IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 19, 2021, accepted September 30, 2021, date of publication October 4, 2021, date of current version October 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3117444

T2T-MAP: A PUF-Based Thing-to-Thing Mutual Authentication Protocol for IoT

KARIM LOUNIS AND MOHAMMAD ZULKERNINE, (Senior Member, IEEE)
Queen's Reliable Software Technology Lab, School of Computing, Queen's University, Kingston, ON K7L 3N6, Canada
Corresponding author: Karim Lounis (karim.lounis@queensu.ca)

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and in part by Canada Research Chairs (CRC) Program.

ABSTRACT As security has always been an afterthought of innovation, the security of IoT (Internet of Things), in general, and authentication, in particular, has become a serious research challenge. Although many authentication protocols have been proposed in the literature during the past decade, most of them do not fulfill the IoT security and performance requirements. Furthermore, only a very small number of these protocols can be used in Thing-to-Thing (T2T) architectures, where Things autonomously authenticate each other without involving any human intervention. In this paper, we propose a novel lightweight T2T mutual authentication protocol (T2T-MAP) using PUFs (Physical Unclonable Functions). The protocol employs PUFs technology to allow each Thing to uniquely identify and authenticate itself in an IoT infrastructure by using the physical randomness of its circuitry. We design the protocol and perform a security analysis to show that it is secure against known attacks. Also, we prove the security of the protocol using a security protocol prover. Finally, we implement a prototype of the protocol on resource-constrained devices and then conduct a performance analysis to demonstrate that the protocol allows fast authentication, reasonable communication overhead, and low energy consumption.

INDEX TERMS PUFs, IoT security, Thing-to-Thing authentication, PUF-based authentication.

I. INTRODUCTION

IoT (Internet of Things) allows the interconnection of an exponentially increasing number of heterogeneous devices, called Things. This interconnection expands the classical network of the Internet from a Machine-to-Machine (M2M) communication system to a Thing-to-Machine (T2M) and Thing-to-Thing (T2T) communication system. This has allowed the emergence of various smart applications and new ubiquitous Internet services in various fields. However, this important evolution of the Internet to a heterogeneous infrastructure has unfortunately invited cybercriminals to exploit the new infrastructure and mount distributed and devastating cyberattacks at large-scale as well as at local-scale [1]–[7]. These cyberattacks have demonstrated to the community how catastrophic and diversified cybercrimes could be in a world where everything is perceived as a connected computer [8]. Interestingly, most of these cyberattacks, if not all, are due to security vulnerabilities in the adopted authentication protocols. In fact, as the service of authentication constitutes the spine of all security properties,[1] any weakness residing on the adopted authentication mechanism will turn the system into a brightening target for cybercriminals to conduct cyberattacks. Moreover, it is important to note that in this new type of heterogeneous networks, the impact of cyberattacks is not limited to information being stolen as in classical systems but may also include the loss of human lives [8]. Therefore, more research is required to develop new authentication protocols that are secure and reliable by-design and conform to IoT security and performance requirements.

The challenge of designing and developing secure and reliable authentication protocols for IoT has become mature and well-known to the research community. Many research works have been conducted for this purpose. In fact, ten years after the emergence of IoT (i.e., early 2010), a large number of authentication protocols, known as lightweight protocols (sometimes ultra-lightweight), were designed and

The associate editor coordinating the review of this manuscript and approving it for publication was Fung Po Tso.

[1]In most, if not all, security mechanisms, and during the execution of the authentication protocol, cryptographic keys are generally established and derived to provide confidentiality, data integrity, non-repudiation, and availability. A vulnerable authentication protocol would result in security breaches w.r.t. the confidentiality, data integrity, and availability of a system.

proposed for various IoT wired or wireless infrastructures (e.g., Cloud, VANETs, MANETs, and WSNs [74]), and IoT applications (e.g., smart healthcare, home automation, and smart transportation) [67]. These protocols relied on the application of classical cryptographic mechanisms, where some of them adopted ECC (Elliptic Curve Cryptography), while others employed lightweight cryptographic operations, such as simple hash functions and bitwise logical operators. However, these protocols suffer from some known vulnerabilities that are related to the incorrect usage of classical cryptographic mechanisms, which has resulted in the re-occurrence of known cyberattacks but with an IoT flavor. In addition, as IoT aims to transform everything into a connected computer, Things are most likely going to be autonomously communicating with each other without any human intervention. This form of communication, commonly known as T2T, constitutes a fundamental communication scheme that future IoT communications tend to adopt. Unfortunately, we have observed that only a few authentication protocols were proposed in the literature for this particular communication scheme (i.e., T2T). The need for developing T2T authentication protocols that conform to IoT security and performance requirements has seriously increased.

There has been a noticeable convergence from the research community to develop lightweight and secure-by-design authentication protocols for IoT using PUFs (Physical Unclonable Functions) [11]–[25]. In fact, PUFs are physical one-way functions constructed from the unique nanoscopic-structure of physical objects (e.g., integrated circuits, crystals, magnets, lens, solar cells, or papers) and their reaction to random events. This intrinsic uniqueness in the structure and reaction is due to the idiosyncrasies in the manufacturing process of the objects. It allows not only the unique identification of an object but also its authentication. Furthermore, it is assumed to be impossible to clone the PUF of an object (and hence the object itself), which somehow can be perceived as a security-by-design that will prevent any possible impersonation and cloning attacks. Thus, PUFs are considered as a reliable and prominent physical security technology to develop lightweight authentication protocols for IoT. Moreover, various semiconductor companies have already adopted PUFs for securing their integrated circuits [58], [59]. Although PUFs are believed to have security-by-design, the feasibility of some attacks based on machine-learning [28]–[36] and side-channel [36]–[43] has been demonstrated.

In this paper, we adopt PUF technology to design and implement a T2T lightweight mutual authentication protocol that has security-by-design for IoT. For short referencing, we attribute the name "T2T-MAP" (Thing-to-Thing Mutual Authentication Protocol) for the protocol. We first design the protocol and perform a security analysis to demonstrate the resilience of the protocol against known attacks. Also, we use a well-established security protocol verifier to automatically prove some security properties of the protocol. Then, we implement the protocol on low-cost devices, concretely, on Arduino development boards. We show that the proposed protocol performs efficiently in terms of execution time, communication overhead, and energy consumption.

Compared to the existing PUF-based authentication protocols [11]–[25], the proposed authentication protocol is secure against the attacks that were reported on those protocols as well as to known attacks [64]. Moreover, the proposed protocol adopts a new concept of extended challenge-response pairs (eCRPs) to obfuscate the classical structure of CRPs (cf. Section IV-A) and make the system resilient to CRPs disclosure and malicious insider attacks. In addition to that, the protocol uses the cryptographic concept of distributed value [9] to allow the transmission of PUF response in a secure way that prevents attackers from conducting modeling attacks. Furthermore, the protocol imposes to each authenticating party to use its embedded PUF circuit to establish mutual authentication, which enforces the service of non-repudiation. Last but not least, the proposed protocol is purely lightweight (from a cryptographic perspective) by applying simple hash functions and logical bitwise operators.

To summarize, the main contributions of this paper are as follows:

1) We design T2T-MAP, a PUF-based mutual authentication protocol for IoT. We adopt the concept of eCRPs to make the protocol secure against CRPs disclosure, malicious insider, and modeling attacks.
2) We analyze the security of the protocol and discuss the resilience of the protocol to some known attacks. Moreover, we use Tamarin [65] to automatically prove the security of the protocol.
3) We implement T2T-MAP on resource-constrained devices and evaluate its performance with respect to the execution time, communication overhead, and energy consumption for different configurations.

The remainder of the paper is organized as follows. In Section II, we provide a brief overview of PUFs and the concept of PUF-based authentication. In Section III, we review some recent PUF-based authentication protocols. In Section IV, we design a T2T PUF-based authentication protocol for IoT. Section V discusses the proposed protocol properties and analyzes its security against attacks. Also, a formal security analysis using Tamarin is conducted in the same section. In Section VI, we implement the protocol and conduct a performance analysis on the protocol. We conclude the paper in Section VII.

II. PUFs (PHYSICAL UNCLONABLE FUNCTIONS)

PUFs (Physical Unclonable Functions)[2] are physical one-way functions constructed from the unique nanoscopic-structure of physical objects (e.g., integrated circuits, crystals, magnets, lens, or solar cells) and their reaction to random events. It allows not only the unique identification of an object but also its authentication. In fact, when a given PUF is excited with a random event, called challenge, the function

[2]PUFs (Physical Unclonable Functions) are also known as POWFs (Physical One-Way Functions) [26] or PRFs (Physical Random Functions) [27].

returns a unique, unpredictable, and reproducible response. The set of all possible challenges and their corresponding responses is often referred to as the CRPs (Challenge-Response Pairs).

Since the emergence of PUFs, researchers have worked on designing and implementing PUFs by exploiting the physical characteristics of different materials, including silicon, crystals, magnets, lens, solar cells, and papers. This has resulted in a large variety of PUFs, including but not limited to, delay-based PUFs (e.g., Arbiter-PUFs [27], ring oscillator-PUFs [60], and glitch-PUFs [45]), memory-based PUFs (e.g., SRAM-PUFs [46], DRAM-PUFs [47], SR Latch PUF [57], and Rowhammer-PUFs [48]),[3] acoustical-PUFs [50], coating-PUFs [49], optical-PUFs (e.g., paper-PUFs [53], Compact Disc-PUF [52], and liquid crystal-PUF [54]),[4] and magnetic-PUFs [51]. These PUFs differ from each other in their environmental application, source of randomness (e.g., semiconductors, lens, crystals, or magnets), excitation mechanisms (e.g., electronic signals, beam of light or laser, or electromagnetic waves), or other parameters. Interested readers are refereed to the work of McGrath *et al.*, [44], where a comprehensive taxonomy of existing PUFs was extensively elaborated.

Among the recently explored security applications of PUFs, the development of on-the-fly and lightweight authentication protocols. These protocols would provide security for resource-constrained devices without having the devices to store any credentials on their limited non-volatile memories. This for example makes them resilient against physical invasive attacks [61]–[63].

To exploit the advantages of PUFs for integrated circuit authentication in general, and device authentication in particular, a typical challenge-response-based protocol is adopted [27]. The protocol consists of two main phases, the registration phase (a.k.a., enrolment phase) and the verification phase (a.k.a., authentication phase). During the registration phase, a device is enrolled in a database (a trust center) by registering pairs of challenges and responses (CRPs), generated from the PUF that is embedded in the device's circuit. Therefore, each registered device d_{id} will have its proper set of challenge and response pairs in the database $\Gamma_{id} = \{(c_0, r_0), \ldots, (c_n, r_n)\}$.

The authentication of a registered device d_p (a.k.a., prover) by another device d_v (a.k.a., verifier) consists of having device d_v randomly select one CRP, e.g., $(c_i, r_i) \in \Gamma_p$, and interrogate the device d_p by sending the challenge value c_i and obtaining the corresponding response r_i'. Once the response r_i' is received, d_v compares it with the registered response r_i. If both responses are identical (or closely identical), i.e., $r_i' \simeq r_i$, the device d_p is authenticated, otherwise, it is rejected.

[3]Delay-based and memory-based PUFs are often classified as electronic CMOS (Complementary Metal Oxide Semiconductor)-PUFs.

[4]In some literature, coating-based and optical-based PUFs are often classified as MEMS (Micro-Electro-Mechanical Systems)-based PUFs.

III. RELATED WORK

There has been a considerable number of research work aiming to develop PUF-based authentication protocols for IoT (Internet of Things) applications. These protocols apply different types of PUFs and aim to provide Things with a secure-by-design authentication mechanism that is resilient to classical attacks as well as to attacks that are particularly related to PUFs. In the following paragraphs, we briefly review some recent PUF-based authentication protocols. Also, the detail of the attacks that we mention for each related work is discussed in [64] and [71].

TABLE 1 summarizes the difference between the reviewed PUF-based authentication protocols. The protocols are compared using six characteristics (Column 2 to 7): (1) Used PUF, shows the type of PUF being used for the implementation of the authentication protocol referred in Column 1. (2) Authentication scheme, indicates the considered architecture to be either a Thing-to-Thing, in the sense resource-constrained to resource-constrained device, or a Thing-to-Machine, in the sense resource-constrained device to a resourceful device architecture. (3) Properties C1, C2, C3, and C4, mean whether the referred protocol uses PUFs on both parties, is lightweight from a cryptographic perspective, scalable, and allows the establishment of cryptographic keys, respectively. (4) Implementation platform specifies the type of hardware used to implement the PUF and the communicating parties. (5) Evaluation, in terms of security C5, performance C6, and circuit size C7. (6) Possible attacks indicate the different attacks that can be generated on the referred protocol as demonstrated in [64] and [71].

In [11], Boyapally *et al.*, proposed a PUF-based authentication protocol for smart meters in the context of smart grid applications. The protocol allows a smart meter to be authenticated to a server using PUFs. It adopts the DAPUF (Double Arbiter-PUF) [10] augmented with a linear feedback shift register (LSFR) module. They discussed the security of the protocol and showed that it is resilient to man-in-the middle attacks as well as to PUF-modeling attacks. Nevertheless, the security of this protocol can be easily breached and the server can be impersonated. We showed in [76] how the security of the protocol can be broken. In [12], Bansal *et al.*, proposed a PUF-based authentication protocol with key establishment for vehicles-smartgrid infrastructures. A formal security analysis was performed using Mao-Boyd Logic to prove the security of their protocol. However, the protocol was shown to be vulnerable to spoofing and message forging attacks. Qureshi and Munir [13] proposed a PUF-based identity-preserving authentication protocol for T2M authentication scheme. The protocol uses shuffling techniques to store obscured and uncorrelated information about the registered devices' PUFs instead of storing plaintext CRPs (Challenge-Response Pairs). The authors showed that the protocol is reliable and resilient to machine-learning attacks. Unfortunately, the protocol was shown to be vulnerable to replay and DoS attacks. Also, the protocol has a security single point of failure that allows the entire system to be compromised. Yanambaka *et al.* [14] proposed an FPGA PUF-based authentication protocol for medical IoT devices. It applies an obfuscation technique to perform authentication without CRPs disclosure. Notwithstanding, the security of the protocol was not assessed. The protocol contains vulnerabilities that can be exploited to generate attacks, such as machine-learning, CRPs disclosure, and replay attacks.

In [15], Nozaki and Yoshikawa, proposed a XOR-arbiter PUF-based one-way authentication protocol that is resistant to machine-learning attacks. It adopts the notion of distributed value [9] to obfuscate the PUF responses when being challenged and allow provers to store partial information about the CRPs of others. Also, it reduces the prediction rate for a successful machine-learning attack when a PUF is challenged. However, no security evaluation was conducted. The protocol is vulnerable to spoofing and CRPs disclosure attacks. In [16], Chatterjee *et al.*, proposed a PUF-based authentication protocol for IoT. The protocol allows an IoT device to get authenticated to a verifier by using its proper PUF and applying elliptic curve operations, pairing operations, and a hash function. A prototype of the protocol was implemented for a video surveillance application and an informal security analysis was performed. Nevertheless, the protocol is vulnerable to DoS and impersonation attacks. In [17], Liang *et al.*, developed a PUF-based authentication protocol for RFID systems in the context of IoT. The protocol allows an RFID-tag to be authenticated to a back-end server. The authors used BAN (Burrows-Abadi-Needham) logic to prove the correctness of the protocol and performed an informal security analysis of the protocol with respect to modeling attacks, man-in-the-middle attack, replay attack, CRPs exposure, and spoofing attacks. A performance analysis was also performed. However, the protocol was found to be vulnerable to de-synchronization, CRPs disclosure, and spoofing attacks. Kim *et al.* [18] proposed a PUF-based authentication protocol for device-server authentication. Nonetheless, the security of the protocol was not evaluated. Also, it was shown that the protocol is vulnerable to CRPs disclosure, key disclosure, and DoS attacks. In [19], Mahalat *et al.*, proposed a PUF-based authentication protocol for secure Wi-Fi authentication of IoT devices. The protocol is adapted for a T2M authentication scheme. Notwithstanding, similar to other protocols, it has been demonstrated to be vulnerable to spoofing, CRP disclosure, and DoS attacks. In [20], Mughal *et al.*, proposed a PUF-based authentication protocol, called PAS (PUF-based Authentication Scheme), for smart devices in IoT smart home applications. The protocol allows users, equipped with smartphones, to be authenticated to a gateway so that they can send command-messages to connected smart devices (e.g., a lightbulb or thermostat). Notwithstanding, the protocol is vulnerable to message forging, key disclosure, and DoS attacks.

