Concealing KETJE: A Lightweight PUF-based Privacy Preserving Authentication Protocol

Gerben Geltink

Institute for Computing and Information Sciences,
Radboud University, Nijmegen, The Netherlands
g.geltink@gmail.com

Abstract. In this paper, we focus on the design of a novel authentication protocol that preserves the privacy of embedded devices. A Physically Unclonable Function (PUF) generates challenge-response pairs that form the source of authenticity between a server and multiple devices. We rely on Authenticated Encryption (AE) for confidentiality, integrity and authenticity of the messages. A challenge updating mechanism combined with an authenticate-before-identify strategy is used to provide privacy. The major advantage of the proposed method is that no shared secrets need to be stored into the device's non-volatile memory. We design a protocol that supports server authenticity, device authenticity, device privacy, and memory disclosure. Following, we prove that the protocol is secure, and forward and backward privacy-preserving via game transformations. Moreover, a proof of concept is presented that uses a 3-1 Double Arbiter PUF, a concatenation of repetition and BCH error-correcting codes, and the AE-scheme KETJE. We show that our device implementation utilizes 8,305 LUTs on a 28 nm Xilinx Zynq XC7Z020 System on Chip (SoC) and takes only 0.63 ms to perform an authentication operation.

Keywords: Privacy-preserving authentication protocol, Physically Unclonable Function, Authenticated Encryption, SoC, FPGA.

1 Introduction

Nowadays, RFID-technology and the Internet of Things (IoT) are hot topics due to the increasing desire to simplify our everyday lives via the use of pervasive devices. Hence, we see a shift from simple identification of devices towards complex authentication protocols, in which a challenging feature to implement is the protection of the entity's privacy. Because these entities belong to individuals who may want to preserve their privacy, we notice a shift on focusing more on privacy-preserving authentication protocols [6]. With the use of state-of-the-art cryptographic techniques, device-to-server authentication can be implemented while protecting the privacy with respect to outsiders.

One solution is to use symmetric key cryptography, with a pre-shared key and a key-updating mechanism in order to randomize device credentials at each successful authentication. However, storing these keys requires non-volatile

memory which is easily compromised by an attacker. Another option is to use PUFs, physical entities that are similar to algorithmic one-way functions. PUFs act on challenges, returning noisy PUF responses that are close enough between equal challenges on the same PUF instance, but far enough between different instances. As a result, one only needs to store a challenge which, similar to the aforementioned construction, is updated on a successful authentication. The strength of this construction is that these challenges are not secret and can safely be stored in non-volatile memory. By using a PUF, one needs to implement a Fuzzy Extractor (FE) [7] that can produce an unpredictable key from the noisy PUF responses. On top of that, a FE provides for the recovery of old PUF responses from fresh PUF responses using error-correcting codes.

This research focusses on integrating a single, compact mode, namely Authenticated Encryption, into a PUF-based privacy-preserving authentication protocol. In contrast to [2], we construct a secure FE and abstain from using a pre-shared key between server and devices. With this, we hope to improve overall efficiency of the protocol.

The remainder of this paper is structured as follows. Section 2 describes the related work and our contributions. In Section 3 we introduce the protocol, describing the security considerations and overall design. Following, in Section 4 we theoretically support the protocol by proving the security and privacy of the protocol. In Section 5 we give a proof of concept of the proposed protocol, showing that a concrete software/hardware realization is possible. Then, in Section 6 we present the results, giving an analysis of the implemented PUF as well as giving the performance and a comparison to relevant previous works, i.e. [2,15]. Finally, in Section 7 we conclude the paper.

2 Related Work

Many PUF based protocols have been proposed [2, 9, 15]. Herrewegge et al. propose using a reverse Fuzzy Extractor, putting the computationally less complex generation procedure in the device, and the more complex reproduction procedure on the server [9]. However, the proof of concept is subjected to a PRNG exploitation [6]. Moriyama et al. propose a provably secure privacy-preserving authentication protocol that uses a different PUF response at every authentication, and thus changing the device credential after every successful authentication [15]. Aysu et al. [2] propose a provably secure protocol based on [9, 15]. While the authors present the first effort to describe an end-to-end design and evaluation of a provable secure privacy-preserving PUF-based authentication protocol, the interleaved FE construction is vulnerable to linear equation analysis [2, p. 12]. Moreover, the authors use an additional pre-shared key that does not increase the entropy of the communication messages.

