key</td>
<td>$K_C$</td>
<td>128</td>
</tr>
<tr>
<td>Derived session key</td>
<td>$K_S$</td>
<td>128</td>
</tr>
<tr>
<td>Timestamp signature</td>
<td>$H_T$</td>
<td>128</td>
</tr>
<tr>
<td>Random Value</td>
<td>$R_L$</td>
<td>128</td>
</tr>
<tr>
<td>Access Rights</td>
<td>$AR$</td>
<td>128</td>
</tr>
<tr>
<td>LI’s secret code</td>
<td>$H_K$</td>
<td>128</td>
</tr>
<tr>
<td>List of LR{s temporary identities</td>
<td>$l_j$</td>
<td>-</td>
</tr>
</tbody>
</table>

4.5 Protocol Description

The proposed protocol operates in two phases. Phase A involves interaction between LI and CS, and Phase B involves interaction between LI and LR.

Phase A: Interaction between CS and LI

LI requests authorization from CS to access information stored in LR{s. CS also uses this phase to synchronize time with LI. LI securely connects to CS and sends message $a_1$ for requesting authorization to access a cluster of LR{s in its vicinity. Message $a_1$ contains LR’s identifier $ID_L$ and its geographic location. CS receives LI’s request and generates a list $l_j$ of temporary identities for each LR within a cluster (cf. Line (4)). LI’s key $K_I$ (cf. Line (5)) and secret code $H_K$ (cf. Line (6)). CS sends back message $a_2$ via the established secure channel (cf. Line (7)). LI receives and decrypts message $a_2$ from CS containing $K_L$, $H_K$, $AR$, $W_S$ and $l_j$.

$$LI \xrightarrow{a_1} CS : \text{Request} \hspace{1cm} \text{(1)}$$

$$CS \xrightarrow{a_2} LI : \begin{cases} \text{Get Time Window} : W_S = [T_0, T_Z] \hspace{1cm} \text{(2)} \\ \forall LI \text{ generate } Id_{Temp} = HMAC_{id}(T_0) \hspace{1cm} \text{(3)} \\ \text{Create list } l_j = \{Id_{Temp}, Id_{Temp}, ..., Id_{Temp}\} \hspace{1cm} \text{(4)} \\ K_L = HMAC_{K_C}(Id_{Temp} || AR \| W_S) \hspace{1cm} \text{(5)} \\ H_K = HMAC_{K_C}(W_S) \hspace{1cm} \text{(6)} \end{cases}$$

Phase B: Mutual Authentication Between LR and LI

Mutual authentication between LI and LR is completed in three exchanges without assistance from CS.

LI encrypts its identifier $ID_L$ using its secret code $H_K$ (cf. Line (10)) and calculates timestamp signature $H_T$ (cf. Line (11)). LI broadcasts message $b_1$ containing $Id_L$, $timestamp$ $T_L$, signature $H_T$, together with Access Rights $AR$, and Time Window $W_S$ from CS. LI precalculates and stores in a table the values of $H_T$ corresponding to each $Id_{Temp}$ (cf. Line (12)). This step facilitates the search of $H_T$ values once message $b_2$ is received.

Upon receipt of message $b_1$, LR verifies $T_Z$ and $T_{LI}$ against its stored timestamp $T_C$, where $T_Z$ is last 32 bits of timestamp $W_S$ as explained in Section 4.3. LI aborts the current session if the timestamp parameters are not correct.

B1. Validation of Key $K_I$

Each valid LR can easily calculate $H_K$ using $W_S$ and $K_C$ (cf. Line (16)), then use it to recover $ID_L$ (cf. Line (17)) which is used to calculate $K_L$ (cf. Line (18)). Using $K_L$, LR authenticates timestamp signature $H_T$ (cf. Line (20)). If $H_T$ is invalid, the session is terminated, else LR updates its timestamp $T_C$ (cf. Line (21)). A valid $H_T$ in Line (20) proves the validity of key $K_I$ used by LI.

After verification of $K_I$, LR generates a temporary identity using its identifier $Id_L$ and $T_0$ from $W_S$ (cf. Line (24)) and also generates a signature $H_1$ for $T_L$ (cf. Line (25)). Likewise, LR generates a random $R_L$ (cf. Line (26)), which is used to calculate a cipher $c_1$ and a signature $H_2$ (cf. Lines (27) and (28)). Cipher $c_2$ and signatures $H_1$ and $H_2$ are sent to LI via message $b_2$.