Barbareschi *et al.* [21] proposed the PHEMAP (PUF-based Mutual Authentication Protocol), an SRAM PUF-based mutual authentication protocol for T2T authentication schemes. The protocol strongly relies on synchronization to operate correctly. For that reason, it has been shown to be vulnerable to desynchronization and DoS attacks. In [22], Yilmaz *et al.* proposed an arbiter PUF-based authentication protocol for T2M authentication scheme. Based on the use of a neural network model of the PUF, the protocol allows a verifier to authenticate a prover without having to store the CRPs on the verifier's memory. The protocol makes use of a device MAC address and a timestamp to obfuscate the PUF response by applying the RC5 cipher. However, the security of the protocol was not evaluated. Also, the protocol is vulnerable to insider spoofing attacks, CRP disclosure, and RC5 cracking. Feng *et al.*, [23] proposed a PUF-based lightweight attestation and authentication protocol, called AAoT, for IoT and cyber-physical systems. The protocol uses PUFs along with fuzzy extractors to derive a secret key that is used to establish mutual authentication between devices. However, the security of the protocol was neither discussed nor assessed. The protocol was proven to be vulnerable to spoofing, CRP disclosure, and replay attacks. Idriss and Bayoumi [24] proposed a PUF-based authentication protocol with a key exchange for resource-constrained systems, such as RFID systems. The protocol relies on a challenge-challenge scheme to establish authentication. The authors analyzed the security of the protocol against guessing and challenge collection attacks. Nevertheless, the protocol was pointed out to be insecure against spoofing, man-in-the-middle, and CRP disclosure. Clupek and Zeman [25] proposed a PUF-based mutual lightweight authentication protocol for T2T authentication schemes. It uses a third trusty party (TTP) as a trusted authority to establish a robust authentication between Things. It relies on the use of low-cost PUFs, simple hash functions, and the binary eXclusive-OR operator. The authors have claimed that the protocol is secure and provides security services. However, it was demonstrated that the protocol is vulnerable to certain secret disclosure, message forging, and modeling attacks.

Some more recent work have proposed PUF-based authentication solutions to solve the PUF-related challenge of not exposing the CRPs during authentication or storage (a.k.a., AA protocols). For example, Chatterjee *et al.*, [72] proposed 3PAA, a private PUF-based protocol for anonymous authentication. The protocol relies on the concept of zero-knowledge and elliptic curve cryptography (ECC) to allow k-times anonymous authentication of users to an application provider (AP) through a trusted party (group manager, or GM) without revealing the CRPs. This makes the system more resilient to PUF modeling attacks. The protocol was implemented and the security of the protocol was formally verified. A performance evaluation was also conducted. However, the protocol does not enforce mutual authentication between a user and an application provider. The protocol only allows the application provider to verify the authenticity of the user, whereas the latter has no ability to verify that it is communicating with the legitimate application provider and not with an attacker. Hence, if the AP is compromised, it may allow a particular user to authenticate and use the service more than k times. In T2T-MAP, the mutual authentication and non-repudiation are guaranteed using PUFs. That is to say, each party can verify that it is communicating with the legitimate counterpart as we discuss in Section V. Additionally, the application provider is vulnerable to a spoofing attack. In fact, an attacker can impersonate the application provider and run the tracing phase to remove the access log entry of a victim user from the GM. To that end, the attacker intercepts an authentication session between the victim user and the application provider and use the messages to initiate the tracing procedure. In this procedure, the attacker does not have to authenticate itself to the GM as the legitimate application provider. The GM just checks the value of the parameter that the attacker obtains after intercepting the authentication session. As a consequence to this attack, the GM removes the access log entry of the application provider for the user, which would cause inconsistencies in the system.

Braeken [73] proposed another method to provide authentication between two IoT devices using PUFs and a common trusted cluster node, where the CRPs are not exposed. Nevertheless, in the context of our paper, this research work as well as the work in [72] are not lightweight from the cryptographic perspective as they adopt elliptic curve cryptography (ECC). Moreover, their protocols' performance evaluation is conducted using non-resource constrained hardware, such as the Intel i5 processors, which we believe cannot be used to conclude that a protocol would provide similar performance on resource-constrained devices such as the ones we consider in this paper (i.e., Arduino boards).

IV. THE PROPOSED PROTOCOL (T2T-MAP)

In this section, we propose T2T-MAP, a secure lightweight T2T PUF-based authentication protocol for IoT (Internet of Things). It allows two Things to mutually authenticate each other through a central gateway. We start by introducing the concept of eCRPs (extended CRPs) that T2T-MAP uses for authentication. Then, we present the phases of T2T-MAP for its establishment. Next, we discuss the security and performance properties and analyze the security of T2T-MAP against known attacks. TABLE 2 reports a summary of the adopted notation in the T2T-MAP authentication protocol.

A. THE CONCEPT OF EXTENDED CRPs (eCRPs)

From the related work, we have learned that if the CRPs (Challenge-Response Pairs) of registered Things are stored on a verifier in plaintext, they are subject to physical invasion attacks where they can be disclosed. Also, if for each registered Thing a verifier stores a large set of CRPs, the whole system becomes less efficient and not scalable as Things have limited storage capacity. Therefore, to thwart these security and performance issues, we have adopted the following two approaches:

1) FIRST APPROACH

We have transformed the classical CRP structure from a 2-tuple (c, r) to a 5-tuple $(x_1, x_2, x_3, \alpha_1, \alpha_2)$, which we have called the extended CRP, or eCRP for short. We denote the eCRP by the letter Γ. For example, if a Thing T_i stores an eCRP about Thing T_j, we write $\Gamma_i(T_j)$. The content of an eCRP $\Gamma_i(T_j)$ consists of three challenge values, x_1^i, x_2^i, and x_3^i, and two obfuscated response values α_1^{ij} and α_2^{ij}. These obfuscated response values are computed using the challenges and the PUF functions of both involved Things as expressed in the following two equations, where $\Psi_i(\cdot)$ and $\Psi_j(\cdot)$ are the PUF functions of Thing T_i and T_j, respectively.

$$\alpha_1^{ij} = \Psi_j(\Psi_i(x_1^i)) \oplus \Psi_j(\Psi_i(x_2^i)) \tag{1}$$

and

$$\alpha_2^{ij} = \Psi_j(\Psi_i(x_1^i)) \oplus \Psi_j(\Psi_i(x_3^i)) \tag{2}$$

Under this form, the obfuscated response values α_1^{ij} and α_2^{ij} can only and exclusively be used by Thing T_i and Thing T_j, and cannot be used by either of them to impersonate the other (e.g., in case of a malicious insider). Also, if Thing T_i is compromised, the values of α_1^{ij} and α_2^{ij} cannot be used by T_k to impersonate Thing T_i or T_j.

We note that this method of obfuscating the response by combining two responses together into one single response is inspired by the concept of distributed value used in cryptography [9]. Also, the idea of using the PUF function of both authenticating parties to construct responses is inspired by the work of Yanambaka *et al.* [14].

2) SECOND APPROACH

Considering the limited storage-capacity of resource-constrained devices, we only store one eCRP per Thing on a Thing's memory. That is, if a Thing T_0 is configured to authenticate n other Things $T_{i\neq 0}$, it only stores n extended CRP. However, we note that the eCRP is constructed based on three classical CRPs as we have used three challenges, x_1^i, x_2^i and x_3^i, and computed the obfuscated response values of α_1^{ij} and α_2^{ij} by combining their responses. In anyways, this way of storing the eCRPs allows using a given challenge-response pair more than once. In fact, it is not the case in classical PUF-based authentication protocols, where a strong PUF is required to generate a large number of CRPs, and each CRP has to be used only once.

B. T2T-MAP PHASES

Similar to all other PUF-based authentication protocols, T2T-MAP consists of two phases: the enrolment and the authentication phase. In this subsection, we present how the enrolment phase occurs in T2T-MAP, then show the verification and authentication. It is important to note that in addition to performing authentication, T2T-MAP allows the establishment of a symmetric cryptographic key between two Things once mutually authenticated.

1) ENROLMENT PHASE

This setup phase is conducted in a secure environment, where two Things (here the gateway is represented by a

TABLE 1. This table summaries the differences between the reviewed PUF-based authentication protocols. Due to space limitation, we have entitled some columns with a numbered letter *Ci*, where C0: Authentication scheme (T2M or T2T), C1: Mutual authentication using PUFs (from both authenticating parties), C2: lightweight (no asymmetric cryptography and no pre-established keys), C3: Scalability, C4: Key establishment (for message authentication, confidentiality, and data integrity), C5: Security evaluation, C6: Performance evaluation, C7: PUF-circuit size evaluation. The details of possible attacks are discussed in Section III of [64]. In this table, the symbols ✓, ✗, and ●, indicate Yes, No, and Possible, respectively.

Protocol	Used PUF	C0	Properties				Implementation platform	Evaluation			Possible attacks on the protocol
			C1	C2	C3	C4		C5	C6	C7	
This work (T2T-MAP)	Abstract PUF (Hash function)	T2T	✓	✓	✓	✓	ARDUINO MEGA & DUE boards	✓	✓	●	None
Boyapally et al., (2020) [11]	5-4 DAPUF	T2M	✗	✗	✓	✓	Digilent Nexys-4 Xilinx Artix-7 FPGA	✓	✓	✓	Spoofing attacks Replay attacks
Bansal et al., (2020) [12]	Unspecified	T2T	✗	✓	✓	✓	Simulation	✓	✓	✗	Spoofing attacks Message forging Key cracking CRPs disclosure
Qureshi et al., (2020) [13]	Modified APUF	T2M	✗	✓	✓	✗	ZYNQ-7000 SoC Xilinx ZC-706	✓	✓	✓	Spoofing attacks Replay attack DoS attacks
Yanambaka et al., (2019) [14]	Unspecified	T2M	✗	✓	✓	●	Altera DE2 microcontroller Raspberry Pi	✗	✓	✗	CRPs disclosure Modeling attacks Replay attack
Nozaki et al., (2019) [15]	XOR-APUF	T2M	✗	✓	●	✗	Xilinx Virtex-5 XC5VLX30 FPGA	✗	✗	✓	Spoofing attacks CRPs disclosure
Chatterjee et al., (2019) [16]	5-4 DAPUF	T2M	✗	✗	✓	✓	Digilent Nexys-4 Xilinx Artix-7 FPGA	✓	✓	✓	Spoofing attacks Key disclosure DoS attacks
Liang et al., (2019) [17]	TSMCA-PUF	T2M	✗	✓	✓	✗	Virtex II-PRO XUPV5-LX110T	✓	✓	✓	Desynchronization CRPs disclosure DoS attacks
Kim et al., (2019) [18]	Unspecified	T2M	✗	✗	✓	✗	STM32F4-MCU	✗	✗	✗	CRPs disclosure Key disclosure DoS attacks
Mahalat et al., (2018) [19]	Unspecified	T2T	✗	✓	✓	✗	—	✓	✓	✗	Spoofing attacks CRP disclosure DoS attacks
Mughal et al., (2018) [20]	Unspecified	T2M	✗	✗	●	✗	Unspecified	✗	✓	✗	Message forging Key disclosure DoS attacks
Barbareschi et al., (2018 – 2019) [21]	SRAM-PUF	T2T	✗	✓	✓	✗	STM73F7-MCU Cubietruck-board	✓	✓	✗	Desynchronization DoS attacks
Yilmaz et al., (2018) [22]	Unspecified	T2M	✗	✗	●	✗	Zolertia RE-Mote	✗	✗	✗	Spoofing attacks CRP disclosure RC5 cracking
Feng et al., (2018) [23]	SRAM-PUF	T2T	✗	✓	●	✗	MSP-EXP430G2 & MSP430G2553	✓	✓	✗	Spoofing attacks CRP disclosure Replay attack
Idriss et al., (2017) [24]	Unspecified	T2M	✗	✓	✓	✗	—	✓	✗	✗	Spoofing attacks MITM attack CRPs disclosure
Clupek et al., (2016) [25]	Unspecified	T2T	●	✓	●	●	—	✓	✗	✗	Secret disclosure Message forging Modeling attacks

TABLE 2. Summary of the notation adopted in T2T-MAP authentication protocol.

Symbol	Description
T_i	Thing T_i.
T_0	Gateway.
$T_{i\|j}$	Thing T_i and T_j, respectively.
x^i_v	The v^{th} challenge x generated by Thing T_i.
α^{ij}_v	The v^{th} obfuscated response α used by a Thing T_i to verify the authenticity of Thing T_j.
$\Psi_i(\cdot)$	The PUF function of Thing T_i.
$\Psi_i(x^i_v)$	The response of the PUF $\Psi_i(\cdot)$ on the v^{th} challenge x^i.
β^{ij}_v	The v^{th} obfuscated response β used by a Thing T_j to prove it authenticity to Thing T_j.
$\mathcal{H}(\cdot)$	A public hash function.
$\oplus$	The exclusive OR operator ($x \oplus y = x \cdot \overline{y} + \overline{x} \cdot y$).
$\Gamma i(T_j)$	The extended CRP (eCRP) used by Thing T_i to challenge and verify the authenticity of Thing T_j.
N_i	A nonce generated by Thing T_i.
N_{ij}	The Xor of Nonces N_i and N_j, i.e., $N_i \oplus N_j$.
$\mathcal{H}(*)$	The hash of a message m (payload) for integrity protection.
K	A cryptographic key.

Thing, e.g, T_j), T_i and T_j, are brought next to each other to exchange 3 challenges and reveal 2 obfuscated responses each. Both Things, T_i and T_j, start by generating random challenges, $x^{i|j}_1$, $x^{i|j}_2$ and $x^{i|j}_3$, then use their local PUF functions, $\Psi_i(\cdot)$ and $\Psi_j(\cdot)$, to compute the responses $\Psi_{i|j}(x^{i|j}_1)$, $\Psi_{i|j}(x^{i|j}_2)$, and $\Psi_{i|j}(x^{i|j}_3)$. Once computed, these responses constitute the new challenges to be sent to the other Thing. Each Thing $T_{i|j}$ sends the new challenges to the other Thing $T_{j|i}$. Upon receiving the challenges, each Thing uses its proper PUF to compute 3 response values over the received challenges, i.e., $\Psi_{j|i}(\Psi_{i|j}(x^{i|j}_1))$, $\Psi_{j|i}(\Psi_{i|j}(x^{i|j}_2))$, and $\Psi_{j|i}(\Psi_{i|j}(x^{i|j}_3))$. These responses are then used to compute two values $\alpha^{ij|ji}_1$ and $\alpha^{ij|ji}_2$. By xoring the first response with the second one, the value of $\alpha^{ij|ji}_1$ is computed. Also, the value of $\alpha^{ij|ji}_2$ is computed by xoring the first response with the third one, as expressed in Equation 1 and 2. Finally, both Things send to each other the values of $\alpha^{ij|ij}_1$ and $\alpha^{ji|ij}_2$ so that each one of them constructs the extended CRP as follows:

$$\Gamma_{i|j}(T_{j|i}) = (x^{i|j}_1, x^{i|j}_2, x^{i|j}_3, \alpha^{ij|ji}_1, \alpha^{ij|ji}_2) \tag{3}$$

2) AUTHENTICATION PHASE

This phase is illustrated by the MSC[5] of FIGURE 2, where we have set $i = 1$ and $j = 2$ for simplicity. The gateway is generally assumed to have more resources than Things. Although the gateway is also a Thing in the context of IoT, we rather prefer to explicitly call it a gateway instead of a Thing to make a difference in the resource-capabilities. The steps of the authentication are as follows:

Step 1. T2T-MAP assumes that a Thing $T_{i|j}$ starts the protocol by sending a direct request for authentication to another Thing $T_{j|i}$. This step is not illustrated in FIGURE 2. Then, Things T_i and T_j use the eCRP that is related to the gateway to retrieve the challenges $x^{i|j}_1$, $x^{i|j}_2$, and $x^{i|j}_3$ and apply their local PUF to produce the responses $\Psi_{i|j}(x^{i|j}_1)$, $\Psi_{i|j}(x^{i|j}_2)$, and $\Psi_{i|j}(x^{i|j}_3)$, which are then sent to the gateway as challenges along with a nonce $N_{i|j}$.