We summarize the contributions of this research as follows: (i) We introduce a novel PUF-based privacy-preserving authentication protocol using AE. (ii) Further, we prove that the protocol is mathematically secure, and forward

and backward privacy-preserving. (iii) Finally, we present a proof of concept of the device on a development board and the server on a PC.

3 Protocol Design

In this section, we present the novel protocol design. Before doing that, we describe the notation that is used throughout this paper and we describe the security considerations for the protocol.

3.1 Notation

We denote the security level as k (in bits). $A, A', A^1 \in \mathcal{A} \subseteq \{0,1\}^*$ denote three distinct binary strings. B_i denotes the i'th bit of B. $\langle C, D \rangle$ denotes a tuple of strings C and D. $\mathbf{HD}(Y, Y')$ denotes the Hamming distance between two vectors $Y, Y' \leftarrow \mathcal{Y}$ of the same length. $\mathbf{HW}(Y)$ denotes the Hamming weight of vector $Y \leftarrow \mathcal{Y}$. $\mathbf{H}(Y)$ denotes the Shannon entropy of a discrete random variable $Y \leftarrow \mathcal{Y}$. $\tilde{\mathbf{H}}_\infty(Y)$ denotes the min-entropy of a random variable $Y \in \mathcal{Y}$. The entropy of a binary variable $Y \leftarrow \{0,1\}^l$ with probabilities $\mathbf{Pr}(Y_i = 1) = p$ and $\mathbf{Pr}(Y_i = 0) = 1 - p$ $(0 \leq i < l)$ is defined in the binary entropy function:

$$\mathbf{h}(p) = -p \log_2(p) - (1-p) \log_2(1-p). \tag{1}$$

Besides, $Y \leftarrow \mathbf{puf}_i(X) \in \mathcal{P}$ denotes a PUF instance $\mathbf{puf}_i \in \mathcal{P}$ which takes challenge X and produces response Y. Here, the $\mathcal{P}$ denotes the PUF class that contains all PUF instances of a PUF construction. A Fuzzy Extractor (FE) consists of two algorithms: a key generation algorithm **Gen** and a reconstruction algorithm **Rec**. **Gen** takes as input variable Z and outputs key R and helper data H, **Rec** recovers the key R from input variable Z' and helper data H. An AE-scheme with associated data (AEAD-scheme) is a three-tuple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ [16]. Associated to Π are sets of strings $\mathcal{N} \subseteq \{0,1\}^*$ indicating the nonce, $\mathcal{M} \subseteq \{0,1\}^*$ indicating the message and $\mathcal{A}^{\mathcal{D}} \subseteq \{0,1\}^*$ indicating the associated data. The key space $\mathcal{K}$ is a finite nonempty set of strings. The encryption algorithm $\mathcal{E}$ is a deterministic algorithm that takes strings $K \in \mathcal{K}$, $N \in \mathcal{N}$, $M \in \mathcal{M}$ and $A \in \mathcal{A}^{\mathcal{D}}$ and returns string $\langle C, T \rangle = \mathcal{E}_K^{N,A}(M)$. The decryption algorithm $\mathcal{D}$ is a deterministic algorithm that takes strings $K \in \mathcal{K}$, $N \in \mathcal{N}$, $A \in \mathcal{A}^{\mathcal{D}}$, $C \in \{0,1\}^*$ and $T \in \{0,1\}^*$ and returns $\mathcal{D}_K^{N,A}(\langle C, T \rangle)$, which is either a string in $\mathcal{M}$ or the distinguished symbol INVALID. We require that $\mathcal{D}_K^{N,A}(\mathcal{E}_K^{N,A}(M)) = M$ for all $K \in \mathcal{K}$, $N \in \mathcal{N}$, $M \in \mathcal{M}$ and $A \in \mathcal{A}^{\mathcal{D}}$.

3.2 Security Considerations

The security considerations we take are based on assumptions made in earlier work on lightweight authentication protocols [2, 9, 15].

Devices are enrolled in a secure environment using a one-time interface. Following, a trusted server and a number of devices will authenticate each other

while devices need to remain anonymous. For the communication, we consider that our channel is ideal, i.e. no errors will occur in the channel. After enrollment, the server remains trusted but devices are subjected to an attacker. The attacker may not know the identity of a device such that the device cannot be tracked.

We identified that the attacker may have two goals, i.e. the attacker may want to: (i) impersonate a device which will result in a violation of the security; (ii) trace devices in between authentications which will result in a violation of the privacy. The power of the attacker is that he can change all communication between the server and devices. Moreover, he may know the result of the authentication and can access the non-volatile memory of the devices, which he cannot modify (which is needed for the privacy-preserving proof)[1]. He can also not perform implementation attacks on the device and the server or reverse engineer the PUF such that he can predict PUF responses. Also, he does not have access to the intermediate registers on the device and cannot physically trace every device in between authentications. Furthermore, the attacker is not able to use other (non-cryptographic) mechanisms to identify a device [11].

