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
[1]{c.mtita, p.daragon}@traxens.com
[2]Institut Mines-Télécom/Télécom SudParis, CNRS UMR 5157 SAMOVAR, Évry, France
[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 AVISPA 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 [26], [7], [31] because constrained devices do not have enough capabilities to provide access rights to various external entities [3] 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 AVISPA 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 *LR*s and administers *LR*s.
2. Lightweight Initiator (*LI*): A terminal accessing information stored in *LR*s. *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. *LR*s 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 *LR*s within a small geographic region sharing the same secret key K_C that is used to verify authenticity of external entities interacting with *LR*s within a cluster. *LI* is an external device accessing information stored in *LR*s within a given cluster. Prior to accessing *LR*s in a cluster, *LI* must securely connect to *CS* and request authorization to access *LR*s within a given

cluster. *LI*'s request contains its geographic position, authenticating parameters and the identifier ID_L. 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* K_L, *access rights AR*, *Time Window* W_S and list of temporary identities L_j 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, *LI*s 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], [24]. However, connection-oriented model faces major limitations [32], 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 *Serverless search and authentication RFID 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 [30]. 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 improvement. 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) ***H**LR **O**ffline **A**uthentication*, 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 Tsudik [34]. Tsudik'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. Tsudik 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 Tsudik [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, Tsudik'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 relisient our protocol can be against attacks.
Game 1: β masquerades as LI

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

Game 2: β tracks LR_i

- **Phase 2.1**: β colludes with a legitimate device *LI* and listens to exchanges between *LI* and responder LR_1 and then between *LI* and LR_2.
- **Phase 2.2**: Challenger selects LR_i, $i \in \{1,2\}$, β listens to exchanges between *LI* and LR_i, and β sends a guess i value to the challenger.
  β wins the game if i is correct. The protocol is considered private if β cannot win the game with probability greater than 0.5.

Game 3: β depletes LR's resources

- **Phase 3.1**: β eavesdrops messages b_1 between *LI*s and *LR*s.
- **Phase 3.2**: β sends forged b_1 messages to a targeted *LR* within a cluster.
  β 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-**H**ash **M**essage **A**uthentication **C**ode* (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. ***P**seudo **R**andom **N**umber **G**enerator (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_L 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_C 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 bits parameter made from two timestamps, 32 bits start timestamp T_0 and 32 bits end timestamp T_Z parameters used to show K_L's validity.

Table 1: *Protocol notations with size estimations*

Parameter name	Symbol	Bits
LI's system Time	T_{LI}	32
LR's stored timestamp	T_C	32
Start Time Window	T_0	32
End Time Window	T_Z	32
Time Window	W_S	64
LI's Identifier	ID_L	128
LI's Key	K_L	128
LR's identifier	Id_i	128
LR's cluster Key	K_C	128
Derived session key	K_S	128
Timestamp signature	H_T	128
Random Value	R_1	128
Access Rights	AR	128
LI's secret code	H_K	128
List of *LR*s temporary identities	L_j	-

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_L (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.

$$LI \xrightarrow{a_1} CS \quad : \quad Request \tag{1}$$
$$CS \quad : \quad Get\ Time\ Window : W_S = [T_0, T_Z] \tag{2}$$
$$: \quad \forall\ LI\ generate\ Id_{Temp_i} = HMAC_{Id_i}(T_0) \tag{3}$$
$$: \quad Create\ list\ L_j = \{Id_{Temp_1}, Id_{Temp_2}, ..., Id_{Temp_i}\} \tag{4}$$
$$: \quad K_L = HMAC_{K_C}(ID_L||AR||W_S) \tag{5}$$
$$: \quad H_K = HMAC_{K_C}(W_S) \tag{6}$$
$$LI \xleftarrow{a_2} CS \quad : \quad K_L,\ H_K,\ W_S,\ AR,\ L_j \tag{7}$$
$$LI \quad : \quad Get\ K_L,\ H_K,\ W_S,\ AR,\ L_j \tag{8}$$

Listing 1: *Authorization phase between CS and LI*

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 e_{ID}, *timestamp* T_{LI}, *timestamp 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_{1_i} corresponding to each Id_{Temp_i} (cf. Line (12)). This step facilitates the search of H_{1_i} 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 4.4. *LI* aborts the current session if the timestamp parameters are not correct.