Step 2. The gateway G receives the challenges from both Things and applies its local PUF to compute the values of $\beta^{i0|j0}_1$ and $\beta^{i0|j0}_2$ for each Thing, as follows:

$$\beta^{i0|j0}_1 = \Psi_0(\Psi_{i|j}(x^{i|j}_1)) \oplus \Psi_0(\Psi_{i|j}(x^{i|j}_2)) \tag{4}$$

and

$$\beta^{i0|j0}_2 = \Psi_0(\Psi_{i|j}(x^{i|j}_1)) \oplus \Psi_0(\Psi_{i|j}(x^{i|j}_3)) \tag{5}$$

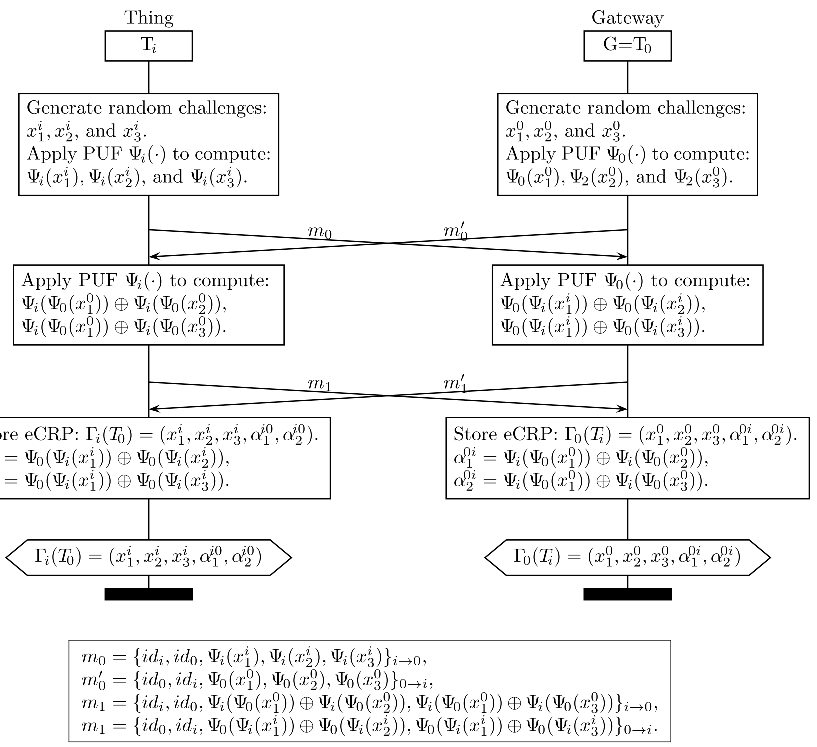
The gateway generates a nonce N_0 and then takes the first value $\beta^{i0|j0}_1$ and hashes it along with the xor of its nonce N_0 and the corresponding Thing's nonce $N_{i|j}$, i.e., computes $\mathcal{H}(\beta^{i0|j0}_1, N_{i|j} \oplus N_0)$. This hash is again hashed along with the nonces to compute the obfuscated response value $\mathcal{H}(\mathcal{H}(\beta^{i0|j0}_1, N_{i|j} \oplus N_0), N_{i|j}, N_0)$. This value is the response of the gateway to the challenges.

Step 3. The gateway uses the extended CRP that is related to each Thing to retrieve the challenges x^0_1, x^0_2, and x^0_3 and applies the local PUF function to produce the responses $\Psi_0(x^0_1)$, $\Psi_0(x^0_2)$, and $\Psi_0(x^0_3)$, which are the challenges to be sent to Things. The gateway then constructs and sends two messages m_1 and m'_1 that contain the identities of Things along with its identity, the nonces, the three challenges, the computed obfuscated response, and a hash value, denoted by $\mathcal{H}(*)$, that is computed over the entire message for message integrity protection.

Step 4. Things receive the gateway's messages and start by verifying its integrity, i.e., $\mathcal{H}(*)$.[6] Then, the extended

[5]MSC (Message Sequence Chart) is a graphical language for the description of the interaction between different components of a system. This language is standardized by the ITU (International Telecommunication Union) [75].

[6]For the sake of efficiency, we have decided to start the verification of the integrity of the messages before their authenticity. Our decision is based on the principle of fast decision with less computation. In fact, integrity verification consists of 1 hash operation and 1 comparison, whereas the verification of the authenticity consists of 2 hash operations and 1 comparison. In the worst case, the order does not matter as both orderings end up performing 3 hash operations and 2 comparisons.



Thing
T_i
Gateway
$\mathrm{G}=\mathrm{T}_0$
Generate random challenges: x_1^i, x_2^i, and x_3^i. Apply PUF $\Psi_i(\cdot)$ to compute: $\Psi_i(x_1^i), \Psi_i(x_2^i)$, and $\Psi_i(x_3^i)$.
Generate random challenges: x_1^0, x_2^0, and x_3^0. Apply PUF $\Psi_0(\cdot)$ to compute: $\Psi_0(x_1^0), \Psi_2(x_2^0)$, and $\Psi_2(x_3^0)$.
m_0
m_0'
Apply PUF $\Psi_i(\cdot)$ to compute: $\Psi_i(\Psi_0(x_1^0)) \oplus \Psi_i(\Psi_0(x_2^0))$, $\Psi_i(\Psi_0(x_1^0)) \oplus \Psi_i(\Psi_0(x_3^0))$.
Apply PUF $\Psi_0(\cdot)$ to compute: $\Psi_0(\Psi_i(x_1^i)) \oplus \Psi_0(\Psi_i(x_2^i))$, $\Psi_0(\Psi_i(x_1^i)) \oplus \Psi_0(\Psi_i(x_3^i))$.
m_1
m_1'
Store eCRP: $\Gamma_i(T_0) = (x_1^i, x_2^i, x_3^i, \alpha_1^{i0}, \alpha_2^{i0})$. $\alpha_1^{i0} = \Psi_0(\Psi_i(x_1^i)) \oplus \Psi_0(\Psi_i(x_2^i))$, $\alpha_2^{i0} = \Psi_0(\Psi_i(x_1^i)) \oplus \Psi_0(\Psi_i(x_3^i))$.
Store eCRP: $\Gamma_0(T_i) = (x_1^0, x_2^0, x_3^0, \alpha_1^{0i}, \alpha_2^{0i})$. $\alpha_1^{0i} = \Psi_i(\Psi_0(x_1^0)) \oplus \Psi_i(\Psi_0(x_2^0))$, $\alpha_2^{0i} = \Psi_i(\Psi_0(x_1^0)) \oplus \Psi_i(\Psi_0(x_3^0))$.
$\Gamma_i(T_0) = (x_1^i, x_2^i, x_3^i, \alpha_1^{i0}, \alpha_2^{i0})$
$\Gamma_0(T_i) = (x_1^0, x_2^0, x_3^0, \alpha_1^{0i}, \alpha_2^{0i})$
$m_0 = \{id_i, id_0, \Psi_i(x_1^i), \Psi_i(x_2^i), \Psi_i(x_3^i)\}_{i\to 0}$,
$m_0' = \{id_0, id_i, \Psi_0(x_1^0), \Psi_0(x_2^0), \Psi_0(x_3^0)\}_{0\to i}$,
$m_1 = \{id_i, id_0, \Psi_i(\Psi_0(x_1^0)) \oplus \Psi_i(\Psi_0(x_2^0)), \Psi_i(\Psi_0(x_1^0)) \oplus \Psi_i(\Psi_0(x_3^0))\}_{i\to 0}$,
$m_1 = \{id_0, id_i, \Psi_0(\Psi_i(x_1^i)) \oplus \Psi_0(\Psi_i(x_2^i)), \Psi_0(\Psi_i(x_1^i)) \oplus \Psi_0(\Psi_i(x_3^i))\}_{0\to i}$.

FIGURE 1. The enrolment phase of T2T-MAP. During this phase Things T_i ($\forall i \in \mathbb{N}^*$) and the gateway G exchange challenges and responses to construct each an extended CRP (a 5-tuple), denoted by $\Gamma_i(T_0)$ ($\Gamma_0(T_i)$, respectively). The 5-tuple $\Gamma_i(T_0)$ ($\Gamma_0(T_i)$, respectively) stores 3 challenges, x_1^i, x_2^i and x_3^i (x_1^0, x_2^0 and x_3^0, respectively), and 2 authentication values, α_1^{i0} and α_2^{i0} (α_1^{0i} and α_2^{0i}, respectively). These two values are computed using the locally generated challenges and the PUF functions of each authenticating party, i.e., $\Psi_i(\cdot)$ of Thing T_i and $\Psi_0(\cdot)$ of the gateway G.

CRP is used to retrieve the value of $\alpha_1^{i0|j0}$ which is hashed along with the xor of the nonces and then hashed again with the nonce of the concerned Thing $T_{i|j}$ and the nonce of the gateway N_0. For authentication, the result of hashing $\alpha_1^{i0|j0}$, i.e., $\mathcal{H}(\mathcal{H}(\alpha_1^{i0|j0}, N_{i|j} \oplus N_0), N_{i|j}, N_0)$, is compared to the received value $\mathcal{H}(\mathcal{H}(\beta_1^{i0|j0}, N_{i0|j0} \oplus N_0), N_{i|j}, N_0)$. If both values are equal, the gateway is authenticated by both Things.

Step 5. Both Things then use the received challenges to apply their local PUF function and compute the corresponding responses, $\Psi_{i|j}(\Psi_0(x_1^0))$, $\Psi_{i|j}(\Psi_0(x_2^0))$, and $\Psi_{i|j}(\Psi_0(x_3^0))$. Using the computed values, both Things compute the values of $\beta_1^{0i|0j}$ and $\beta_2^{0i|0j}$, as follows:

$$\beta_1^{0i|0j} = \Psi_{i|j}(\Psi_0(x_1^0)) \oplus \Psi_{i|j}(\Psi_0(x_2^0)) \tag{6}$$

and

$$\beta_2^{0i|0j} = \Psi_{i|j}(\Psi_0(x_1^0)) \oplus \Psi_{i|j}(\Psi_0(x_3^0)) \tag{7}$$

Once computed, the value of $\beta_1^{0i|0j}$ is hashed along with the xor of the nonces and then hashed again with the nonces, i.e., $\mathcal{H}(\mathcal{H}(\beta_1^{0i|0j}, N_{i|j} \oplus N_0), N_{i|j}, N_0)$. A message m_2 (and m_2') is constructed using the identities of the authenticating parties, the nonces, the hashed response, and the hash of the entire message $\mathcal{H}(*)$ for integrity protection. The message is then sent to the gateway.

Step 6. The gateway receives the messages m_2 and m_2' from Thing T_i and T_j, respectively. It uses the stored value $\alpha_1^{0i|0j}$ to compute the value of $\mathcal{H}(\mathcal{H}(\alpha_1^{0i|0j}, N_{i|j} \oplus N_0), N_{i|j}, N_0)$ and compare it to the received obfuscated response value $\mathcal{H}(\mathcal{H}(\beta_1^{0i|0j}, N_{i|j} \oplus N_0), N_{i|j}, N_0)$. If they are equal for both Things, then both Things are authenticated.

Step 7. The gateway generates a secret random nonce L_0 and computes two additional values α_3^{0i} and α_3^{0j} using the following equation:

$$\alpha_3^{0i|0j} = \alpha_1^{0i|0j} \oplus \alpha_2^{0i|0j} \tag{8}$$

Also, the gateway computes the values $L_0 \oplus \mathcal{H}(\alpha_3^{0i|0j})$ and $\mathcal{H}(\alpha_2^{0i|0j}) \oplus \alpha_2^{0j|0i}$, and then sends two messages m_3 and m_3' to Thing T_i and T_j, respectively. These messages contain the identity of the authenticating parties, the nonces, the value $L_0 \oplus \mathcal{H}(\alpha_3^{0i|0j})$, the value $\mathcal{H}(\alpha_2^{0i|0j}) \oplus \alpha_2^{0j|0i}$, and the hash of the entire message.

Step 8. Things receive the gateway's messages and start by verifying its integrity, i.e., $\mathcal{H}(*)$. The extended CRP is used to retrieve the value of $\alpha_2^{i0|j0}$ which is then hashed along with the xor of the nonces ($N_{i0|j0} = N_{i|j} \oplus N_0$) and then hashed again with the nonce so that it can be compared with the received value $\mathcal{H}(\mathcal{H}(\beta_2^{i0|j0}, N_{i|j} \oplus N_0), N_{i|j}, N_0)$ to authenticate the source of the message. Once authenticated, Things compute the value of $\beta_3^{0i|0j}$ as follows:

$$\beta_3^{0i|0j} = \beta_1^{0i|0j} \oplus \beta_2^{0i|0j} \tag{9}$$

Using the previously computed value $\beta_2^{0i|0j}$ in *Step* 5 (Equation 7) and the value of $\beta_3^{0i|0j}$ (Equation 9), Things compute the value of $\alpha_2^{0i|0j} = (\mathcal{H}(\alpha_2^{0j|0i}) \oplus \alpha_2^{0i|0j}) \oplus \mathcal{H}(\beta_2^{0j|0i})$ and the value of $L_0 = (L_0 \oplus \mathcal{H}(\alpha_3^{0i|0j})) \oplus \mathcal{H}(\beta_3^{0i|0j})$. At this stage, both Things have a shared secret nonce L_0 as well as a part of the extended CRP of the other Thing. These two information will be used to establish a T2T authentication between T_i and T_j.

Step 9. In this step, both Things start authenticating each other. To that end, Thing $T_{i|j}$ that first initiated the protocol generates a key K then xores the key with the hash of the xor of the obfuscated response $\alpha_2^{0j|0i}$ of Thing $T_{j|i}$ and the secret nonce L_0, to compute the value $K \oplus \mathcal{H}(\alpha_2^{0j|0i} \oplus L_0)$. This value allows to securely send and share the key with the other Thing. Then, it uses the nonces and the secret nonce L_0 to compute the value of $\mathcal{H}(\mathcal{H}(\alpha_2^{0j|0i}, N_{ij}), N_i, N_j, L_0)$. This value allows authenticating the source of the message to be sent as well as to link this second part of the authentication to the first part that was performed with the gateway (through the use of the secret nonce L_0). These values are sent to the other party along with a message integrity code, i.e., $\mathcal{H}(*)$.

Step 10. Upon the reception of the message from $T_{i|j}$, Thing $T_{j|i}$ checks the integrity of the message, i.e., $\mathcal{H}(*)$, and then uses the previously computed value $\beta_2^{0j|0i}$ in *Step* 5 (Equation 7) to check the authenticity of the source of the message as well as its linkability to the previous authentication part with the gateway. This consists of comparing the received value $\mathcal{H}(\mathcal{H}(\alpha_2^{0j|0i}, N_{ij}), N_i, N_j, L_0)$ with $\mathcal{H}(\mathcal{H}(\beta_2^{0j|0i}, N_{ij}), N_i, N_j, L_0)$. If the values are equal, then $T_{j|i}$ authenticates $T_{i|j}$. In this case, $T_{j|i}$ extracts the key K by computing $K = (K \oplus \mathcal{H}(\alpha_2^{0j|0i} \oplus L_0)) \oplus \mathcal{H}(\beta_2^{0j|0i} \oplus L_0)$. Once the key is computed, Thing $T_{j|i}$ hashes the obfuscated response $\alpha_2^{0i|0j}$ of Thing $T_{i|j}$ along with the key K to prove the correct key computation to Thing $T_{i|j}$. Also, it computes the value $\mathcal{H}(\mathcal{H}(\alpha_2^{0i|0j}, N_{ij}), N_i, N_j, L_0)$ for message source authenticity and sends these values along with the message integrity code.

Step 11. Thing $T_{i|j}$ receives the message of Thing $T_{j|i}$ (i.e., m_5) and starts by checking its integrity code, i.e., $\mathcal{H}(*)$. Then it uses the previously computed value $\beta_2^{0i|0j}$ in *Step* 5 (Equation 7) to check the authenticity of the source of the message as well as its linkability to the previous authentication part with the gateway by comparing the received value $\mathcal{H}(\mathcal{H}(\alpha_2^{0i|0j}, N_{ij}), N_i, N_j, L_0)$ with $\mathcal{H}(\mathcal{H}(\beta_2^{0i|0j}, N_{ij}), N_i, N_j, L_0)$. If the values are equal, then $T_{i|j}$ authenticates $T_{j|i}$. Also, Thing $T_{i|j}$ checks that the received value $\mathcal{H}(K \oplus \alpha_2^{0i|0j} \oplus L_0)$ is equal to $\mathcal{H}(K \oplus \beta_2^{0i|0j} \oplus L_0)$ to confirm that Thing $T_{j|i}$ has correctly computed the key.

This second part of the protocol is illustrated by the MSC of FIGURE 3, where we have set $i = 1$ and $j = 2$ for simplicity.

It is important to note that the protocol has actually been designed to operate on two types of configurations: (1) Thing-to-Thing authentication through a gateway, and (2) Thing-to-Thing authentication without a gateway. The first configuration is the one proposed in this paper, whereas the second has been discussed in details in [71].

The reason for having two possible configurations is to make the authentication protocol suitable for a number of applications that require heterogeneous architectures and typologies. Here are some key differences between the two configurations: (1) In T2T without gateway configuration (Configuration 2), Things have to locally store authentication information about each other, whereas in T2T with gateway (Configuration 1), they only store authentication information about the gateway (the gateway stores authentication information about the Things). Therefore, Configuration 2 is suitable for scenarios where Things have the ability to locally store authentication information. It is also suitable in a scenario where the presence of a gateway is impossible (e.g., WSNs in hostile environment, jet fighters at higher altitudes, etc,). (2) Configuration 2 is more suitable for architectures that involve a higher mobility, whereas Configuration 1 is more appropriate for architecture with less mobility. (3) Configuration 1 allows a greater scalability than Configuration 2 due to the fact that Things do not store authentication information locally about all other Things. However, the current version of Configuration 1 uses one single gateway that may constitute a bottleneck when the number of Things increases. (4) Configuration 2 can be considered less risky than Configuration 1. In fact, the gateway is a third party that may constitute a security risk in the case where it is compromised (e.g., insider attack). (5) Configuration 2 is convenient for network paradigms where Things directly communicate and authenticate each other without a third party (e.g., a vehicle authenticating a road sign), whereas Configuration 1 is suitable for Things that communicate through a third party (e.g., in a smart homes application, a user uses its smartphone to authenticate with a smart thermostat through a Wi-Fi access point and regulate the room temperature).