3.3 Protocol

The setup phase of the proposed protocol is illustrated in Protocol 1, the authentication phase is illustrated in Protocol 2. The protocol is based on a PUF that produces noisy, but recoverable, responses on equal challenges due to the unique physical characteristics of the IC [13]. Because of this behavior, the PUF is identifiable from other PUFs. A FE can extract a key from this noisy data produced by the PUF using helper data generated from a previous key-extraction [7]. However, the recovery procedure is of a higher complexity than the generation of the helper data that is used for this reconstruction. A reverse FE reverses this behavior by placing the helper data generation in the device and the more complex reconstruction in the server [9]. In order to preserve privacy, the device credential is updated every successful authentication, which results in fresh PUF responses, and thus fresh keys.

The setup phase (Protocol 1) works as follows. In a trusted environment, the server produces a random challenge X^1. The device uses this challenge to produce a PUF response Y^1 which is being sent to the server. The challenge is being stored in the device non-volatile memory. The server stores the response in a database on index n, indicating the number of the device. Notice that the response is stored at Y and Y^{old} in order to prevent desynchronization, which occurs when there is a loss of communication in the transmission of T^2 in the authentication phase and only Y is stored.

The authentication phase (Protocol 2) works as follows. First, the server generates an unpredictable challenge A and sends this to the device. The device

[1] A modification of challenge X in non-volatile memory does not break the security of the protocol, only the theoretical privacy preservation because an attacker can distinguish a device with modified challenge X (cannot successfully authenticate) from a device with unmodified challenge X (can successfully authenticate).

Server $\mathcal{S}(\{\langle Y, Y^{old}\rangle\}_n)$		**Device** $\mathbf{Dev}_i(\mathbf{puf}_i(\cdot), X)$
$X^1 \leftarrow \mathbf{TRNG}$		
	$\xrightarrow{X^1}$	
		$Y^1 \leftarrow \mathbf{puf}_i(X^1)$
		$X := X^1$
	$\xleftarrow{Y^1}$	
$\langle Y, Y^{old}\rangle_n := \langle Y^1, Y^1\rangle$		
$n := n + 1$		

Protocol 1. Setup phase.

Server $\mathcal{S}(\{\langle Y, Y^{old}\rangle\}_n)$		**Device** $\mathbf{Dev}_i(\mathbf{puf}_i(\cdot), X)$
$A \leftarrow \mathbf{TRNG}$		
	$\xrightarrow{A}$	
		$Y^{1'} \leftarrow \mathbf{puf}_i(X)$
		$\langle R, H\rangle \leftarrow \mathbf{FE.Gen}(Y^{1'})$
		$X^2 \leftarrow \mathbf{TRNG}$
		$Y^2 \leftarrow \mathbf{puf}_i(X^2)$
		$N \leftarrow \mathbf{TRNG}$
		$\langle C^1, T^1\rangle \leftarrow \mathcal{E}_R^{N\|0,A}(Y^2)$
	$\xleftarrow{\langle H, N, C^1, T^1\rangle}$	
$T^2 \leftarrow \mathbf{TRNG}$		
for $0 \leq i < n$:		
$R = \mathbf{FE.Rec}(Y, H)$		
if $Y^2 \leftarrow \mathcal{D}_R^{N\|0,A}(\langle C^1, T^1\rangle)$:		
$\langle \cdot, T^2\rangle \leftarrow \mathcal{E}_R^{N\|1,A}(\cdot)$		
$\langle Y, Y^{old}\rangle_{\mathbf{num}} := \langle Y^2, Y\rangle$		
"if no device was authenticated" :		
"repeat search with old values"		
	$\xrightarrow{T^2}$	
		$\langle \cdot, T^{2'}\rangle \leftarrow \mathcal{E}_R^{N\|1,A}(\cdot)$
		if $T^{2'} == T^2$:
		$X := X^2$

Protocol 2. Authentication phase. $|A|, |H|, |N|, |C^1|, |T^1|, |T^2| \geq k$ and PUF responses Y should contain enough entropy w.r.t. H s.t. $|R| \geq k$.