B1. Validation of Key K_L
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_L used by *LI*.

After verification of K_L, *LR* generates a temporary identity using its identifier Id and T_0 from W_S (cf. Line (24)) and also generates a signature H_1 for T_{LI} (cf. Line (25)). Likewise, *LR* generates a random R_1 (cf. Line (26)), which is used to calculate a cipher e_1 and a signature H_2 (cf. Lines (27) and (28)). Cipher e_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_i} values. Once found, R'_1 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 $H_R = HMAC_{Id_{Temp}}(R'_1)$. 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).

$$LI \quad : \quad Get\ current\ timestamp\ T_{LI} \tag{9}$$
$$: \quad e_{ID} = HMAC_{H_K}(T_{LI}) \oplus ID_L \tag{10}$$
$$: \quad H_T = HMAC_{K_L}(T_{LI}) \tag{11}$$
$$: \quad Calculate\ H_{1_i} = HMAC_{Id_{Temp_i}}(T_{LI})\ \forall\ Id_{Temp_i} \tag{12}$$
$$LR \xleftarrow{b_1} LI \quad : \quad H_T,\ e_{ID}\ , AR\ ,\ W_S,\ T_{LI} \tag{13}$$

$$LR \quad : \quad if\ T_Z < T_C\ or\ T_Z < T_{LI}\ or\ T_{LI} < T_C\ or\ T_0 < T_{LI} \tag{14}$$
$$END\ SESSION \tag{15}$$
$$: \quad H'_K = HMAC_{K_C}(W_S) \tag{16}$$
$$: \quad ID'_L = HMAC_{H'_K}(T_{LI}) \oplus e_{ID} \tag{17}$$
$$: \quad K'_L = HMAC_{K_C}(ID'_L||AR||W_S) \tag{18}$$
$$: \quad H'_T = HMAC_{K'_L}(T_{LI}) \tag{19}$$

$$: \quad if(H'_T == H_T) \quad \%\% \ LI's \ Key \ K_L \ is \ valid \tag{20}$$
$$T_C = T_{LI} \tag{21}$$
$$else \tag{22}$$
$$END \ SESSION \tag{23}$$
$$: \quad Calculate \ Id_{Temp} = HMAC_{Id}(T_0) \tag{24}$$
$$: \quad H_1 = HMAC_{Id_{Temp}}(T_{LI}) \tag{25}$$
$$: \quad Generate \ random \ R_1 \tag{26}$$
$$: \quad e_1 = R_1 \oplus Id_{Temp} \tag{27}$$
$$: \quad H_2 = HMAC_{Id_{Temp}}(R_1||K'_L) \tag{28}$$
$$LI \xleftarrow{b_2} LR \quad : \quad e_1, \ H_1, \ H_2 \tag{29}$$
$$LI \quad : \quad Search \ for \ H_1 \ in \ a \ list \ of \ H_{1_i} \tag{30}$$
$$: \quad R'_1 = e_1 \oplus Id_{Temp} \tag{31}$$
$$: \quad H'_2 = HMAC_{Id_{Temp}}(R'_1||K_L) \tag{32}$$
$$: \quad if(H'_2 \neq H_2) \tag{33}$$
$$END \ SESSION \tag{34}$$
$$: \quad else \quad \%\% \ AR \ is \ authentic \tag{35}$$
$$H_R = HMAC_{Id_{Temp}}(R'_1) \tag{36}$$
$$K_S = HMAC_{Id_{Temp}}(R'_1||T_{LI}) \tag{37}$$
$$LR \xleftarrow{b_3} LI \quad : \quad H_R \tag{38}$$
$$LR \quad : \quad H'_R = HMAC_{Id_{Temp}}(R_1) \tag{39}$$
$$: \quad if(H'_R == H_R) \quad \%\% \ LI \ is \ authentic \tag{40}$$
$$K_S = HMAC_{Id_{Temp}}(R_1||T_{LI}) \tag{41}$$

Listing 2: *Mutual authentication phase between LI and LR*

5 Protocol Validation

We used *Automated Validation of Internet Security Protocols and Applications* ***(AVISPA)*** [4] tool to validate our protocol. AVISPA's current version integrates four back-ends, *On-the-fly Model-Checker* ***(OFMC)***, *Constraint-Logic-based Attack Searcher* ***(CL-AtSe)***, *SAT-based Model-Checker* ***(SATMC)***, and *Tree Automata based on Automatic Approximations for the Analysis of Security Protocols* ***(TA4SP)***. The attacker model is *Dolev-Yao attacker* [12] i.e an attacker has full capabilities over the newtork and can listen or intercept communication, inject new messages or modify messages in transit. The *High Level Protocol Specification Language* (HLPSL) [35] code for our protocol used in AVISPA is presented in *Annex 1*.