V. T2T-MAP PROPERTIES AND SECURITY ANALYSIS

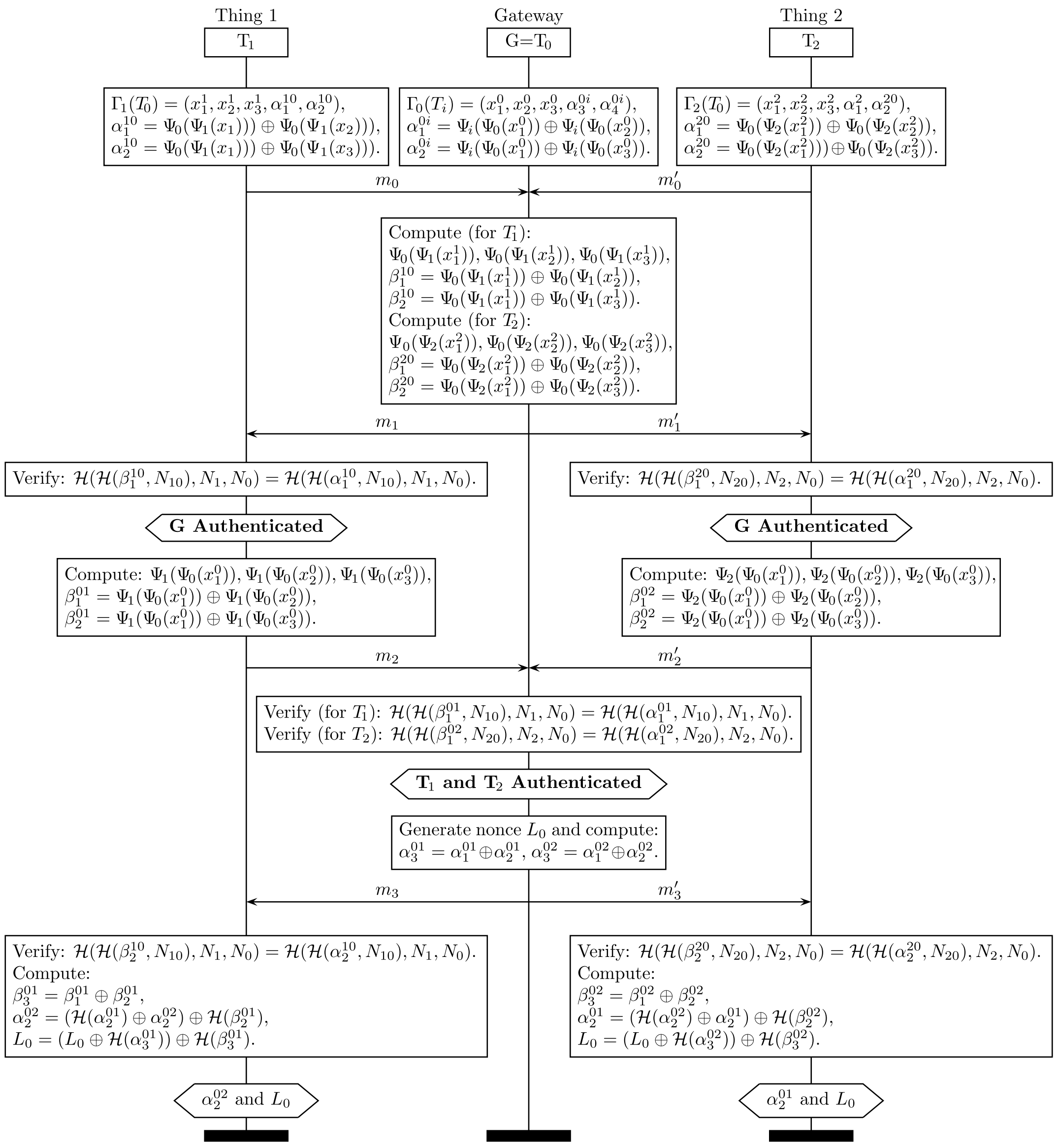
In this section, we discuss some security as well as performance properties of T2T-MAP. Next, we perform a security analysis on the protocol by discussing the resilience of T2T-MAP against some known attacks and then by formally proving its security using Tamarin security protocol prover.

A. PROTOCOL PROPERTIES

T2T-MAP provides a set of security and performance properties that conform to IoT security and performance requirements. In the following paragraphs, we present these properties:

1) LIGHTWEIGHT

From the cryptographic perspective, T2T-MAP is considered as a lightweight, if not an ultra-lightweight, authentication protocol for the following reasons: The protocol is based on the use of PUFs (Physical Unclonable Functions) to prove device identities. Moreover, it uses simple hash functions as well as the bitwise logical exclusive OR operator (XOR) to establish authentication and secret key


Thing 1
T_1
Gateway
$\mathrm{G}=\mathrm{T}_0$
Thing 2
T_2
$\Gamma_1(T_0) = (x_1^1, x_2^1, x_3^1, \alpha_1^{10}, \alpha_2^{10})$, $\alpha_1^{10} = \Psi_0(\Psi_1(x_1))) \oplus \Psi_0(\Psi_1(x_2)))$, $\alpha_2^{10} = \Psi_0(\Psi_1(x_1))) \oplus \Psi_0(\Psi_1(x_3)))$.
$\Gamma_0(T_i) = (x_1^0, x_2^0, x_3^0, \alpha_3^{0i}, \alpha_4^{0i})$, $\alpha_1^{0i} = \Psi_i(\Psi_0(x_1^0)) \oplus \Psi_i(\Psi_0(x_2^0))$, $\alpha_2^{0i} = \Psi_i(\Psi_0(x_1^0)) \oplus \Psi_i(\Psi_0(x_3^0))$.
$\Gamma_2(T_0) = (x_1^2, x_2^2, x_3^2, \alpha_1^2, \alpha_2^{20})$, $\alpha_1^{20} = \Psi_0(\Psi_2(x_1^2)) \oplus \Psi_0(\Psi_2(x_2^2))$, $\alpha_2^{20} = \Psi_0(\Psi_2(x_1^2))) \oplus \Psi_0(\Psi_2(x_3^2))$.
m_0
m'_0
Compute (for T_1): $\Psi_0(\Psi_1(x_1^1)), \Psi_0(\Psi_1(x_2^1)), \Psi_0(\Psi_1(x_3^1))$, $\beta_1^{10} = \Psi_0(\Psi_1(x_1^1)) \oplus \Psi_0(\Psi_1(x_2^1))$, $\beta_2^{10} = \Psi_0(\Psi_1(x_1^1)) \oplus \Psi_0(\Psi_1(x_3^1))$.
Compute (for T_2): $\Psi_0(\Psi_2(x_1^2)), \Psi_0(\Psi_2(x_2^2)), \Psi_0(\Psi_2(x_3^2))$, $\beta_1^{20} = \Psi_0(\Psi_2(x_1^2)) \oplus \Psi_0(\Psi_2(x_2^2))$, $\beta_2^{20} = \Psi_0(\Psi_2(x_1^2)) \oplus \Psi_0(\Psi_2(x_3^2))$.
m_1
m'_1
Verify: $\mathcal{H}(\mathcal{H}(\beta_1^{10}, N_{10}), N_1, N_0) = \mathcal{H}(\mathcal{H}(\alpha_1^{10}, N_{10}), N_1, N_0)$.
Verify: $\mathcal{H}(\mathcal{H}(\beta_1^{20}, N_{20}), N_2, N_0) = \mathcal{H}(\mathcal{H}(\alpha_1^{20}, N_{20}), N_2, N_0)$.
G Authenticated
G Authenticated
Compute: $\Psi_1(\Psi_0(x_1^0)), \Psi_1(\Psi_0(x_2^0)), \Psi_1(\Psi_0(x_3^0))$, $\beta_1^{01} = \Psi_1(\Psi_0(x_1^0)) \oplus \Psi_1(\Psi_0(x_2^0))$, $\beta_2^{01} = \Psi_1(\Psi_0(x_1^0)) \oplus \Psi_1(\Psi_0(x_3^0))$.
Compute: $\Psi_2(\Psi_0(x_1^0)), \Psi_2(\Psi_0(x_2^0)), \Psi_2(\Psi_0(x_3^0))$, $\beta_1^{02} = \Psi_2(\Psi_0(x_1^0)) \oplus \Psi_2(\Psi_0(x_2^0))$, $\beta_2^{02} = \Psi_2(\Psi_0(x_1^0)) \oplus \Psi_2(\Psi_0(x_3^0))$.
m_2
m'_2
Verify (for T_1): $\mathcal{H}(\mathcal{H}(\beta_1^{01}, N_{10}), N_1, N_0) = \mathcal{H}(\mathcal{H}(\alpha_1^{01}, N_{10}), N_1, N_0)$.
Verify (for T_2): $\mathcal{H}(\mathcal{H}(\beta_1^{02}, N_{20}), N_2, N_0) = \mathcal{H}(\mathcal{H}(\alpha_1^{02}, N_{20}), N_2, N_0)$.
T_1 and T_2 Authenticated
Generate nonce L_0 and compute: $\alpha_3^{01} = \alpha_1^{01} \oplus \alpha_2^{01}$, $\alpha_3^{02} = \alpha_1^{02} \oplus \alpha_2^{02}$.
m_3
m'_3
Verify: $\mathcal{H}(\mathcal{H}(\beta_2^{10}, N_{10}), N_1, N_0) = \mathcal{H}(\mathcal{H}(\alpha_2^{10}, N_{10}), N_1, N_0)$. Compute: $\beta_3^{01} = \beta_1^{01} \oplus \beta_2^{01}$, $\alpha_2^{02} = (\mathcal{H}(\alpha_2^{01}) \oplus \alpha_2^{02}) \oplus \mathcal{H}(\beta_2^{01})$, $L_0 = (L_0 \oplus \mathcal{H}(\alpha_3^{01})) \oplus \mathcal{H}(\beta_3^{01})$.
Verify: $\mathcal{H}(\mathcal{H}(\beta_2^{20}, N_{20}), N_2, N_0) = \mathcal{H}(\mathcal{H}(\alpha_2^{20}, N_{20}), N_2, N_0)$. Compute: $\beta_3^{02} = \beta_1^{02} \oplus \beta_2^{02}$, $\alpha_2^{01} = (\mathcal{H}(\alpha_2^{02}) \oplus \alpha_2^{01}) \oplus \mathcal{H}(\beta_2^{02})$, $L_0 = (L_0 \oplus \mathcal{H}(\alpha_3^{02})) \oplus \mathcal{H}(\beta_3^{02})$.
α_2^{02} and L_0
α_2^{01} and L_0

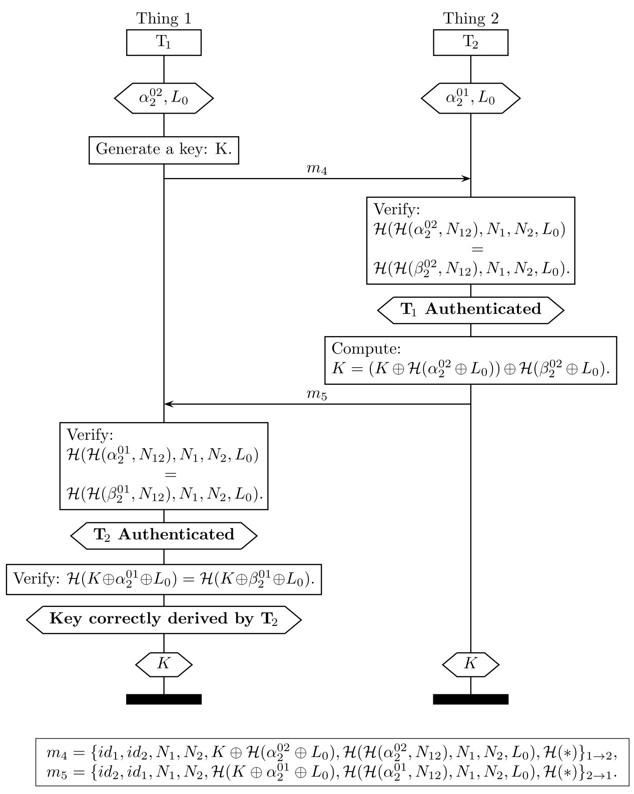
$$
\begin{aligned}
m_0 &= \{id_1, id_0, id_m, N_1, \Psi_1(x_1^1), \Psi_1(x_2^1), \Psi_1(x_3^1), \mathcal{H}(*)\}_{1\to 0},\\
m'_0 &= \{id_2, id_0, id_m, N_2, \Psi_2(x_1^2), \Psi_2(x_2^2), \Psi_2(x_3^2), \mathcal{H}(*)\}_{2\to 0},\\
m_1 &= \{id_0, id_1, id_m, N_1, N_0, \Psi_0(x_1^0), \Psi_0(x_2^0), \Psi_0(x_3^0), \mathcal{H}(\mathcal{H}(\beta_1^{10}, N_{10}), N_1, N_0), \mathcal{H}(*)\}_{0\to 1},\\
m'_1 &= \{id_0, id_2, id_m, N_2, N_0, \Psi_0(x_1^0), \Psi_0(x_2^0), \Psi_0(x_3^0), \mathcal{H}(\mathcal{H}(\beta_1^{20}, N_{20}), N_2, N_0), \mathcal{H}(*)\}_{0\to 2},\\
m_2 &= \{id_1, id_0, id_m, N_1, N_0, \mathcal{H}(\mathcal{H}(\beta_1^{01}, N_{10}), N_1, N_0), \mathcal{H}(*)\}_{1\to 0},\\
m'_2 &= \{id_2, id_0, id_m, N_2, N_0, \mathcal{H}(\mathcal{H}(\beta_1^{02}, N_{20}), N_2, N_0), \mathcal{H}(*)\}_{2\to 0},\\
m_3 &= \{id_0, id_1, id_m, N_1, N_0, \mathcal{H}(\mathcal{H}(\beta_2^{10}, N_{10}), N_1, N_0), \mathcal{H}(\alpha_2^{01}) \oplus \alpha_2^{02}, L_0 \oplus \mathcal{H}(\alpha_3^{01}), \mathcal{H}(*)\}_{0\to 1},\\
m'_3 &= \{id_0, id_2, id_m, N_2, N_0, \mathcal{H}(\mathcal{H}(\beta_2^{20}, N_{20}), N_2, N_0), \mathcal{H}(\alpha_2^{02}) \oplus \alpha_2^{01}, L_0 \oplus \mathcal{H}(\alpha_3^{02}), \mathcal{H}(*)\}_{0\to 2}.
\end{aligned}
$$

FIGURE 2. Message sequence chart of T2T-MAP. This first part of the protocol allows only a mutual authentication of two Things to a gateway. The notation $M_{x\to y}$ denotes a message M sent from x to y (where $x, y \in \{0, 1, 2\}$, 0 refers to the gateway G, 1 refers to Thing T_1, and 2 refers to Thing T_2). The PUF of G, T_1, and T_2, are $\Psi_0(\cdot)$, $\Psi_1(\cdot)$, and $\Psi_2(\cdot)$, respectively. Also, N_0, N_1, and N_2, are three nonces generated by G, T_1, and T_2, receptively, $\oplus$ is the bitwise eXclusive OR operator, N_{ij} is the XOR of the two nonces N_i and N_j ($N_{ij} = N_i \oplus N_j$), and $\mathcal{H}(\cdot)$ is the hash function. The variables x_1^i, x_2^i, and x_3^i, where $i \in \{0, 1, 2\}$, are challenges generated by a Thing T_i, whereas α_i^{jk} and β_i^{jk}, where $i \in \{1, 2, 3\}$, $j \in \{0, 1, 2\}$, and $k \in \{0, 1, 2\}$, are values that are computed using the PUF of two parties j and k over specific challenges. Finally, the notation $\mathcal{H}(*)$ refers to the hash of an entire message (i.e., its message integrity code, or MAC). This value is verified upon the reception of each message m_i.

establishment. In case encryption is needed, the protocol can adopt a lightweight encryption algorithm such as ChaCha.[7]

[7]ChaCha (RFC7905) is a lightweight stream cipher proposed by D. J. Bernstein in 2008. It is a more efficient version of the Salsa20 cipher.

Furthermore, the lightweight property of the protocol covers the fact that the protocol allows a relatively low storage overhead (see scalability below), low communication overhead and energy consumption, as we will demonstrate in Section VI-B.