B2. Authentication of LR

LI receives message $b_2$ and quickly finds a value that corresponds to a signature $H_1$ within a table of precalculated $H_1$ values. Once found, $R_L$ can be easily recovered as indicated in Line (31) and its validity tested by comparing calculated $H_2$ against the received $H_2$. Validity of Line (33) authenticates LR in LI and prompts LI to calculate $H_R = HMAC_{Id_{Temp}}(K'_I)$. $H_R$ is sent to LR.

B3. Authentication of LI

LR receives and verifies $H_R$, if correct, LR authenticates LI (cf. Line (40)).

B4. Generation of session key $K_S$

Lines (37) and (41) show generations of key $K_S$ on LI and LR respectively after a successful mutual authentication session. The session key $K_S$ is generated from random values exchanged between LR and LI. With a shared session key $K_S$, LR and LI can securely exchange data, for instance using Advanced Encryption Standard (AES).

$$LI \xrightarrow{b_2} CS : \begin{cases} H_K = HMAC_{K_C}(W_S) \hspace{1cm} \text{(16)} \\ Id_L = HMAC_{K_C}(T_L) \oplus id_{ID} \hspace{1cm} \text{(17)} \\ K'_I = HMAC_{K_C}(Id_{Temp} \| AR \| W_S) \hspace{1cm} \text{(18)} \\ H_T = HMAC_{K'_I}(T_L) \hspace{1cm} \text{(19)} \end{cases}$$

$$LR \xrightarrow{b_1} LI : \begin{cases} H_T, c_1, AR, W_S, T_L \hspace{1cm} \text{(13)} \end{cases}$$

$$LI \xrightarrow{b_2} LR : \begin{cases} \text{if } T_Z < T_C \text{ or } T_Z < T_{LI} \text{ or } T_L < T_C \text{ or } T_0 < T_{LI} \hspace{1cm} \text{(14)} \end{cases}$$

END SESSION (15)

$$H_K = HMAC_{K_C}(W_S) \hspace{1cm} \text{(16)}$$

$$Id_L = HMAC_{K_C}(T_L) \oplus id_{ID} \hspace{1cm} \text{(17)}$$

$$K'_I = HMAC_{K_C}(Id_{Temp} \| AR \| W_S) \hspace{1cm} \text{(18)}$$

$$H_T = HMAC_{K'_I}(T_L) \hspace{1cm} \text{(19)}$$

Listing 1: Authorization phase between CS and LI

...
We analyze our proposed protocol in terms of performance, privacy and security.

6.1 LR’s Performance Analysis

In our scenario, LR is the most resource constrained device but holds all the sensitive information, hence we analyze its performance in the following areas:

- **Communication Cost:** Our proposed protocol uses simple primitives such as comparison, XoR and HMAC. Among the chosen primitives, HMAC is more resource demanding. Moreover, HMAC demands more resources than a normal Hash function but guarantees optimal security by reducing the number of collisions compared to a normal Hash function [17].

- **Storage Cost:** Storage is a critical issue in LR, hence our protocol requires only few space for storage as each device stores only minimal initial parameters - \( K_C, I_d \) and \( T_C \), amounting to 288 bits (36 bytes) for LR. During runtime, our protocol requires a maximum total of 736 bits (92 bytes) of storage in LR (corresponds to Line (18) of our protocol sequence).

- **Computational Cost:** Constrained devices expend a lot of energy in transmitting and receiving information [22, 25, 28]. A protocol with short messages or pseudo-random strings guarantees reduced energy consumption. Our protocol exchanges three messages during mutual authentication stage with a total of 992 bits (124 bytes).

6.2 Security and Privacy Analysis

We analyze our protocol against attack scenarios put forth in Section 4.2. This is a complementary analysis to AVISPA’s validation as it analyses the mechanisms of the exchanged information in details. Game 1: \( \beta \) masquerades as LI; Referring to Game 1 in Section 4.2, \( \beta \)'s objective is to send valid messages \( b_1 \) and \( b_3 \). That is, \( \beta \) can either try to crack the key \( K_L \) along with \( I_D_L \) or generate a valid message \( b_1 \) based on previously sniffed messages \( b_2 \) and \( b_3 \) during Phase 1.1.