uses the challenge X stored in its non-volatile memory to produce a PUF response $Y^{1'}$. From this PUF response, helper data H and an unpredictable key R is generated using the FE's generation procedure **FE.Gen**. Consecutively, a new challenge X^2 is randomly generated by the device such that it can be updated

on a successful authentication. This challenge is fed to the device's PUF in order to receive a new PUF response Y^2. Following, a nonce N is randomly generated such that the PUF response can be encrypted using the AEAD-scheme. The resulting cipher-text C^1, its tag T^1 and the nonce N will be sent to the server. The server performs an exhaustive search over the database, recovering a key for each index. These keys are used to try to decrypt the cipher-text C^1 using the tag T^1, challenge A and nonce N. If there is a successful authentication, the server produces another tag T^2 using $\mathcal{E}$, but with nonce $N^2 \parallel 1$ instead of $N^2 \parallel 0$ in order to create another instance of $\mathcal{E}$. This tag is sent to the device. Moreover, the server updates the old PUF-response Y with the new PUF response Y^2. If there were no successful authentications, the server repeats the procedure over the previous PUF responses in the database. If after this there were still no successful authentications, the server responds with a random value for T^2. Finally, the device checks the tag T^2 with its own produced tag in order to reveal whether the authentication succeeded. If the authentication succeeded, the device updates the old challenge X with the new challenge X^2.

4 Security Analysis

In this section, we describe the security analysis of the proposed protocol. We first present the security model and the formal security definitions before proving the security, and forward and backward privacy.

4.1 Security Model

The security model is composed of the communication model, the security experiment and the privacy experiment.

Communication Model We take one trusted server $\mathcal{S}(\{\langle Y, Y^{old}\rangle\}_n)$ with n devices $\mathbf{Dev}_i(\mathbf{puf}_i(\,\cdot\,), X)$. Here, the set of n devices is denoted as $\Delta := \{\mathbf{Dev}_0, \mathbf{Dev}_1, \ldots, \mathbf{Dev}_{n-1}\}$. We denote the security parameter as k.

Following [2, 15], devices will be enrolled in a trusted environment using a one-time interface, this happens in a setup phase using a setup algorithm $\mathbf{Setup}(1^k)$ which generates public parameter P and shared-secret Y. Here P denotes all the public parameters available to the environment and Y denotes the secret PUF response. During the authentication phase, the server $\mathcal{S}$ remains trusted, however, the devices Δ and the communication channel will be subjected to the actions of an attacker. At the end of the authentication phase, both the server and the device will output acceptance ($B_0 = 1$) or rejection ($B_0 = 0$) as result of the authentication.

We call the sequence of communication between the server and the device a session, which is distinguished by a session identifier I, the transcript of the authentication phase. Whenever the communication messages generated by the server and the device are honestly transferred until they authenticate each other, we call that a session has a matching session. The correctness of the proposed

authentication protocol is that the server and the device always accept the session if the session has the matching session.

Security Following [2,15], we consider the canonical security level for authentication protocols, namely the resilience to the man-in-the-middle attack. This means that power of an attacker is modeled by letting the attacker control all communication between server and devices. Supplementary to the security requirement of resilience to man-in-the-middle attacks, we permit the attacker to access the information stored in the non-volatile memory of the device in between sessions.

Experiment 1 illustrates the security evaluation on a theoretical level. In this experiment, $\mathbf{Exp}^{\mathbf{Sec}}_{\Psi,\mathcal{A}}(k)$ denotes the security experiment between the proposed protocol Ψ and an attacker $\mathcal{A}$ with security parameter k.

$\mathbf{Exp}^{\mathbf{Sec}}_{\Psi,\mathcal{A}}(k)$

$\langle P, Y\rangle \leftarrow \mathbf{Setup}(1^k)$
$\langle \mathbf{Dev}_i, I'\rangle \leftarrow \mathcal{A}^{\langle \mathbf{Launch},\mathbf{SendServer},\mathbf{SendDev},\mathbf{Result},\mathbf{Reveal}\rangle}(P, \mathcal{S}, \Delta)$
$B_0 := \mathbf{Result}(\mathbf{Dev}_i, I')$
Output B_0

Experiment 1. Security experiment.