Thing 1
T_1
Thing 2
T_2
α_2^{02}, L_0
α_2^{01}, L_0
Generate a key: K.
m_4
Verify: $\mathcal{H}(\mathcal{H}(\alpha_2^{02}, N_{12}), N_1, N_2, L_0) = \mathcal{H}(\mathcal{H}(\beta_2^{02}, N_{12}), N_1, N_2, L_0)$.
T_1 Authenticated
Compute: $K = (K \oplus \mathcal{H}(\alpha_2^{02} \oplus L_0)) \oplus \mathcal{H}(\beta_2^{02} \oplus L_0)$.
m_5
Verify: $\mathcal{H}(\mathcal{H}(\alpha_2^{01}, N_{12}), N_1, N_2, L_0) = \mathcal{H}(\mathcal{H}(\beta_2^{01}, N_{12}), N_1, N_2, L_0)$.
T_2 Authenticated
Verify: $\mathcal{H}(K \oplus \alpha_2^{01} \oplus L_0) = \mathcal{H}(K \oplus \beta_2^{01} \oplus L_0)$.
Key correctly derived by T_2
K
K

$$m_4 = \{id_1, id_2, N_1, N_2, K \oplus \mathcal{H}(\alpha_2^{02} \oplus L_0), \mathcal{H}(\mathcal{H}(\alpha_2^{02}, N_{12}), N_1, N_2, L_0), \mathcal{H}(*)\}_{1\rightarrow 2},$$
$$m_5 = \{id_2, id_1, N_1, N_2, \mathcal{H}(K \oplus \alpha_2^{01} \oplus L_0), \mathcal{H}(\mathcal{H}(\alpha_2^{01}, N_{12}), N_1, N_2, L_0), \mathcal{H}(*)\}_{2\rightarrow 1}.$$

FIGURE 3. The second part of T2T-MAP, where both Things, T_1 and T_2, authenticate each other and agree on a symmetric cryptographic key K to be used to secure future communications.

2) MUTUAL AUTHENTICATION

T2T-MAP provides mutual authentication. In fact, to establish authentication between three entities, each entity is required to use its own PUF function to generate authentication responses that will prove its identity.

3) SCALABILITY

The protocol can be qualified as scalable for the following reason. In terms of storage overhead, each individual Thing T_i has to store $2n + 3$ values, where $n > 0$ is the number of Things $T_{j \neq i}$ registered on Thing T_i's memory. For instance, each Thing T_i stores only 5 values when $n = 1$. Each Thing will only store the eCRP of the gateway (the challenges x_1, x_2, and x_3, are the same for all Things). Hence, if the variables are expressed in 32 bytes, each Thing will permanently have to store only 160 bytes, which is a reasonable overhead. This overhead grows linearly following the function $f(n) = 2n + 3$, where $n > 0$ is the number of registered eCRPs on a Thing.

4) KEY ESTABLISHMENT

T2T-MAP allows the establishment of a secret key at the end of the authentication. This key can be used for encryption (e.g., using ChaCha) and data integrity. The key is randomly generated during the authentication and immediately destroyed at the end of the session.

5) FRESHNESS AND LIVENESS

T2T-MAP protocol uses nonces for each authentication session, which makes, to some extent, each authentication session fresh and unique. This would prevent any replay attack using messages from a previous authentication session. Also, the protocol uses the nonce L_0 to allow Things T_i and T_j verify that the current communication is part of the previous communication with the gateway. This would prevent any session hijacking attempts.

6) AVAILABILITY

For the authentication protocol to be available at all times, each authenticating party has to have its authentication information, i.e., eCRPs, up-to-date. In the current version of T2T-MAP, the stored eCRPs are static and not updated, which would make the protocol always available (no desynchronization). However, following the security principles, these eCRPs need to be updated at some point. Thus, we assume the existence of a secure and frequent eCRP-updating procedure to allow Things to store and use new eCRPs. For example, this eCRP-updating procedure may consist of storing one additional eCRP per device during the enrolment phase and particularly use these additional eCRPs for the update procedure. This would prevent the possibility of forward security-related attacks as well as brute force attacks as discussed in the next paragraph.

7) FORWARD SECURITY

This property is strongly related to the lifetime of the used authentication information (e.g., the lifetime of an eCRP). If the lifetime is too long, an attacker would have enough time to brute force and disclose authentication information to perform the impersonation. For example, through a session hijacking attack (cf., next subsection), an attacker would be able to learn 50% of the eCRP of a given Thing T_i (i.e., α_2^{0i}). Then, through brute-forcing and one single XOR-operation, the attacker will be able to learn the remaining part of the eCRP (i.e., α_1^{0i} and α_3^{0i}). Using this information, the attacker can completely impersonate Thing T_i during a future authentication session. Therefore, to mitigate this attack and provide forward security, we assume the existence of a secure eCRP-updating procedure that will allow Things to frequently and securely update the stored eCRPs.

8) NON-REPUDIATION

As PUFs are widely assumed to be unclonable functions and the authentication in T2T-MAP relies on the use of PUFs, we can claim that the security service of non-repudiation is guaranteed and enforced in T2T-MAP. Thus, no authenticating party can deny having participated in an authentication session.

B. SECURITY ANALYSIS

In what follows, the security strength of T2T-MAP against known attacks is discussed:

1) MACHINE LEARNING ATTACKS

These attacks are prevented by not exchanging the responses in plaintext during an authentication. T2T-MAP adopts a combination of two challenges, e.g., $\beta_1^{j|i} = \Psi_{i|j}(\Psi_{j|i}(x_1^{j|i})) \oplus \Psi_{i|j}(\Psi_{j|i}(x_2^{j|i}))$, and sends the response double hashed along with nonces, e.g., $\mathcal{H}(\mathcal{H}(\beta_1^{j|i}, N_{ij}), N_i, N_j)$. This obfuscation technique prevents eavesdroppers from collecting challenges and their corresponding responses during authentication to build a prediction model. Also, the base value of the challenges is not sent during authentication. In fact, what is being sent is the response from the local PUF for a challenge that is never revealed. For example, when Thing T_1 tries to authenticate the gateway, it sends the value $\Psi_1(x_1^1)$ and keeps x_1^1 confidential.

2) REPLAY ATTACK

A replay attack is prevented through the use of nonces. For example, throughout the authentication, three nonces are used: N_0 (generated by the gateway), N_1 (generated by Thing T_1), and N_2 (generated by Thing T_2). The obfuscated responses (i.e., $\beta_1^{ij|ji}$, $\beta_2^{ij|ji}$, and $\beta_3^{ij|ji}$) are never sent in plaintext. They are first hashed along with the xor of the nonces ($N_{ij} = N_{ji} = N_i \oplus N_j$) and then hashed again with the nonces, e.g., $\mathcal{H}(\mathcal{H}(\beta_1^{j|i}, N_{ij}), N_i, N_j)$. This makes the responses linked to the current session and cannot be reused in another session. An attacker may think of intercepting some messages and using the same nonce $N_{i|j}$ to replay the messages and spoof Thing $T_{i|j}$. This would be difficult, or impossible, for the attacker as the latter cannot force the other party, $T_{j|i}$, to reuse the same nonce $N_{j|i}$, used in a previous authentication session.

3) COMPROMISING GATEWAY/THING

If an attacker manages to gain access to the gateway or a Thing, it will not be able to infer and learn the eCRP of any other Thing. In fact, the eCRP, $\Gamma_{i|j}(T_{j|i})$, which is stored on Thing $T_{i|j}$ about Thing $T_{j|i}$ is information that is used by Thing $T_{i|j}$ to only verify the authenticity of Thing $T_{j|i}$. It cannot be used in anyways by Thing $T_{i|j}$ to prove that it is Thing $T_{j|i}$. Furthermore, an eCRP, $\Gamma_{i|j}(T_{i|j})$, is tightly-coupled to the two Things $T_{i|j}$ and $T_{j|i}$, and cannot be used by another Thing T_k ($k \neq i \neq j$).

4) SESSION HIJACKING

In this attack scenario, a malicious insider Thing, let us say T_k, performs a legitimate authentication with Thing T_i through the gateway and obtains the value α_2^{0i}. As previously discussed, this value is used by Thing T_i to prove its identity to another Thing T_j. Then, after some time, the malicious Thing T_k eavesdrops another authentication between the gateway and Things T_i and T_j. At the stage where the authentication with the gateway is achieved and both Things T_i and T_j have received from the gateway the information $\alpha_2^{0i|0j}$ about the counterpart $T_{j|i}$, the malicious Thing T_k spoofs Thing T_i and blocks its messages so that they do not reach Thing T_j. At this point, the malicious Thing T_k, can prove to T_j that it is T_i as it has the value α_2^{0i} that it has learned from a previous legitimate authentication session. Therefore, to prevent such an attack, we have used the nonce L_0 of the gateway as an indicator of the current session. The value of L_0 is secretly shared among the legitimate parties. It will be used by the communicating Things, in this case T_i and T_j, to verify whether the received messages are somehow related to the authentication session that occurred with the gateway (liveness). Therefore, without the knowledge of L_0, the malicious Thing T_k will not be able to hijack the session.

5) INSIDER SPOOFING

A malicious Thing, say T_k, could operate a legitimate authentication with another Thing, say T_i, and obtain the response information α_2^{0i} to spoof T_i later on. In this situation, the malicious Thing will not be able to spoof T_i as it has only one obfuscated response out of two obfuscated responses to prove that it is T_i. In addition, the initial part of the authentication with the gateway requires the attacker the knowledge of the information α_1^{0i}, which T_k cannot obtain. Also, the value α_1^{0i} is never sent in plaintext but always double hashed along with nonces.

6) BRUTE FORCING PUF-RESPONSES

As the nonces, N_i and N_j, are sent in plaintext during the authentication, an attacker may try to brute force the hash function $\mathcal{H}(\cdot)$ to find the obfuscated response value that was generated by an authenticating party. This would depend on the size of the response. If the response is in n bits, the attacker has to try at most 2^n values before finding the response. To make this attack difficult and infeasible, instead of using the obfuscated response as the first parameter for the hash function, i.e., $\mathcal{H}(\beta_v^{ij|ji}, N_i, N_j)$, where $v \in \{1, 2\}$, we send the hash along with nonces of the hash of the obfuscated response along with the xor of nonces, i.e., $\mathcal{H}(\mathcal{H}(\beta_v^{ij|ji}, N_{ij}), N_i, N_j)$, where $N_{ij} = N_i \oplus N_j$. If the attacker brute forces the transferred hash, it will find the value of $\mathcal{H}(\beta_v^{ij|ji}, N_{ij})$, which needs to be brute-forced again to find the value of $\beta_v^{ij|ji}$. In this case, the attacker will face a time complexity of $\mathcal{O}(2^{n+2})$ to be able to disclose the eCRP of a particular Thing T_i. Also, if the attacker (compromised Thing T_j) applies the session hijacking approach to learn the value of α_2^{0i} of Thing $\mathrm{T}_{i \neq j}$, the attacker will have to brute force a time complexity of $\mathcal{O}(2^{n+1})$. Therefore, we recommend the use of a bit-length of at least 64-bit (i.e., 8 bytes) for the response values so that the brute force attack on the hash function will be infeasible. This would eventually prevent the replay attack.

7) SECRET KEYS DISCLOSURE ATTACK

During the authentication, the protocol allows the communicating parties to establish a secret key to be used for encryption and/or data integrity. This key is immediately destroyed at the end of the communication. Therefore, the key is generated only when needed and erased once the session is terminated, which means that no secret keys are present permanently on the devices. This allows the devices to be resistant to any physical key disclosure attacks.

In addition to the above informal security analysis, we have used Tamarin [65], a well-established software tool for security protocol verification and theorem proving, to formally prove the security of T2T-MAP. To that end, we have implemented T2T-MAP in Tamarin's protocol specification language. This basically consists of writing rules of facts and lemmas. In fact, Tamarin models protocol's set of executions as a labeled transition system (LTS). The states of the LTS are multisets of facts formalizing the local states of the authenticating party running the protocol, the attacker's knowledge, and the exchanged messages. The transitions of the LTS are formalized using rules. Lemmas however, are used to express security properties, e.g., secrecy, to be verified by the tool in order to claim the security property for the protocol. The specification code of the protocol (a.k.a., the security protocol theory in Tamarin) consists of $+/-216$ lines of code. It can be accessed online at [66].

Next, based on the design of the protocol as well as on the above security analysis, we can observe that the security of T2T-MAP strongly relies on the secrecy of the following variables: (1) The challenges x_i^j, where $i, j \in \mathbb{N}$. (2) The obfuscated responses, α_i^{jk} and β_i^{jk}, where $i, j, k \in \mathbb{N}$. (3) The shared nonce L_0 (generated by the gateway). (4) The shared key K. Therefore, we can claim that if these variables are kept secrete and not disclosed to the attacker in anyways, then the protocol cannot be compromised. To verify this claim using Tamarin prover, we have written lemmas to express the secrecy of the aforementioned variables and prove whether the lemmas are always true. After executing the verification of the lemmas on Tamarin (which took around $+/-3$ seconds on a DELL Precision T7500 having an Intel Xeon E5507 at 2.27GHz CPU and 6GB of RAM, and running Linux Ubuntu LTS 20.04 Operating System), the tool proved the lemmas to be always true. This means that there exist no possible execution for the protocol that leads to the disclosure of any of those variables.

VI. PROTOCOL IMPLEMENTATION AND EVALUATION

In this section, we present the implementation detail of T2T-MAP and then evaluate its performance w.r.t. different protocol configurations.

A. PROTOCOL IMPLEMENTATION

To implement T2T-MAP, we have used Arduino development boards as low-cost and resource-constrained devices. We have specifically used two types of Arduino boards, namely, the Arduino Mega 2560 (R3) and the Arduino DUE. These two boards embed different microcontrollers of different processing and storage capacity as illustrated in TABLE 3. Also, we have used different configurations for the protocol with respect to the size of the challenges, the responses, the nonces, and the hash outputs. In this subsection, we present how we have implemented the PUF function, the eCRPs (extended CRPs), the communication, and the protocol code.

TABLE 3. Arduino boards used to implement T2T-MAP.

Board Specifications	ARDUINO MEGA 2560 (R3)	ARDUINO DUE
Microcontroller Name	ATmega2560 AVR RISC	Atmel SAM3X8E ARM Cortex-M3
Processor's Wordsize	8 bits	32 bits
Processor's Frequency	16 MHz	84 MHz
Available RAM (SRAM)	8 KB	96 KB
Available ROM (Flash memory)	256 KB	512 KB
Operating Voltage	5.0V	3.3V

1) PUF IMPLEMENTATION

As there exist many types of PUFs, in this paper, we do not use a particular PUF. Also, we do not design or implement a new one either. We rather consider the use of an abstract PUF and keep the protocol as generic as possible. For the sake of this implementation, we have adopted the BLAKE[8] hash function that is available in the Arduino Cryptographic Library[9] to emulate the abstract PUF as well as the hash function $\mathcal{H}(\cdot)$ used in the protocol. We emphasize that the abstraction is only to implement a generic prototype of T2T-MAP so that performance analysis can be performed. Researchers can use any type of PUF to build their own version of the authentication protocol.

Nevertheless, to use a hash function instead of a PUF to implement the protocol and perform a performance evaluation (in particular, in terms of execution time and energy consumption). We need to verify that the execution time of a PUF as well as its energy consumption are negligible w.r.t. a hash function execution time and energy consumption. This has been verified w.r.t. the results reported in [55]–[57]. In anyways, a hash function will be used to provide the desired output size and allow the use of a PUF function that requires a small area size and produces outputs of smaller sizes.

2) eCRPs IMPLEMENTATION

We have used random nonces as challenges and the hash of the nonces as their corresponding responses. Also, we have used different sizes, namely, 8 bytes, 16 bytes, and 32 bytes, for the challenges and responses, so that we can study the performance of the protocol when the sizes are changed (e.g., when upgrading the security level). The Arduino Cryptographic Library provides a random number generator to generate nonces of arbitrary sizes. Thus, for an n-bit configuration, we have used the random number generator to

[8]BLAKE is a cryptographic hash function based on ChaCha cipher.
[9]https://rweather.github.io/arduinolibs/crypto.html.

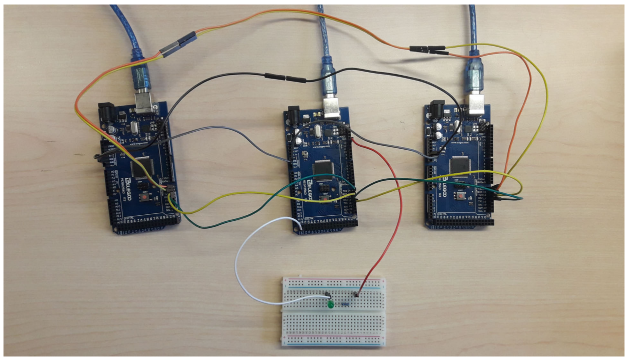


FIGURE 4. T2T with gateway configuration using Arduino Mega 2650 boards. The transmission (TXD) GPIO pin of a board (Thing T_1) is connected to the receiving (RXD) GPIO pin of another board (Thing T_2) and vice-versa. Also, their ground pins (GND) have to be connected to each other.

TABLE 4. The size, in KB, of the protocol's code, compiled for both Arduino boards and for different sizes of the variables.