In Listing 3, AVISPA outputs *SAFE* from three back-ends, *OFMC*, *CL-AtSe*, and *SATMC*. This implies that AVISPA could not reproduce any attack on our proposed protocol. AVISPA gives *INCONCLUSIVE* result in TA4SP back-end because TA4SP does not perform verification. Nevertheless, our protocol is still safe.

```
OFMC      : SAFE
CL−AtSe   : SAFE
SATMC     : SAFE
TA4SP     : INCONCLUSIVE
```

Listing 3: *Validation results after running our protocol in AVISPA*

6 Analysis of the Proposed Solution

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:

Computational 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, Id 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).
Communication Cost: Constrained devices expend a lot of energy in transmitting and receiving information [22], [25], [28]. A protocol with fewer and shorter messages 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 analyse 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: *β masquerades as LI*; Referring to Game 1 in Section 4.2, β's objective is to send valid messages b_1 and b_3. That is, β can either try to crack the key K_L along with ID_L or generate a valid message b_3 based on previously sniffed messages b_2 and b_3 during Phase 1.1.

For cracking key K_L along with ID_L, β 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. Alternativey, β can combine messages b_1, b_2 and b_3 and try to deduce valuable information. However, all messages behave like random or pseudo-random strings. Indeed, e_1 is randomized thanks to random R_1. Values e_{ID}, 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*: β tracks LR*; Following *Game* 2 in Section 4.2, LR_i responds for session j with messages e_{1ij} and e_{2ij} which behave as random or pseudo-random strings. Such that, any response from LR_1 is semantically indistinguishable from responses of LR_2, and previous responses of LR_1. Hence, an adversary β is unable to guess with a probability greater than 0.5 which LR_i sent message *b2*.
Game 3*: β depletes LR's resources*; *Game* 3 in Section 4.2 is a form of *Denial of Service (DoS)* attack such that an adversary β 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 β to deplete an alkaine 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'_K (Equation (16)), and ID'_L (Equation (17)), K'_L (Equation (18)), and H'_T (Equation (19)). As such, each request from β 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 β 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