After the setup phase, and thus after receiving $\langle P, \mathcal{S}, \Delta\rangle$, the attacker $\mathcal{A}$ can query the server $\mathcal{S}$ and the device $\mathbf{Dev}_i$ with the oracle queries $\mathcal{O} := \langle \mathbf{Launch}, \mathbf{SendServer}, \mathbf{SendDev}, \mathbf{Result}, \mathbf{Reveal}\rangle$, where

- $\mathbf{Launch}(1^k)$: launch the server $\mathcal{S}$ to start a new session with security parameter k;
- $\mathbf{SendServer}(M)$: send an arbitrary message M to the server $\mathcal{S}$;
- $\mathbf{SendDev}(\mathbf{Dev}_i, M)$: send an arbitrary message M to device $\mathbf{Dev}_i \in \Delta$;
- $\mathbf{Result}(G, I)$: output whether the session I of G is accepted or not where $G \in \{\mathcal{S}, \Delta\}$;
- $\mathbf{Reveal}(\mathbf{Dev}_i)$: output all the information stored in the non-volatile memory of $\mathbf{Dev}_i$.

The advantage of attacker $\mathcal{A}$ against Ψ is defined as:

$$\mathbf{Adv}^{\mathbf{Sec}}_{\Psi,\mathcal{A}}(k) := \mathbf{Pr}(\mathbf{Exp}^{\mathbf{Sec}}_{\Psi,\mathcal{A}}(k) \rightarrow 1 \mid \text{“}I \text{ of } G \text{ has no matching session”}) \quad (2)$$

We define security of an authentication protocol as follows:

Definition 1 (Security). *An authentication protocol Ψ holds the security against man-in-the-middle attacks with memory compromise if for any probabilistic polynomial time attacker $\mathcal{A}$, $\mathbf{Adv}^{\mathbf{Sec}}_{\Psi,\mathcal{A}}(k)$ is negligible in k (for large enough k).*

Privacy Following [2,15], we define the privacy using indistinguishability between two devices. Here, an attacker selects two devices and tries to distinguish the communication, and thus the identification, between the two devices.

We use the privacy experiment between an attacker $\mathcal{A} := \langle \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3 \rangle$ as illustrated in Experiment 2.

$\mathbf{Exp}^{\mathrm{IND}^*-b}_{\Psi,\mathcal{A}}(k)$

$\langle P, Y \rangle \leftarrow \mathbf{Setup}(1^k)$
$\langle \mathbf{Dev}^*_0, I^{0'}, \mathbf{Dev}^*_1, I^{1'} \rangle \leftarrow \mathcal{A}^{\mathcal{O}}_1(P, \mathcal{S}, \Delta)$
$b \leftarrow \{0, 1\}$
$\Delta' := \Delta \setminus \langle \mathbf{Dev}^*_0, \mathbf{Dev}^*_1 \rangle$
$\psi_0 \leftarrow \mathbf{Execute}(\mathcal{S}, \mathbf{Dev}^*_0)$
$\psi_1 \leftarrow \mathbf{Execute}(\mathcal{S}, \mathbf{Dev}^*_1)$
$\langle I^{0''}, I^{1''} \rangle \leftarrow \mathcal{A}^{\mathcal{O}}_2(\mathcal{S}, \Delta', \mathcal{I}(\mathbf{Dev}^*_b), \psi_0, I^{0'}, \psi_1, I^{1'})$
$\psi'_0 \leftarrow \mathbf{Execute}(\mathcal{S}, \mathbf{Dev}^*_0)$
$\psi'_1 \leftarrow \mathbf{Execute}(\mathcal{S}, \mathbf{Dev}^*_1)$
$B_0 \leftarrow \mathcal{A}^{\mathcal{O}}_3(\mathcal{S}, \Delta', \psi'_0, I^{0''}, \psi'_1, I^{1''})$
Output B_0

Experiment 2. Privacy experiment in which it is allowed to communicate with two devices.

After the setup-phase, and similar to the security experiment, the attacker interacts with the server and two randomly chosen devices through the oracle queries $\mathcal{O}$. These two devices $\mathbf{Dev}^*_0, \mathbf{Dev}^*_1$ are being sent to the challenger who flips a coin to choose with which device the attacker will communicate anonymously. This anonymous communication is accomplished by adding a special identity $\mathcal{I}$ which honestly transfers the communication messages between $\mathcal{A}$ and $\mathbf{Dev}^*_b$.