For cracking key \( K_L \) along with \( I_D_L \), \( \beta \) is able to spoof LI. One of the solutions involves extracting values \( H_T \) and \( T_LI \) from a known message \( b_1 \) and try to crack \( K_L \). This assumes that the HMAC function is not robust to collision attacks, which is contrary to our assumptions of Section 4.3. Alternatively, \( \beta \) can combine messages \( b_1, b_2 \) and \( b_3 \) and try to derive valuable information. However, all messages behave like random or pseudo-random strings. Indeed, \( \epsilon_1 \) is randomized thanks to random \( R_1 \). Values \( \epsilon_1D, H_T, H_2 \) and \( b_3 \) are HMAC outputs and, as stated in [14], they behave as pseudo-random strings and evolve independently from each other as their inputs are different. As such, whatever the number of sniffed messages \( b_1, b_2 \) and \( b_3 \), it is not possible to extract any kind of information, and the game can not succeed.

Game 2: \( \beta \) tracks LR; Following Game 2 in Section 4.2, LR responds for session \( i \) with messages \( \epsilon_{1L} \) and \( \epsilon_{2L} \), which behave as random or pseudo-random strings. Such that, any response from LR can not succeed. Game 3: \( \beta \) depletes LR’s resources; Game 3 in Section 4.2 is a form of Denial of Service (DoS) attack such that an adversary \( \beta \) constantly queries LR to utilize its resources and deplete its energy source. Our protocol tests message \( b_1 \) using four comparison operations, four HMAC operations, and one Xor operation to verify validity. Of these operations, HMAC consumes more energy as explained in section 6.1.

Now, let us quantify the duration of time needed for \( \beta \) to deplete an alkaline long-life AAA battery with total energy of 5071 Joules [16]. If LR conforms to IEEE 802.15.4 [6] with an antenna frequency of 2.4 GHz band, data rate of 250 Kbps, and power consumption of 1.475W in receive mode [9], then it will take approximately 2 ms (milliseconds) to receive 480 bits of data sent in message \( b_1 \) by dissipating 1.475*0.002 = 0.003 Joules. According to [27], HMAC function consumes 1.16μJ (microjoules) per byte of data. Parameters used in calculations have a total of 960 bits or 120 bytes for \( H_T \) (Equation (16)), and \( ID_L \) (Equation (17)), \( K_L \) (Equation (18)), and \( H_T \) (Equation (19)). As such, each request from \( \beta \) costs 1.16*120 = 139.2μJ, which makes a total of 0.003 Joules for receiving and calculations. With this consumption rate, it will take 1,706,718 rounds to deplete the battery.

Suppose \( \beta \) sends message \( b_1 \) to LR every 1 second, it will take around 20 days to deplete the battery, with most of the energy being spent in receiving message \( b_1 \). Hence this game cannot succeed.
7 Conclusion

In this article we presented serverless lightweight mutual authentication protocol for small mobile resource computing devices. The protocol has been thoroughly explained, analyzed and its advantages outlined.

The originality of the protocol is based on the idea that a resource constrained device can mutually authenticate with another device without sharing any information. Our lightweight protocol uses simple primitives such as XoR, comparison and HMAC function, during mutual authentication and it also requires few storage space in each communicating party.

Thorough analysis was done to theoretically verify the security and performance properties of the protocol. The security properties have also been formally validated using AVISPA tool.

References

17. Thorough analysis was done to theoretically verify the security and performance properties of the protocol. The security properties have also been formally validated using AVISPA tool.

Annex 1: HLPSL code for Mutual Authentication Protocol

```
%ROLE ALICE
role alice ([A,S,B agent, K symmetric_key, Suc, H1 : hash_func, M, AR,Na, Ws, text : SND_SA, RCV_SA, SND_BA, RCV_BA : channel(dy)])
played_by A
def local : Na, Ri := text :
state := 0 /
init State := 0
end role

%ROLE BOB
role bob ([A, B, S agent, K symmetric_key, Suc, H1 : hash_func, M, AR,Na, Ws, text : SND_SA, RCV_SA, SND_BA, RCV_BA : channel(dy)])
played_by B
def local : Na, Ri := text :
state := 0 /
init State := 0
end role

%ROLE SERVER
role server ([A, B, S agent, K symmetric_key, text : SND_SA, RCV_SA, SND_BA, RCV_BA : channel(dy)])
played_by S
def local : Na, Ri := text :
state := 0 /
init State := 0
end role

end goal
```