1. Sarah Abughazalah et al. A mutual authentication protocol for low-cost rfid tags formally verified using casperfdr and avispa. In *Information Science and Technology (ICIST), 2013 International Conference on*, pages 44–51. IEEE, 2013.
2. Sheikh Ahamed et al. Erap: Ecc based rfid authentication protocol. In *Future Trends of Distributed Computing Systems, 2008. FTDCS'08. 12th IEEE International Workshop on*, pages 219–225. IEEE, 2008.
3. Moshaddique Al Ameen et al. Security and privacy issues in wireless sensor networks for healthcare applications. *Journal of medical systems*, 36(1):93–101, 2012.
4. Alessandro Armando et al. The avispa tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification*, pages 281–285. Springer, 2005.
5. Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.
6. Bruno Bougard et al. Energy efficiency of the ieee 802.15. 4 standard in dense wireless microsensor networks: Modeling and improvement perspectives. In *Design, Automation, and Test in Europe*, pages 221–234. Springer, 2008.
7. David Carman et al. Constraints and approaches for distributed sensor network security (final). *DARPA Project report,(Cryptographic Technologies Group, Trusted Information System, NAI Labs)*, 1:1, 2000.
8. Christy Chatmon et al. Secure anonymous rfid authentication protocols. *Florida State University, Department of Computer Science, Tech. Rep*, 2006.
9. Jyh-Cheng Chen et al. A comparison of mac protocols for wireless local networks based on battery power consumption. In *INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 150–157. IEEE, 1998.
10. Tien-Ho Chen et al. A robust mutual authentication protocol for wireless sensor networks. *Etri Journal*, 32(5):704–712, 2010.
11. Miaolei Deng et al. Weakness in a serverless authentication protocol for radio frequency identification. In *Mechatronics and Automatic Control Systems*, pages 1055–1061. Springer, 2014.
12. Danny Dolev et al. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.
13. Donald Eastlake and Paul Jones. Us secure hash algorithm 1 (sha1), 2001.
14. Pierre-Alain Fouque et al. Hmac is a randomness extractor and applications to tls. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, pages 21–32. ACM, 2008.
15. Md Endadul Hoque et al. Enhancing privacy and security of rfid system with serverless authentication and search protocols in pervasive environments. *Wireless personal communications*, 55(1):65–79, 2010.
16. http://www.allaboutbatteries.com/Energy tables.html. Battery types, energy and consumptions, June 2011.
17. Marcio Juliato et al. An efficient fault-tolerance technique for the keyed-hash message authentication code. In *Aerospace Conference, 2010 IEEE*, pages 1–17. IEEE, 2010.
18. Hermann Kopetz. Internet of things. In *Real-time systems*, pages 307–323. Springer, 2011.
19. Chin-Feng Lee et al. Server-less rfid authentication and searching protocol with enhanced security. *International Journal of Communication Systems*, 25(3):376–385, 2012.
20. Mingyan Li et al. Multi-domain rfid access control using asymmetric key based tag-reader mutual authentication. In *ICAS2008—Proceedings of the 26th international Congress of the Aeronautical Sciences*, 2008.
21. Luon-Chang Lin et al. Lightweight and serverless rfid authentication and search protocol. In *2009 Second International Conference on Computer and Electrical Engineering*, volume 2, pages 95–99, 2009.
22. Gang Lu et al. Performance evaluation of the ieee 802.15. 4 mac for low-rate low-power wireless networks. In *Performance, Computing, and Communications, 2004 IEEE International Conference on*, pages 701–706. IEEE, 2004.
23. Parikshit Mahalle et al. Identity establishment and capability based access control (iecac) scheme for internet of things. In *Wireless Personal Multimedia Communications (WPMC), 2012 15th International Symposium on*, pages 187–191. IEEE, 2012.
24. Bruce Ndibanje et al. Security analysis and improvements of authentication and access control in the internet of things. *Sensors*, 14(8):14786–14805, 2014.
25. Trevor Pering et al. Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 220–232. ACM, 2006.
26. Selwyn Piramuthu. Rfid mutual authentication protocols. *Decision Support Systems*, 50(2):387–393, 2011.
27. Nachiketh R Potlapally et al. Analyzing the energy consumption of security protocols. In *Proceedings of the 2003 international symposium on Low power electronics and design*, pages 30–35. ACM, 2003.
28. Gregory Pottie et al. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.
29. Mohsen Pourpouneh et al. An improvement over a server-less rfid authentication protocol. *International Journal of Computer Network and Information Security (IJCNIS)*, 7(1):31, 2014.
30. Masoumeh Safkhani et al. On the security of tan et al. serverless rfid authentication and search protocols. In *Radio Frequency Identification. Security and Privacy Issues*, pages 1–19. Springer, 2013.
31. Chiu C Tan et al. Body sensor network security: an identity-based cryptography approach. In *Proceedings of the first ACM conference on Wireless network security*, pages 148–153. ACM, 2008.
32. Chiu Chiang Tan et al. Severless search and authentication protocols for rfid. In *Pervasive Computing and Communications, 2007. PerCom'07. Fifth Annual IEEE International Conference on*, pages 3–12. IEEE, 2007.
33. Caimu Tang et al. An efficient mobile authentication scheme for wireless networks. *Wireless Communications, IEEE Transactions on*, 7(4):1408–1416, 2008.
34. Gene Tsudik. Ya-trap: Yet another trivial rfid authentication protocol. In *Pervasive Computing and Communications Workshops, 2006. PerCom Workshops 2006. Fourth Annual IEEE International Conference on*, pages 4–pp. IEEE, 2006.
35. David von Oheimb. The high-level protocol specification language hlpsl developed in the eu project avispa. In *Proceedings of APPSEM 2005 Workshop*, 2005.
36. John Walters et al. Wireless sensor network security: A survey. *Security in distributed, grid, mobile, and pervasive computing*, 1:367, 2007.