It is trivial that the attacker can trace devices in case the **Reveal** query is issued when there are no successful authentications. Hence, we provide resynchronization before and after the anonymous access by adding the **Execute** query. This query does a normal protocol execution between the server $\mathcal{S}$ and the device $\mathbf{Dev}^*_i$. During this execution, the attacker cannot modify the communication, however the transcript ψ_i is delivered to the attacker. Once an honest protocol execution is finished, no one can trace the device even if the information from the non-volatile memory before and after the session is continuously leaked to the attacker. The advantage of the attacker is defined as:

$$\mathbf{Adv}^{\mathrm{IND}^*}_{\Psi,\mathcal{A}}(k) := |\mathbf{Pr}(\mathbf{Exp}^{\mathrm{IND}^*-0}_{\Psi,\mathcal{A}}(k) \rightarrow 1) - \mathbf{Pr}(\mathbf{Exp}^{\mathrm{IND}^*-1}_{\Psi,\mathcal{A}}(k) \rightarrow 1)| \qquad (3)$$

We define privacy of an authentication protocol as follows:

Definition 2 (Privacy). *An authentication protocol Ψ holds forward and backward privacy if for any probabilistic polynomial time attacker $\mathcal{A}$, $\boldsymbol{Adv}^{\mathrm{IND}^*}_{\Psi,\mathcal{A}}(k)$ is negligible in k (for large enough k).*

4.2 Formal Security Definitions

We define Physically Unclonable Functions, the Fuzzy Extractor and the AEAD-scheme.

Physical Unclonable Functions We define PUFs using the definition described in [2, p. 24].

We denote the set of all possible challenges X which can be applied to an instance of $\mathcal{P}$ as $\mathcal{X}_\mathcal{P}$. We say that the PUF class $\mathcal{P}$ is a $\langle n, l, d, h, \epsilon\rangle$-secure PUF class $\mathcal{P}$ if the following conditions hold:

1. For any PUF instance $\mathbf{puf}_i(\,\cdot\,) \leftarrow \mathcal{P}$ and for any input $X \leftarrow \mathcal{X}_\mathcal{P}$,
$$\mathbf{Pr}(\mathbf{HW}(Y \leftarrow \mathbf{puf}_i(X), Y' \leftarrow \mathbf{puf}'_i(X)) < d) = 1 - \epsilon$$
2. For any two PUF instances $\mathbf{puf}_i(\,\cdot\,), \mathbf{puf}_j(\,\cdot\,) \leftarrow \mathcal{P}$, where $i \neq j$ and for any input $X \leftarrow \mathcal{X}_\mathcal{P}$,
$$\mathbf{Pr}(\mathbf{HW}(Y \leftarrow \mathbf{puf}_i(X), Y' \leftarrow \mathbf{puf}_j(X)) > d) = 1 - \epsilon$$
3. For any PUF instance $\mathbf{puf}_i(\,\cdot\,) \leftarrow \mathcal{P}$ and for any two inputs $X^a, X^b \leftarrow \mathcal{X}_\mathcal{P}$, where $a \neq b$,
$$\mathbf{Pr}(\mathbf{HW}(Y \leftarrow \mathbf{puf}_i(X^a), Y' \leftarrow \mathbf{puf}_i(X^b)) > d) = 1 - \epsilon$$
4. For any PUF instance $\mathbf{puf}_i(\,\cdot\,) \leftarrow \mathcal{P}$ and for any input $X^a \leftarrow \mathcal{X}_\mathcal{P}$,
$$\mathbf{Pr}(\tilde{\mathbf{H}}_\infty(Y \leftarrow \mathbf{puf}_i(X^a) \mid \{Y^j \leftarrow \mathbf{puf}_j(X^b)\}_{0\leq j<n,\ 0\leq b<l,\ i\neq j,\ a\neq b}) > h) = 1-\epsilon$$

These conditions provide that the intra-distance $\mathcal{D}_\mathcal{P}^{\text{intra}}$ is smaller than d, the inter-distance $\mathcal{D}_\mathcal{P}^{\text{inter}}$ (from two metrics) is larger than d and the min-entropy of the PUF class $\mathcal{P}$ is always larger than h.

Definition 3 ($\langle n, l, d, h, \epsilon\rangle$-secure PUF class $\mathcal{P}$). *A PUF class $\mathcal{P}$ satisfies $\langle n, l, d, h, \epsilon\rangle$-secure PUF class $\mathcal{P}$ if all the above conditions hold.*

Fuzzy Extractor We define a Fuzzy Extractor using the definition described in [2, p. 24].

A $\langle d, h, \epsilon\rangle$-FE consists of two algorithms: a key generation algorithm **Gen** and a reconstruction algorithm **Rec**. **Gen** takes as input variable Z and outputs key R and helper data H. For correctness, **Rec** recovers the key R from input variable Z' and helper data H if the **HD** between Z and Z' is at most d. The FE provides unpredictable outputs if the min-entropy of input Z is at least h. In that case, R is statistically ϵ-close to a uniformly random variable in $\{0,1\}^k$, even if the helper data H is disclosed.