Board name	Node	Variables sizes (bytes) 8	16	32
ARDUINO MEGA 2560	Thing T_1	17470	17410	17782
	Thing T_2	17292	17532	17580
	Gateway	19956	18972	20310
ARDUINO DUE	Thing T_1	19620	19740	19980
	Thing T_2	19564	19788	19956
	Gateway	20308	19780	20636

generate nonces in n bits and feed the BLAKE hash function to produce their corresponding responses, which are also in n bits.

3) COMMUNICATION IMPLEMENTATION

To allow different boards to communicate with each other, we have used the embedded serial communication medium, which is also known as the URAT (Universal asynchronous receiver-transmitter) system. In fact, on each programming board, there exist some GPIO (General Purpose Input/Output) pins that are dedicated for communication, namely, TXD for transmission and RXD for reception.[10] Thus, to connect two development boards through the serial medium, the TXD pin of one board has to be connected to the RXD pin of the other board and vice-versa. Additionally, their ground pins (GND) have to be connected to each other. Such wiring is illustrated in FIGURE 4. Furthermore, for the communication speed, we have set the serial communication rate to 200000 baud, which is around 200kbps.

4) PROTOCOL CODE

For the protocol to run over an Arduino board, the protocol has to be first written using C/C++ programming language and then compiled for a specific Arduino board (e.g., Arduino Mega 2560 R3) using the Arduino IDE v1.8.13. The protocol, as well as the firmware, are then uploaded and written into the flash memory of the board (i.e., flashing the board). TABLE 4 reports the size, in KB, of the sketch (code) of the protocol for each Arduino development board, node (Thing T_1, Thing T_2, and Gateway), and for different size of the variables, i.e., challenges, responses, nonces, and message integrity codes.[11] We have noticed that the code of the gateway is larger than the code of Things (T_1 and T_2). This is because the gateway has to communicate with two different Things during a given authentication session. The size of the programs should not necessarily increase by increasing the size of the variables. In fact, the impact of increasing the size of the variables is related to the amount of SRAM (central memory) that will be used to execute the program, which will increase by increasing the size of the allocated variables.

[10]Note that other GPIO pins (e.g., from 2 to 13) can also be configured to be used for communication through a software library, e.g., *SoftwareSerial*.

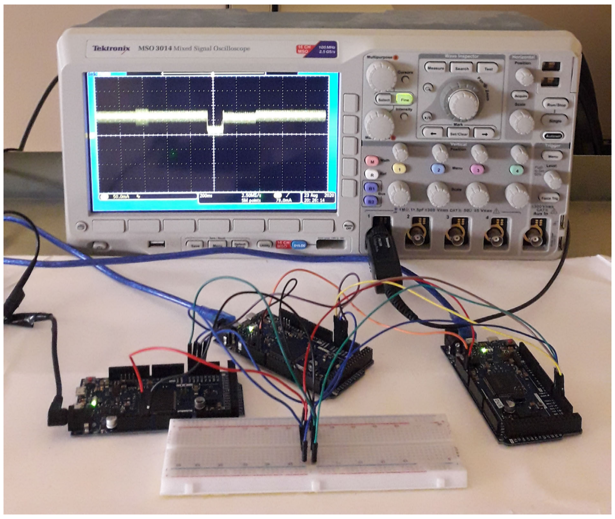


FIGURE 5. The experimental setting: An oscilloscope, model Tektronix MSO3014, three Arduino DUE boards (from left to right: Thing T_1, Gateway, and Thing T_2), and 10Ω resistor wired in series on the power cable of an arbitrary board (Thing T_1 in this figure). The oscilloscope's voltage probe is connected to both resistor's extremities to measure the voltage drop between the resistor.

B. PROTOCOL EVALUATION

In this subsection, we evaluate the performance T2T-MAP in terms of execution time, communication overhead, and power consumption, in the ideal case, i.e., no attackers.

1) EXECUTION TIME AND ENERGY CONSUMPTION

To evaluate the execution time of T2T-MAP, we use a digital oscilloscope and plot the voltage/current variation over time on a given development board. Having the plot of voltage/current over time, we can identify the portion of the signal that represents the execution of the protocol and measure its time duration. This would give us the execution time of the protocol on the measured board. However, we note that measuring the execution time using the oscilloscope can

[11]In the Arduino `platform.txt` file, the compiler can be optimized (by specifying an optimization option: `-Os`, `-O0`, `-O1`, `-O2`, or `-O3`) to reduce the code size which would generally result in faster execution. We left the compiler on its default settings (i.e., option `-Os`).

be performed as part of the energy consumption evaluation process. Therefore, in the next paragraphs, we present the energy consumption evaluation process and perform the execution time measurements at the same time.

To compute the electrical energy consumed by each Thing (or board) during t seconds, we apply the Joule's law as follows, where E is the energy in Joule, V is the supplied voltage in Volt, I is the intensity of the current flowing to the board in Ampere, and t is the time in Second:

$$E_{(Joule)} = V_{(Volt)} \times I_{(Ampere)} \times t_{(Second)} \tag{10}$$

Hence, to calculate the energy that is consumed during the execution of T2T-MAP, we need to know the voltage that is being supplied to the board, the intensity of the electric current flowing within the board during the execution of T2T-MAP, and the execution time of the protocol. The supplied voltage is the operating voltage of the boards. According to the boards' specifications, it is 5V for the Arduino Mega and 3.3V for the Arduino DUE. The execution times of T2T-MAP, for different configurations, can be determined from the voltage/current plot over time as we will see later. The intensity of the electrical current, however, needs to be measured during the execution of T2T-MAP. To that end, we use the Tektronix MSO3014 digital storage oscilloscope to measure the intensity of the current.

To measure the intensity of the electrical current flowing within the board during a period of time using an oscilloscope, we can apply one of the following two approaches: (1) We connect a resistor in series to the power cable of the board. Then, we connect the oscilloscope's voltage probe cable to both resistor's extremities to measure the voltage drop between the resistor. Having the measured voltage drop V and the known resistor's capacity R, we apply the Ohm's law ($V = R \times I$) to calculate the intensity of the electrical current I. (2) We connect the oscilloscope's current probe cable directly to the power cable of the board and measure the intensity of the electrical current that is flowing through it. Although the second approach is straightforward, we have adopted the first approach due to the unavailability of the oscilloscope's current probe cable in our laboratory. Thus, we have used the oscilloscope to measure the voltage drop over a 10Ω resistor and set up the oscilloscope to automatically apply the Ohm's law. This visualizes the current flow instead of the voltage drop (viz., FIGURE 5).

Practically, to measure the intensity of the current during the execution of T2T-MAP with high precision, we need to identify the part of the signal, i.e., the signal of $I(t)$, during which the authentication protocol is executing. To that end, we have reprogrammed each Thing (board) in such a way so that the board performs the following operations: (1) Booting. (2) Stay idle for τ ms. (3) Turn an LED light ON for τ ms and then turn it OFF. (4) Execute the protocol. (5) Turn the LED light ON for τ ms then turn it OFF. For the Arduino Mega board the duration of τ is 1000ms, whereas for the Arduino DUE it is 200ms. This is due to the considerable difference in the execution speed of T2T-MAP on both boards. In this way, we can identify the portion of the signal during which the protocol is executing.

FIGURE 6 and FIGURE 7 illustrate the execution of the protocol's code on the Arduino Mega and Arduino DUE boards for variable sizes 8 bytes, 16 bytes, and 32 bytes, respectively. We note that these figures have been generated using a sampling rate of 100 samples per second. We can observe in FIGURE 6 that the Arduino Mega board draws an $\mathcal{M}$-shaped current intensity during its boot phase (approximately within the first 1 second). Then the current intensity drops down for another 1 second during the board idle state. The current intensity jumps up when the LED light is turned ON for a duration of 1 second. Then, it drops down during the protocol execution. Once the protocol terminates, the current intensity jumps up again due to the lighting of the LED for another 1 second before it turns OFF. At this point, we can easily identify that the part of the signal that represents the execution of the authentication protocol is the part of the signal where the current intensity has dropped down between two consecutive lightings of the LED. The same can be observed in FIGURE 7, except that the Arduino DUE board draws a $\mathcal{V}$-shaped current intensity during its boot phase instead of $\mathcal{M}$. Schematically, the part of the signal that represents the execution of the authentication protocol is situated within the two dashed vertical lines on both figures.

We have performed at least 50 consecutive executions to compute the average time for the protocol to execute on each board. We have measured the time length of the parts of the signal that represents the execution of T2T-MAP with the help of the oscilloscope's cursors. FIGURE 8 illustrates the average execution time of T2T-MAP for different variables sizes (8, 16, and 32 bytes), and for both the development boards, i.e., Arduino Mega 2650 and Arduino DUE. Also, it is important to note that the execution time of the entire authentication protocol is expressed by the execution time of Thing T_1 as the latter is the last Thing that finishes the execution of T2T-MAP.

In the case of the Arduino Mega 2650 (R3), the entire protocol (authentication and key derivation confirmation) requires an average execution time of 214.20ms, 270.20ms, and 348.40ms, when the variables size is 8, 16, and 32 bytes, respectively. It is important to bear in mind that T2T-MAP involves the mutual authentication of three communicating parties, i.e., Thing T_1, Thing T_2, and gateway G, as well as the establishment of a symmetric key.

Also, the authentication of Thing T_1 and T_2 with respect to the gateway is processed sequentially (T_1 then T_2). Although the authentication execution would run faster if the authentication requests are processed in parallel, we decided to keep the processing sequential for simplicity. More importantly, the hardware platform on which the protocol is running is an 8-bit microcontroller that runs at a speed of 16MHz with 8KB of SRAM. Thus, for a resource-constrained platform like this one, the obtained execution time is reasonably acceptable. In the case of the Arduino DUE, however, the protocol runs faster. It requires an average execution time of

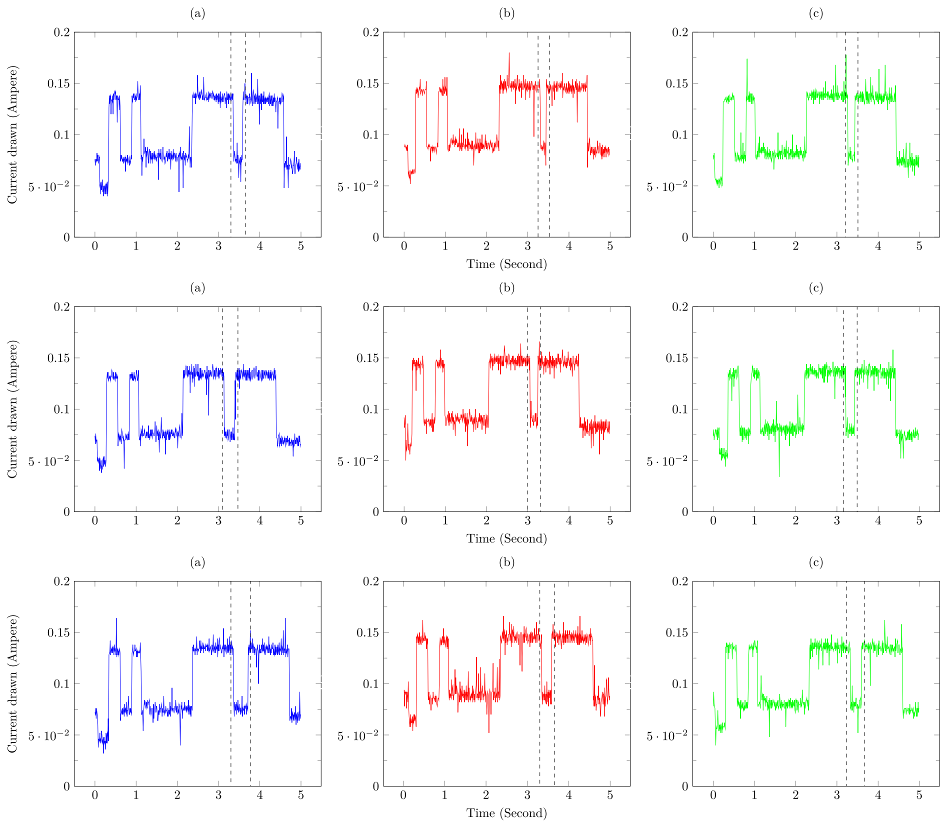


FIGURE 6. **The electrical current (in Ampere) drawn by Thing T_1 (a), Gateway G (b), and Thing T_2 (c), during the execution of T2T-MAP on Arduino Mega boards when the 8-byte (Row 1), 16-byte (Row 2), or 32-byte (Row 3) configuration is adopted for the size of the variables.**

14.76ms, 27.58ms, and 65.00ms, when the variables size is 8, 16, and 32 bytes, respectively. In this second experiment, the protocol runs faster as the hardware platform is a 32-bit microcontroller that operates at a speed of 84MHz with 96KB of SRAM. Although this hardware configuration is still not that powerful, the average execution time of the protocol is excellent. In both cases (i.e., Arduino Mega 2650 and Arduino DUE), the execution time has increased when we have increased the size of the variables from 8 bytes to 16 bytes, and then to 32 bytes. This is totally normal as each board on a given configuration has to process more bytes than in a previous configuration.

If we consider Arduino Mega 2650 board, we can observe that the 16-byte configuration is around 26% slower than the 8-byte configuration, but 100% more secure. Hence, it is worth to sacrifice around 56ms more time for the execution of the protocol to guarantee double security. Moreover, the 32-byte configuration is around 29% slower than the 16-byte configuration, but 100% more secure. For double security, the choice of a 32-byte configuration would be straightforward. Also, although the 32-byte configuration is around 62% slower than the 8-byte configuration, it offers a security strength that is four times stronger than the security of the 8-byte configuration (200% more secure), which confirms our previous choice. Nevertheless, on Arduino DUE board, we can see that by doubling the size of the variables, the execution time doubles. Hence, the choice of a configuration will depend on the application constraints. Although, the 8-byte and the 16-byte configurations are 200% and 100%, respectively, slower than the 32-byte configuration, the latter offers a stronger security with an acceptable execution time (∼65ms).

Next, we have measured the intensity of the electrical current during the execution of T2T-MAP as follows: We have recorded the current intensity values during the protocol execution phase using a sampling rate of 500k samples per second. Then, we have computed the average value of the current intensities.

We have found that that the average electrical current intensity used by Thing T_1, Gateway G, and Thing T_2, is 87mA, 75mA, and 79mA, respectively, when the Arduino Mega board is being used. However, it is 57mA, 61mA, and 58mA, when the Arduino DUE board is being used. At this stage, knowing the average current intensity used by each Thing (board) during the execution of the protocol, and knowing the amount of time needed to execute the protocol, we have computed the amount of energy that is consumed by each Thing during the execution of the protocol

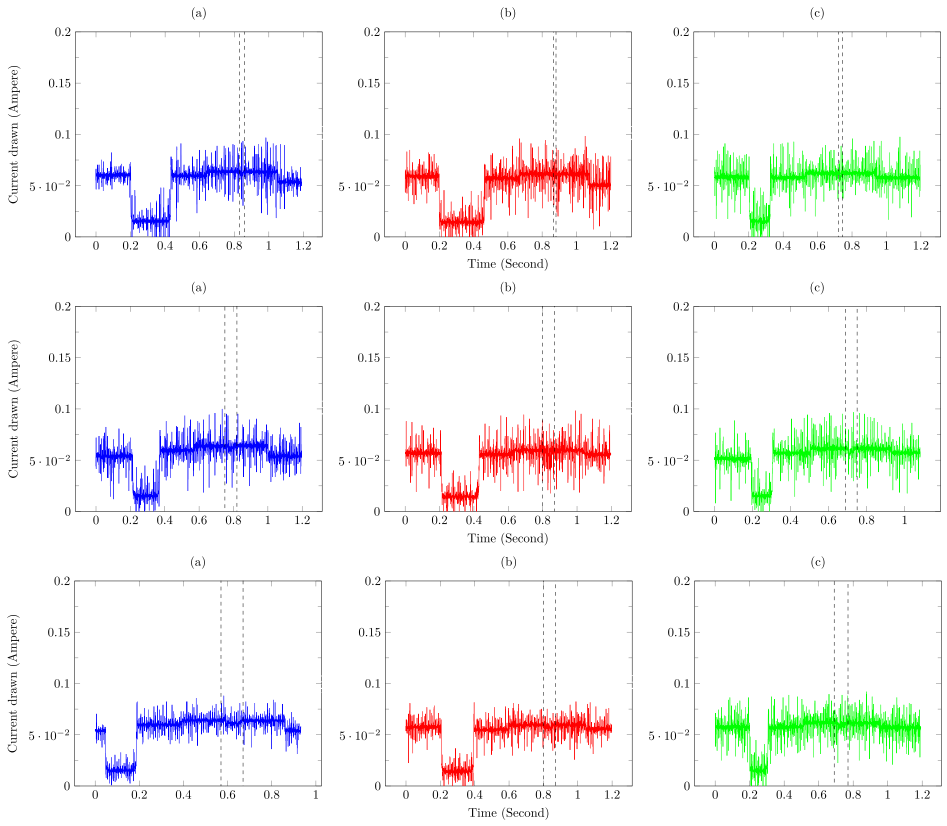


FIGURE 7. The electrical current (in Ampere) drawn by Thing T_1 (a), Gateway G (b), and Thing T_2 (c), during the execution of T2T-MAP on Arduino DUE boards when the 8-byte (Row 1), 16-byte (Row 2), or 32-byte (Row 3) configuration is adopted for the size of the variables.