Annex 1: *HLPSL code for Mutual Authentication Protocol*

```
%%ROLE ALICE
role alice ( A, S, B: agent, K: symmetric_key, Succ, H1 : hash_func, M, AR, Na1, Ws : text, SND_SA, RCV_SA,
        SND_BA, RCV_BA : channel(dy))
played_by A
def=
      local   Lj, Na, R1 : text, State : nat, Idr, Ks, Kd, Hk : symmetric_key, H : hash_func
      const id  : text,
      alice_bob_kd, ks, bob_alice_kd, kd : protocol_id
       init   State := 0
 transition
   1. State    = 0 /\ RCV_SA(start) =|> State' := 2 /\ SND_SA(A.B.id.M)
   2. State    = 2 /\ RCV_SA(A.B.AR'.Ws'.Na1'.{Hk'.Lj'.Kd'}_K) =|>
       State'  := 4 /\ Na' := new() /\ SND_BA(A.B.{id}_Hk'.AR'.Ws'.Na'.Na1') /\ request(A, B, bob_alice_kd, Kd)
   3. State    = 4 /\ RCV_BA(A.B.{R1'}_Idr.H1(Kd'.R1').H1(Na)) =|>
       State':= 6/\SND_BA(A.B.H1(R1'))/\ Ks' := H(Na.R1')/\ witness(A, B, bob_alice_kd, H1)/\ secret(Ks, ks, {A, B})
end role
%%ROLE SERVER
role server (A, B, S  : agent, K, Kc  : symmetric_key, M : text, SND_AS, RCV_AS : channel(dy))
played_by S
def=
      local   State : nat, Na1, Lj, AR, Ws : text, Hk, Kd : symmetric_key, Hash : hash_func
       init State := 1
 transition
          1. State = 1 /\ RCV_AS(A.{id.M}_K) =|>
             State':=  3 /\ Ws' := new() /\ AR' := new() /\ Kd' := Hash(Kc.id.AR'.Ws') /\ Lj' := new()
                         /\ Hk' := new()/\ Na1':= new() /\ SND_AS(A.AR'.Ws'.Na1'.{Hk'.Lj'.Kd'}_K)
          /\ secret(Kd', bob_alice_kd, {A,S,B})
end role
%%ROLE BOB
role bob( A, B : agent, Succ, H1 : hash_func, AR, Na1, Ws : text, Kc  : symmetric_key, SND_AB, RCV_AB :
        channel(dy))
played_by B
def=
      local   State : nat,  Idr, Hk, Ks, Kd : symmetric_key, H : hash_func, Na, R1 : text
          init State := 5
 transition
      1. State = 5 /\ RCV_AB(A.B.{id}_Hk.AR'.Ws'.Na'.Na1') =|>
         State' := 7 /\ R1' := new() /\ Kd' := H(Kc.id.AR.Ws) /\ Idr' := H(id.Na1') /\
                SND_AB(A.B.{R1'}_Idr.H1(Kd'.R1').H1(Na))
                      /\ witness(B, A, bob_alice_kd, Kd') /\ request(B, A, alice_bob_kd , H1)
      2. State = 7  /\ RCV_AB(A.B.H1(R1')) =|> State' := 9 /\ Ks' := H(Na.R1') /\ secret(Ks', ks, {A,B})
end role
%%ROLE SESSION
role session ( A, S, B : agent, M, AR, Na1, Ws : text, Succ, H : hash_func, K, Kc: symmetric_key)
def=
  local  SSA, RSA, SBA, RBA, SAS, RAS, SAB, RAB : channel(dy)
composition
      alice (A, S, B, K, Succ, H, M, AR, Na1, Ws, SSA, RSA, SBA, RBA) /\server(A, B, S, K, Kc, M, SAS, RAS)
              /\bob(A, B, Succ, H, AR, Na1, Ws, Kc, SAB, RAB)
end role
role environment()
def=
      const a, s, b: agent, ksi, kc, k, ka, ki : symmetric_key, succ, h: hash_func, id, m, ar, na1, ws : text,
              bob_alice_r1, alice_bob_na: protocol_id
intruder_knowledge = {a, b, s, ki, ws, ar, na1, m, succ, ksi}
composition
      session(a, s, b, m, ar, na1, ws, succ, h,  k, kc) /\ session(i, s, b, m, ar, na1, ws, succ, h, ki, kc)
end role
      goal
       %secrecy of shared keys
          secrecy_of ks, kd
          authentication_on bob_alice_kd
          authentication_on alice_bob_na
   end goal
environment()
```