Definition 4 ($\langle d, h, \epsilon\rangle$-secure FE). *A FE satisfies $\langle d, h, \epsilon\rangle$-secure FE if the following conditions hold:*

1. $\mathbf{Pr}(R := \mathbf{Rec}(Z', H) \mid \langle R, H\rangle = \mathbf{Gen}(Z),\ \mathbf{HD}(Z, Z') \leq d) = 1$
2. *If $\tilde{\mathbf{H}}_\infty(Z) \geq h$, $\langle R, H\rangle = \mathbf{Gen}(Z)$ is statistically ϵ-close to $\langle R', H\rangle$ where $R' \leftarrow \{0,1\}^k$ is chosen uniformly at random.*

AEAD-scheme The security of the AEAD-scheme Π is defined by the following experiment (chosen-plaintext attack) between a challenger and an attacker $\mathcal{A}$: First, the challenger randomly selects coin $b \leftarrow \{0,1\}$ and secret key $K \leftarrow \{0,1\}^k$. The challenger then prepares a truly random function **RF**. Following, the attacker $\mathcal{A}$ can adaptively issue an oracle query to the challenger to obtain a response of a function. If $b = 1$ and the attacker $\mathcal{A}$ sends message $M \leftarrow \{0,1\}^*$, challenge $N \leftarrow \{0,1\}^k$ and associated data $A \leftarrow \{0,1\}^*$, the challenger responds with $\langle C, T\rangle = \mathcal{E}_K^{N,A}(M)$. On the other hand, if $b = 0$, the challenger inputs the message $M \leftarrow \{0,1\}^*$, challenge $N \leftarrow \{0,1\}^k$ and associated data $A \leftarrow \{0,1\}^*$ to **RF** and responds with its result $\langle C', T'\rangle$. Finally, the attacker outputs a guess b'. If $b' = b$, the attacker wins the experiment. Similarly, this construction can be applied to test the security of the decryption algorithm $\mathcal{D}_K^{N,A}(\langle C, T\rangle)$.

The advantage of the attacker to win the experiment is defined by $\mathbf{Adv}_{\mathcal{A}}^{\Pi}(k) = |2 \cdot \mathbf{Pr}(b' = b) - 1|$.

Definition 5 (ϵ-secure AEAD-scheme). *An AEAD-scheme is an ϵ-secure AEAD-scheme if for any probabilistic polynomial time attacker $\mathcal{A}$, $\mathbf{Adv}_{\mathcal{A}}^{\Pi}(k) \leq \epsilon$.*

4.3 Security Proofs

In this section, we give the security proof and privacy proof for the proposed protocol. We follow the proof by game transformations as described in [2,15].

Theorem 1 (Security). *Let PUF instance $\mathbf{puf}^* \leftarrow \mathcal{P}$ be a $\langle n, l, d, h, \epsilon_1\rangle$-secure PUF, FE be a $\langle d, h, \epsilon_2\rangle$-secure FE and the AEAD-scheme be an ϵ_3-secure AEAD-scheme. Then our protocol Ψ is secure against man-in-the-middle attacks with memory compromise. Especially, we have $\mathbf{Adv}_{\Psi,\mathcal{A}}^{\mathbf{Sec}}(k) \leq l \cdot n \cdot (\epsilon_1 + \epsilon_2 + \epsilon_3)$.*

Proof. The aim of the attacker $\mathcal{A}$ is to violate the security experiment which means that either the server or a device accepts a session without it being the matching session. We call S_i the advantage that the attacker wins the game in **Game** i. We consider the following game transformations:

Game 0: This is the original game between the challenger and the attacker.

Game 1: The challenger randomly guesses the device $\mathbf{Dev}^* \leftarrow \Delta$. If the attacker does not impersonate $\mathbf{Dev}^*$ to the server, the challenger aborts the game. Thus, the attacker needs to participate in session I^* and cannot tamper with the communication.

Game 2: Assume that l is the upper bound of the number of sessions that the attacker can establish in the game. For $0 \leq j < l$, we evaluate or change the variables related to the session between the server and $\mathbf{Dev}^*$ up to the l-th session as the following games:

- **Game 2-j-1:** The challenger evaluates the output from the PUF instance $\mathbf{puf}^*$ implemented in $\mathbf{Dev}^*$ at the j-th session. If the intra-distance is larger than d, the inter-distance is smaller than d or the min-entropy of the output is smaller than h, the challenger aborts the game.
- **Game 2-j-2:** The output from the FE H is changed to a random variable.