TABLE 5. The amount of energy (mJ) consumed during the execution of T2T-MAP on Thing T_1, Gateway G, and Thing T_2, on both Arduino boards (i.e., Mega and DUE), and for different sizes of the variables (i.e., 8, 16, and 32 bytes).

Board name	Node	Variables sizes (bytes)		
		8	16	32
ARDUINO MEGA 2560	Thing T_1	93.18	117.54	151.55
	Thing T_2	61.42	75.52	98.40
	Gateway	53.32	71.61	91.32
ARDUINO DUE	Thing T_1	02.77	05.19	12.22
	Thing T_2	02.06	04.31	11.82
	Gateway	01.35	01.89	09.07

by applying the Joule's law (viz., Equation 10). TABLE 5 reports the energy consumed by each Thing during the execution of the protocol, on both boards, and for different variable sizes. These results express a low power consumption that is reasonably suitable for resource-constrained devices. On the Arduino DUE board, we can clearly observe a low power consumption, which does not go upper than 13mJ, when the maximum variable size configuration is adopted (i.e., 32 bytes).

2) COMMUNICATION OVERHEAD EVALUATION

In this subsection, we analyze the communication overhead of T2T-MAP during its execution for different variable sizes. TABLE 6 reports the size, in bytes, of the messages that are exchanged during T2T-MAP execution for both platforms and for different variable sizes. The size of the messages doubles along with the size of the variables. For example, message m_0 that is sent from Thing T_1 to the gateway has a total size of 40 bytes when the variables are expressed in 8 bytes. This size doubles to 80 bytes when the variables are expressed in 16 bytes, and to 160 bytes when the variables are in 32 bytes. FIGURE 9 illustrates the number of bytes each authenticating party sends during the authentication. The number of bytes sent by Thing T_1 and Thing T_2 is equal. The gateway, however, needs to send almost the double of what both Things send during the execution of the protocol as the gateway needs to communicate with both Things. Overall, the protocol requires 432 bytes (3456 bits), 864 bytes (6912 bits), and 1728 bytes (13824 bits), when the size of the variables is 8 bytes, 16 bytes, and 32 bytes, respectively. These amounts of bytes are reasonably acceptable for a resource-constrained environment where the network bandwidth is limited. In fact, as we aim to implement the protocol in an IoT application that uses a short-range wireless technology (e.g., Wi-Fi, Bluetooth, or ZigBee), the communication overhead would not be affected by the adopted technology. For example, if we consider the application of the 16-byte configuration on wireless networks such as Wi-Fi, Bluetooth,

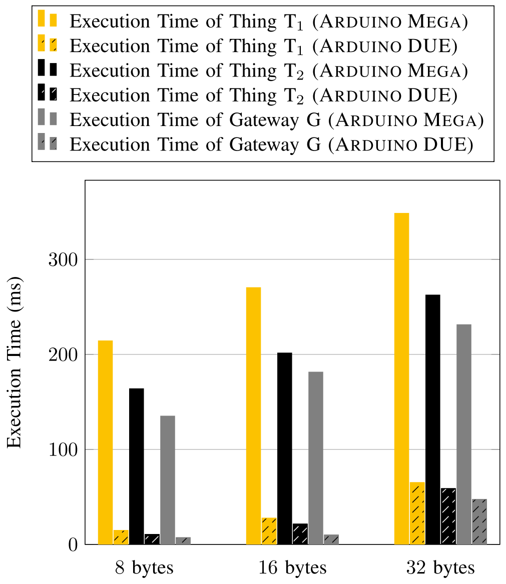


FIGURE 8. The execution time (ms) of T2T-MAP measured using the oscilloscope for Thing T_1, Thing T_2, and Gateway G, on both Arduino boards (i.e., Mega and DUE), and for different size configurations, i.e., 8, 16, and 32 bytes.

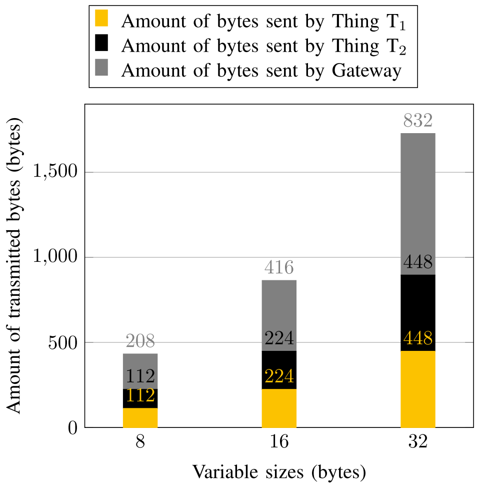


FIGURE 9. The number of Bytes (all messages) sent by each authenticating party, i.e., Thing T_1, Thing T_2, and Gateway G, during the authentication for different variable sizes (i.e., 8, 16, and 32 bytes).

or ZigBee, then each individual message of the protocol will fit within the payload of these technologies. Moreover, it is important to note that we have arbitrarily chosen that all fields (i.e., nonces, challenges, and hash function outputs) in a given message have the same size, either 8 bytes, 16 bytes, or 32 bytes. It is possible to express these fields in different sizes, which would certainly reduce the size of certain messages and improve the communication overhead.

3) DISCUSSION

In the following paragraphs, we discuss the security and performance features of T2T-MAP and compare them with protocols of the related work.

TABLE 6. The size of the exchanged messages (bytes) during T2T-MAP execution for each authenticating party and for each variable size configuration.

Node	Message MSC	Message Sent to	Size (bytes)		
			8	16	32
T_1	m_0	Gateway G	40	80	160
	m_2	Gateway G	32	64	128
	m_4	Thing T_2	40	80	160
T_2	m'_0	Gateway G	40	80	160
	m'_2	Gateway G	32	64	128
	m_5	Thing T_1	40	80	160
G	m_1	Thing T_1	56	112	224
	m'_1	Thing T_2	56	112	224
	m_3	Thing T_1	48	96	192
	m'_3	Thing T_2	48	96	192

T2T-MAP provides security as well as performance properties. For security, the protocol allows the mutual authentication of devices using their embedded PUF circuits. This important feature makes T2T-MAP more secure than the reviewed related work protocols, where the PUF is used only by one authenticating party. For example, by imposing the use of PUFs on each authenticating party, T2T-MAP enforces the service of non-repudiation. Also, the protocol allows the establishment of a secret key that will be used to provide message confidentiality and integrity. In addition, the protocol provides forward security and prevents replay attacks through the use of an eCRP-update procedure and fresh nonces, respectively. Furthermore, we have shown that most of the reviewed protocols (9 out of 15) are vulnerable to CRPs disclosure (e.g., during their storage or transmission) and PUF impersonation by malicious insiders. T2T-MAP thwarts these attacks by adopting the concept of extended CRPs along with the cryptographic concept of distributed value (cf., Section VI-A). Moreover, in Section VI-B, we have discussed its resilience against machine learning attacks, replay attacks, node compromising attacks, session hijacking, malicious insider spoofing, brute force attacks, and secret key disclosure. Additionally, we have used Tamarin security protocol verifier to automatically prove the security of T2T-MAP by proving the property of secrecy.

In terms of performance characteristics, T2T-MAP is a lightweight protocol that uses PUFs, simple hash functions, and the bitwise exclusive logical OR operator (XOR). Also, in the case where an encryption is used to secure the communications, lightweight ciphers, such as ChaCha, can be used to maintain the property of lightweightness. Also, we emphasis that the latter property expresses the low storage overhead, low communication overhead, and low energy consumption of the protocol. These important features allow the protocol to be adopted by resource-constrained devices. In addition, the protocol is considered to be scalable as it theoretically allows a large number of Things to be mutually and securely authenticated without any constraints.

We have implemented the protocol on resource-constrained devices and evaluated the performance of the protocol w.r.t. execution time, energy consumption, and communication overhead, for three different configurations and under two different hardware platforms. T2T-MAP allows a mutual authentication of three parties (e.g., Thing T_1, Thing T_2, and gateway) to be performed in 214.20ms, 270.20ms, and 348.40ms, when variables (i.e., nonces, PUF's challenges, PUF's responses, and hash outputs) are expressed in 8 bytes, 16 bytes, and 32 bytes, respectively, on Arduino Mega boards. This authentication is even faster on Arduino DUE boards, where it takes around 14.76ms, 27.58ms, and 65.00ms, for its completion for the respective variable sizes. The execution times are within the range that is reasonably acceptable for resource-constrained applications. Also, compared to the execution times reported by the related work on resourceful hardware, we can claim that our protocol is highly competitive and provides fast authentication.

With respect to energy consumption, T2T-MAP shows reasonable results. Depending on the device (i.e., being Thing T_1, Thing T_2, or gateway), the energy consumption varies between 53.32mJ and 93.18mJ, 71.61mJ and 117.54mJ, and 91.32mJ and 151.55mJ, for the respective variable sizes on Arduino Mega boards. However, energy consumption is considerably lower on Arduino DUE boards as it varies between 1.35mJ and 2.77mJ, 1.89mJ and 5.19mJ, and 9.07mJ and 12.22mJ, for the respective variables sizes. These values are reasonably low for resource-constrained devices, such as the ones used in IoT infrastructures.

During the protocol communication, the bandwidth consumption reveals an acceptable overhead. In fact, the protocol requires the transmission of 208 bytes, 416 bytes, 832 bytes, for the respective variable sizes. This communication overhead is reasonably suitable for networks with limited bandwidths. It is possible to distribute the size of the variables in a non-uniform way where the different fields are expressed in unequal sizes. This would certainly improve the communication overhead as certain messages will have a smaller size.

To appraise the efficiency of our authentication protocol, we wanted to compare our performance results with the results of the related work. Nevertheless, we found that it was a very challenging task that may end up drawing unfair conclusions. In fact, we were not able to perform any consistent comparison due to the following reasons: (1) Many implementations from the related work use powerful devices that are not resource-constrained devices, which makes it unfair to compare with an implementation that is fully resource-constrained. (2) Most of the related work does not provide enough information about the configurations and the results. This leads to an incomplete comparison. (3) Most of, if not all, related work, do not provide the complete structure of the messages that are exchanged during the execution of the protocol. Making random assumptions on the structure of the messages results in an unfair comparison.

Finally, we report in the first row of TABLE 1 the features and properties of T2T-MAP so that it can be compared to the ones of the related work PUF-based authentication protocols.

VII. CONCLUSION

IoT (Internet of Things) is a networking paradigm that allows billions of heterogeneous devices, called Things, to be connected to the Internet. This important evolution has and still expanding the classical network of the Internet from a Machine-to-Machine (M2M) communication system to a Things-to-Machine (T2M) and Things-to-Things (T2T) communication system. However, on the downside, it has invited cybercriminals to exploit the new heterogeneous infrastructure and mount catastrophic and diversified cyberattacks. Interestingly, most of these cyberattacks, if not all, are due to security vulnerabilities in the adopted authentication protocols. This has turned the attention of many researchers and industrial companies to invest a large amount of efforts to come up with new authentication protocols that are suitable for IoT.

Although many IoT authentication protocols have been proposed in the literature during the past decade, most of them, if not all, do not have a security-by-design and do not fulfill the IoT security and performance requirements. Furthermore, most of these protocols were not designed to be used in T2T architectures, where Things are supposed to autonomously and securely authenticate each other without any human intervention. This has turned our attention to investigate in this research direction to analyze the current advancement and propose possible improvements.

Therefore, in this paper, we have proposed T2T-MAP, a lightweight mutual authentication protocol for T2T architectures in the context of IoT. The protocol applies PUFs (Physical Unclonable Functions), as a physical security-by-design technology, to allow Things to efficiently authenticate each other with the lowest cost. Compared to the existing PUF-based authentication protocols from the literature [11]–[25], we claim that T2T-MAP is more resilient to various attacks, such as CRPs (Challenge-Response Pairs) disclosure, malicious insider, replay, session hijacking, brute force, secret disclosure, machine-learning, and node compromising attacks. Also, it provides mutual authentication, non-repudiation, forward-security, and liveness. Furthermore, besides being resilient to various attacks, T2T-MAP is scalable, lightweight, fast, and energy-efficient.

To summarize, the main contributions of this paper are as follows:

1) We have designed a lightweight secure-by-design authentication protocol for IoT. The protocol uses PUFs technology to allow resource-constrained devices to mutually authenticate each other without involving any human intervention. We have adopted the concept of extended CRPs (eCRPs) to make the protocol secure against CRPs disclosure, malicious insider, and modeling attacks. These attacks were shown to be possible on many authentication protocols from the related work.
2) We have discussed the resilience of T2T-MAP w.r.t. machine-learning, replay, node compromising, session

hijacking, insider spoofing, brute force, and secret disclosure attacks. In addition, we have employed, Tamarin, a security protocol verification tool to verify the security of the protocol.

3) We have implemented the protocol on resource-constrained devices and on different platforms using three different configurations. These configurations differ by the size of the variables (i.e., nonce, PUF's challenges, PUF's responses, and hash function outputs) that may be manipulated by the protocol. We have evaluated and discussed the performance of T2T-MAP with respect to the execution time, communication overhead, and energy consumption. We have shown that T2T-MAP runs at a fairly acceptable speed, has a reasonable communication overhead, and consumes a negligible amount of energy.

Despite all the advantageous features that T2T-MAP provides, the protocol still has some limitations as nothing is perfect. First of all, the protocol is designed in such a way so that each device, i.e., Thing, stores only one eCRP about any other device. We have assumed (in Section VI-A) the existence of an eCRP update procedure so that devices can renew the stored eCRPs and maintain the property of forward-security as well as the resilience against brute force attacks. This eCRP update procedure may consist of storing one additional eCRP per device during the enrolment phase and particularly use these additional eCRPs for the update procedure. We plan to investigate and integrate this procedure on a future version of the protocol. Furthermore, along with all related work protocols, T2T-MAP is vulnerable to race condition-based attacks discussed in [67]–[70]. We plan to improve the security of T2T-MAP with respect to these attacks. We also plan to use an existing PUF, e.g., an SRAM-PUF, to deploy the protocol on a real-life IoT application that adopts short-range wireless technologies, such as Wi-Fi, Bluetooth, ZigBee, or RFID, and mitigate some of the recently reported attacks on these technologies.

DISCLAIMER

This work is part of a Ph.D. thesis [71] that was published at Queen's University in January 2021. In the thesis, a PUF-based authentication protocol was designed and developed for two configurations: (i) Thing-2-Thing authentication through a gateway and (ii) Thing-2-Thing authentication without a gateway. The first configuration is presented in this paper. Thus, the reader would find high similarity between the content of this work and the part of the thesis that discusses the protocol in terms of structure and content.

REFERENCES

[1] The-Guardian. (Accessed: Sep. 25, 2020). *DDoS Attack That Disrupted Internet Was Largest of Its Kind in History, Experts Say*. Accessed: 2016. [Online]. Available: https://www.theguardian.com/technology/2016/oct/26/ddosattack-dyn-mirai-botnet

[2] NewYork-Times. (Accessed: Sep. 25, 2020). *Stuxnet Worm Attack on Iranian Nuclear Facilities*. Accessed: 2011. [Online]. Available: https://www.nytimes.com/2011/01/16/world/middleeast/16stuxnet.html

[3] K. Zetter. (Accessed: Sep. 25, 2020). *Inside the Cunning, Unprecedented Hack of Ukraine's Power Grid*. Accessed: 2016. [Online]. Available: https://www.wired.com/2016/03/inside-cunning-unprecedented-hackukraines-power-grid

[4] TheRegister. (Accessed: Sep. 25, 2020). *Finns Chilling as DDoS Knocks Out Building Control System*. Accessed: 2016. [Online]. Available: https://www.theregister.com/2016/11/09/finns_chilling_as_ddos_knocks_out_building_control_system/

[5] The-Guardian. (Accessed: Sep. 25, 2020). *Fiat Chrysler Recalls 1.4m Vehicles in Wake of Jeep Hacking Revelation*. Accessed: 2015. [Online]. Available: https://www.theguardian.com/business/2015/jul/24/fiat-chrysler-recalljeep-hacking

[6] BleepingComputer. (Accessed: Sep. 25, 2020). *BrickerBot Dev Claims Cyber-Attack That Affected Over 60,000 Indian Modems*. Accessed: 2017. [Online]. Available: https://www.bleepingcomputer.com/news/security/brickerbotdev-claims-cyber-attack-that-affected-over-60-000-indian-modems

[7] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn, "IoT goes nuclear: Creating a ZigBee chain reaction," *IEEE Secur. Privacy*, vol. 16, no. 1, pp. 54–62, Jan. 2018.

[8] B. Schneier, *Click Here to Kill Everybody: Security and Survival in a Hyper-Connected World*. New York, NY, USA: 2W. Norton & Company, 2018, ch. 5, p. 2.

[9] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[10] T. Machida, D. Yamamoto, M. Iwamoto, and K. Sakiyama, "A new mode of operation for arbiter PUF to improve uniqueness on FPGA," in *Proc. Federated Conf. Comput. Sci. Inf. Syst.*, Sep. 2014, pp. 871–878.

[11] H. Boyapally, P. Mathew, S. Patranabis, U. Chatterjee, U. Agrawal, M. Maheshwari, S. Dey, and D. Mukhopadhyay, "Safe is the new smart: PUF-based authentication for load modification-resistant smart meters," *IEEE Trans. Dependable Secure Comput.*, early access, May 6, 2020, doi: 10.1109/TDSC.2020.2992801.

[12] G. Bansal, V. Chamola, B. Sikdar, N. Kumar, and M. Guizani, "Lightweight mutual authentication protocol for V2G using physical unclonable function," *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7234–7246, Jul. 2020.

[13] M. A. Qureshi and A. Munir, "PUF-IPA: A PUF-based identity preserving protocol for Internet of Things authentication," in *Proc. IEEE 17th Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2020, pp. 1–7.

[14] V. Yanambaka, S. Mohanty, E. Kougianos, D. Puthal, and L. Rachakonda, "PMsec: PUF-based energy-efficient authentication of devices in the Internet of Medical Things (IoMT)," in *Proc. IEEE Int. Symp. Smart Electron. Syst. (iSES) (Formerly iNiS)*, Dec. 2019, pp. 320–321.

[15] Y. Nozaki and M. Yoshikawa, "Secret sharing schemes based secure authentication for physical unclonable function," in *Proc. IEEE 4th Int. Conf. Comput. Commun. Syst. (ICCCS)*, Feb. 2019, pp. 445–449.

[16] U. Chatterjee, V. Govindan, R. Sadhukhan, D. Mukhopadhyay, R. S. Chakraborty, D. Mahata, and M. M. Prabhu, "Building PUF based authentication and key exchange protocol for IoT without explicit CRPs in verifier database," *IEEE Trans. Depend. Sec. Comput.*, vol. 16, no. 3, pp. 424–437, May/Jun. 2019.

[17] W. Liang, S. Xie, J. Long, K.-C. Li, D. Zhang, and K. Li, "A double PUF-based RFID identity authentication protocol in service-centric Internet of Things environments," *Inf. Sci.*, vol. 503, pp. 129–147, Dec. 2019.

[18] B. Kim, S. Yoon, Y. Kang, and D. Choi, "PUF based IoT device authentication scheme," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2019, pp. 1460–1462.

[19] M. H. Mahalat, S. Saha, A. Mondal, and B. Sen, "A PUF based light weight protocol for secure WiFi authentication of IoT devices," in *Proc. 8th Int. Symp. Embedded Comput. Syst. Design (ISED)*, Dec. 2018, pp. 183–187.

[20] M. A. Muhal, X. Luo, Z. Mahmood, and A. Ullah, "Physical unclonable function based authentication scheme for smart devices in Internet of Things," in *Proc. IEEE Int. Conf. Smart Internet Things (SmartIoT)*, Aug. 2018, pp. 160–165.

[21] M. Barbareschi, A. D. Benedictis, and N. Mazzocca, "A PUF-based hardware mutual authentication protocol," *J. Parallel Distrib. Comput.*, vol. 119, pp. 107–120, Sep. 2018.

[22] Y. Yilmaz, S. R. Gunn, and B. Halak, "Lightweight PUF-based authentication protocol for IoT devices," in *Proc. IEEE 3rd Int. Verification Secur. Workshop (IVSW)*, Jul. 2018, pp. 38–43.

[23] W. Feng, Y. Qin, S. Zhao, and D. Feng, "AAoT: Lightweight attestation and authentication of low-resource things in IoT and CPS," *Comput. Netw.*, vol. 134, pp. 167–182, Apr. 2018.

[24] T. Idriss and M. Bayoumi, "Lightweight highly secure PUF protocol for mutual authentication and secret message exchange," in *Proc. IEEE Int. Conf. RFID Technol. Appl. (RFID-TA)*, Sep. 2017, pp. 214–219.

[25] V. Clupek and V. Zeman, "Robust mutual authentication and secure transmission of information on low-cost devices using physical unclonable functions and hash functions," in *Proc. 39th Int. Conf. Telecommun. Signal Process. (TSP)*, Jun. 2016, pp. 100–103.

[26] P. S. Ravikanth, "Physical one-way functions," Ph.D. dissertation, School Archit. Planning, Massachusetts Inst. Technol., Cambridge, MA, USA, 2001.

[27] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *Proc. 9th ACM Conf. Comput. Commun. Secur. (CCS)*, 2002, pp. 148–160.

[28] J. Zhang, L. Wan, Q. Wu, and G. Qu, "DMOS-PUF: Dynamic multi-key-selection obfuscation for strong PUFs against machine learning attacks," 2018, *arXiv:1806.02011*. [Online]. Available: http://arxiv.org/abs/1806.02011

[29] J. Delvaux, "Machine-learning attacks on PolyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF–FSMs," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 8, pp. 2043–2058, Aug. 2019.

[30] U. Ruhrmair, J. Solter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, "PUF modeling attacks on simulated and silicon data," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 11, pp. 1876–1891, Nov. 2013.

[31] J. Tobisch and T. B. Georg, "On the scaling of machine learning attacks on PUFs with application to noise bifurcation," in *Proc. Int. Workshop Radio Freq. Identificat., Secur. Privacy Issues*. Cham, Switzerland: Springer, 2015, pp. 17–31.

[32] F. Ganji, T. Shahin, and J. P. Seifert, "Why attackers win: On the learnability of XOR arbiter PUFs," in *Proc. Int. Conf. Trust Trustworthy Comput.* Cham, Switzerland: Springer, 2015, pp. 22–39.

[33] U. Rührmair and J. Sölter, "PUF modeling attacks: An introduction and overview," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6.

[34] M. Yoshikawa and Y. Nozaki, "Helper data aware cloning method for physical unclonable function," in *Proc. IEEE Int. Conf. Smart Cloud (SmartCloud)*, Nov. 2017, pp. 47–51.

[35] M. S. Alkatheiri and Y. Zhuang, "Towards fast and accurate machine learning attacks of feed-forward arbiter PUFs," in *Proc. IEEE Conf. Dependable Secure Comput.*, Aug. 2017, pp. 181–187.

[36] A. Mahmoud, U. Rührmair, M. Majzoobi, and F. Koushanfar, "Combined modeling and side channel attacks on strong PUFs," Cryptol. ePrint Arch., TU München, Munich, Germany, Tech. Rep. 2013/632, 2013. [Online]. Available: https://eprint.iacr.org/2013/632

[37] S. Wei, J. B. Wendt, A. Nahapetian, and M. Potkonjak, "Reverse engineering and prevention techniques for physical unclonable functions using side channels," in *Proc. 51st Annu. Design Autom. Conf. Design Autom. Conf. (DAC)*, 2014, pp. 1–6.

[38] U. Rührmair, X. Xu, J. Sölter, A. Mahmoud, M. Majzoobi, F. Koushanfar, and W. Burleson, "Efficient power and timing side channels for physical unclonable functions," in *Proc. Cryptograph. Hardw. Embedded Syst. (CHES)*, 2014, pp. 476–492.

[39] J. Delvaux and I. Verbauwhede, "Side channel modeling attacks on 65 nm arbiter PUFs exploiting CMOS device noise," in *Proc. IEEE Int. Symp. Hardware-Oriented Secur. Trust (HOST)*, Jun. 2013, pp. 137–142.

[40] R. Kumar and W. Burleson, "Side-channel assisted modeling attacks on feed-forward arbiter PUFs using silicon data," in *Proc. 11th Int. Workshop Radio Freq. Identificat.*, vol. 9440. Cham, Switzerland: Springer, 2015, pp. 53–67.

[41] Y. Nozaki and M. Yoshikawa, "Power consumption aware machine learning attack for feed-forward arbiter PUF," in *Proc. Int. Conf. Comput. Inf. Sci.*, 2019, pp. 49–62.

[42] D. Merli, D. Schuster, F. Stumpf, and G. Sigl, "Semi-invasive EM attack on FPGA RO PUFs and countermeasures," in *Proc. Workshop Embedded Syst. Secur. (WESS)*, vol. 2, 2011, pp. 1–9.

[43] D. Merli, D. Schuster, F. Stumpf, and G. Sigl, "Side-channel analysis of PUFs and fuzzy extractors," in *Proc. Conf. Trust Trustworthy Comput. (TRUST)*, vol. 6740, 2011, pp. 33–47.

[44] T. McGrath, I. E. Bagci, Z. M. Wang, U. Roedig, and R. J. Young, "A PUF taxonomy," *Appl. Phys. Rev., Rev.*, vol. 6, Mar. 2019, Art. no. 011303.

[45] J. H. Anderson, "A PUF design for secure FPGA-based embedded systems," in *Proc. 15th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2010, pp. 1–6.

[46] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *Proc. 9th Int. Workshop Cryptograph. Hardw. Embedded Syst.*, 2007, pp. 63–80.

[47] F. Tehranipoor, N. Karimian, K. Xiao, and J. Chandy, "DRAM based intrinsic physical unclonable functions for system level security," in *Proc. 25th Ed. Great Lakes Symp. VLSI*, May 2015, pp. 15–20.

[48] A. Schaller, W. Xiong, N. A. Anagnostopoulos, M. U. Saleem, S. Gabmeyer, S. Katzenbeisser, and J. Szefer, "Intrinsic rowhammer PUFs: Leveraging the rowhammer effect for improved security," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2017, pp. 1–7.

[49] P. Tuyls, G. J. Schrijen, B. Skoric, J. V. Geloven, N. Verhaegh, and R. Wolters, "Read-proof hardware from protective coatings," in *Proc. Cryptograph. Hardw. Embedded Syst. (CHES)*, 2006, pp. 369–383.

[50] S. Vrijaldenhoven, "Acoustical physical uncloneable functions," M.S. thesis, Dept. Math. Comput. Sci., Eindhoven Univ. Technol., Eindhoven, The Netherlands, 2004.

[51] R. S. Indeck and M. W. Müller, "Method and apparatus for fingerprinting magnetic media," U.S. Patent 53 65 586 A, Nov. 15, 1994.

[52] G. Hammouri, A. Dana, and B. Sunar, "CDs have fingerprints too," in *Proc. 11th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, 2009, pp. 348–362.

[53] National-Research-Council, *Counterfeit Deterrent Features for the Next-Generation Currency Design*. National Academies Press, 1993, pp. 1–144, doi: 10.17226/2267.

[54] G. Lenzini, S. Ouchani, P. Roenne, P. Y. A. Ryan, Y. Geng, J. Lagerwall, and J. Noh, "Security in the shell: An optical physical unclonable function made of shells of cholesteric liquid crystals," in *Proc. IEEE Workshop Inf. Forensics Secur. (WIFS)*, Dec. 2017, pp. 1–6.

[55] S. Khoshroo, "Design and evaluation of FPGA-based hybrid physically unclonable functions," M.S. thesis, Dept. Elect. Comput. Eng., Western Univ., London, ON, Canada, 2013, pp. 1–107, vol. 1281.

[56] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas, "Extracting secret keys from integrated circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 10, pp. 1200–1205, Oct. 2005.

[57] L. Lin, D. Holcomb, D. K. Krishnappa, P. Shabadi, and W. Burleson, "Low-power sub-threshold design of secure physical unclonable functions," in *Proc. 16th ACM/IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, 2010, pp. 43–48.

[58] NXP. (2018). *Secure Storage With SRAM PUF on NXP LPC54S0xx*. AN12292. [Online]. Available: https://www.nxp.com/docs/en/application-note/AN12292.pdf

[59] Microsemi, "Using SRAM-PUF system service in SmartFusion2—Libero SoC v11.7," Appl. Note AC434, 2016, pp. 1–19.

[60] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proc. 44th ACM/IEEE Design Autom. Conf.*, Jun. 2007, pp. 9–14.

[61] R. Anderson and M. Kuhn, "Tamper resistance: A cautionary note," in *Proc. 2nd USENIX Workshop Electron. Commerce*, vol. 2, 1996, p. 1.

[62] R. J. Anderson and M. G. Kuhn, "Low cost attacks on tamper resistant devices," in *Proc. 5th Int. Workshop Secur. Protocols*. Berlin, Germany: Springer, 1998, pp. 125–136.

[63] D. Nedospasov, J.-P. Seifert, C. Helfmeier, and C. Boit, "Invasive PUF analysis," in *Proc. Workshop Fault Diagnosis Tolerance Cryptogr.*, Aug. 2013, pp. 30–38.

[64] K. Lounis and M. Zulkernine, "Security analysis of PUF-based authentication protocols for Internet of Things," *ACM, Digit. Threats, Res. Pract.*, pp. 1–33, 2021, doi: 10.1145/3487060.

[65] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *Proc. 25th Int. Conf. Comput. Aided Verification*, 2013, pp. 696–701.

[66] K. Lounis and M. Zulkernine. (2020). *Tamarin Specification Code for the Proposed PUF-Based T2T Authentication Protocol*. Github. [Online]. Available: https://github.com/KarimLounis/T2T_protocol/blob/master/T2TC1.spthy

[67] K. Lounis and M. Zulkernine, "Attacks and defenses in short-range wireless technologies for IoT," *IEEE Access*, vol. 8, pp. 88892–88932, 2020.

[68] K. Lounis and M. Zulkernine, "WPA3 connection deprivation attacks," in *Proc. 14th Int. Conf. Risks Secur. Internet Syst.*, vol. 12026, 2019, pp. 164–176.

[69] K. Lounis and M. Zulkernine, "Bad-token: Denial of service attacks on WPA3," in *Proc. 12th Int. Conf. Secur. Inf. Netw. (SIN)*, 2019, pp. 1–8.

[70] K. Lounis and M. Zulkernine, "Exploiting race condition for Wi-Fi denial of service attacks," in *Proc. 13th Int. Conf. Secur. Inf. Netw.*, İstanbul, Turkey, Nov. 2020, pp. 1–8.

[71] K. Lounis, "Security of wireless short-range technologies and an authentication protocol for IoT," Ph.D. dissertation, School Comput., Queen's Univ., Kingston, ON, Canada, 2020, pp. 1–323. [Online]. Available: https://qspace.library.queensu.ca/handle/1974/28649

[72] U. Chaterjee, D. Mukhopadhyay, and R. S. Chakraborty, "3PAA: A private PUF protocol for anonymous authentication," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 756–769, 2021.

[73] A. Braeken, "PUF-based authentication and key exchange for Internet of Things," in *IoT Security: Advances in Authentication*. Hoboken, NJ, USA: Wiley, 2020, ch. 10, doi: 10.1002/9781119527978.ch10.

[74] C. Benzaid, K. Lounis, A. Al-Nemrat, N. Badache, and M. Alazab, "Fast authentication in wireless sensor networks," *Future Gener. Comput. Syst.*, vol. 55, pp. 362–375, Feb. 2016.

[75] S. Mauw and V. Bos, "Drawing message sequence charts with LATEX," *TUGBoat J.*, vol. 22, nos. 1–2, pp. 87–92, 2001.

[76] K. Lounis, "PUF security: Reviewing the validity of spoofing attack against safe is the new smart," Cryptol. ePrint Archive, 2021. [Online]. Available: https://eprint.iacr.org/2021/985.pdf

KARIM LOUNIS received the first master's degree in networks and distributed systems from the University of Science and Technology Houari Boumediene (USTHB), Algeria, in 2013, the second master's degree in security of information systems from the University of East-Paris Creteil (UPEC), France, in 2014, and the Ph.D. degree in computer science from Queen's University, Canada, in 2020. After completing his second master's degree, he worked as a Research Assistant on information security with the system security group (CISPA—Center for IT-Security, Privacy and Accountability), Saarland University, Germany, from 2014 to 2015, and then with the security and trust of software systems group (SnT—Interdisciplinary Centre for Security, Reliability and Trust), University of Luxembourg, Luxembourg, from 2015 to 2017. He is currently an Assistant Researcher in network security at the School of Computing, Queen's University. His research interests include information security, networks security, and the IoT security.

MOHAMMAD ZULKERNINE (Senior Member, IEEE) is currently a Professor and Canada Research Chair with the School of Computing, Queen's University, Canada, where he leads the Queen's Reliable Software Technology (QRST) Research Group. In 2003, he joined Queen's and spent his sabbatical as a Visiting Professor at the University of Trento, Italy, and a Researcher at Irdeto Canada. His current research interests include building reliable and secure software systems and he has extensive publications in this area. He has led major research projects supported by a number of provincial and federal agencies and industry partners. He is a Senior Member of ACM and a Licensed Professional Engineer in the province of Ontario, Canada. He has been at leadership positions, such as the general chair, the organizing chair, and the program chair of many major research conferences and workshops (More information about Dr. Zulkernine are available at https://research.cs.queensu.ca/home/mzulker/).