Game 2-j-3: The output from the encryption algorithm $\mathcal{E}_R^{N\|0,A}(Y)$ of the AEAD-scheme is derived from a truly random function **RF**.

Game 2-j-4: The output from the encryption algorithm $\mathcal{E}_R^{N\|1,A}(\,\cdot\,)$ of the AEAD-scheme is derived from a truly random function **RF**.

The strategy of the security proof is to change the communication messages corresponding to the target device $\mathbf{Dev}^*$ to random variables. However, we must take care of the PUF construction and challenge-update mechanism in our protocol that updates the PUF response. Hence, we must proceed with the game transformation starting from the first invocation of device $\mathbf{Dev}^*$. The communication messages gradually change from **Game** 2-j-1 to **Game** 2-j-4, and when these are finished, we can move to the next session. This strategy is recursively applied up to the upper bound of l of the sessions that the attacker can establish.

In short, if the implemented PUF instance creates enough entropy, the FE can provide variables that are statistically close to random strings. Then, this output can be applied as a key for the AEAD-scheme which both authenticate the device as well as encrypt the next PUF response. Finally, the server can be authenticated using the AEAD-scheme without encrypting a message.

Lemma 1 (Random Guess). $S_0 = n \cdot S_1$ *(where n is the number of devices).*

Subproof. The violation of security means that there is a session which the server or device accepts while the communication is modified by the attacker. Since we assume that the number of devices is at most n, the challenger can correctly guess the related session with a probability of at least $1/n$. ◇

Lemma 2 (PUF Response). $|S_1 - S_{2\text{-}1\text{-}1}| \leq \epsilon_1$ *and* $|S_{2\text{-}(j-1)\text{-}4} - S_{2\text{-}j\text{-}1}| \leq \epsilon_1$ *for any* $1 \leq j < l$ *if the PUF instance* $\mathbf{puf}^*$ *is a* $\langle n, l, d, h, \epsilon_1\rangle$*-secure PUF.*

Subproof. We now assume that the PUF instance $\mathbf{puf}^*$ satisfies a $\langle n, l, d, h, \epsilon_1\rangle$-secure PUF in advance. This means that the intra-distance $\mathcal{D}_{\mathcal{P}}^{\text{intra}}$ is smaller than d, the inter-distance $\mathcal{D}_{\mathcal{P}}^{\text{inter}}$ is larger than d and the min-entropy of the PUF class $\mathcal{P}$ is always larger than h except the negligible probability ϵ_1. Since S_1 and $S_{2\text{-}(j-1)\text{-}4}$ assume these conditions except the negligible probability ϵ_1 and $S_{2\text{-}1\text{-}1}$ and $S_{2\text{-}j\text{-}1}$ require these conditions with probability 1, respectively, the gap between them is bounded by ϵ_1. ◇

Lemma 3 (FE Output). $\forall\ 0 \leq j < l$, $|S_{2\text{-}j\text{-}1} - S_{2\text{-}j\text{-}2}| \leq \epsilon_2$ *if the FE is a* $\langle d, h, \epsilon_2\rangle$*-secure FE.*

Subproof. From the subproof of Lemma 2, we can assume that the PUF instance $\mathbf{puf}^*$ provides enough min-entropy h. Then the property of the $\langle d, h, \epsilon_2\rangle$-secure FE guarantees that the output for the FE is statistically close to random and no attacker can distinguish the difference between the two games. ◇

Lemma 4 (Authenticated Encryption). $\forall\ 0 \leq j < l$, $|S_{2\text{-}j\text{-}2} - S_{2\text{-}j\text{-}3}| \leq \mathbf{Adv}_{\mathcal{A}}^{\Pi}(k)$ *for a probabilistic polynomial time algorithm* $\mathcal{B}$*.*

Subproof. We construct the algorithm $\mathcal{B}$ which breaks the security of our AEAD-scheme Π. $\mathcal{B}$ can access the real encryption algorithm $\mathcal{E}_R^{N\|0,A}(Y)$, the real decryption algorithm $\mathcal{D}_R^{N\|0,A}(\langle C^1, T^1\rangle)$ or the truly random function **RF**. $\mathcal{B}$ sets up all secret keys and simulates our protocol except the n-th session (the current session). When the attacker invokes the n-th session $\mathcal{B}$ sends the uniformly random distributed challenge $A \leftarrow \{0,1\}^k$ as the output of the server. When the attacker $\mathcal{A}$ sends the challenge A^* to a device $\mathbf{Dev}_i$, $\mathcal{B}$ randomly selects a nonce N and issues this to the oracle instead of the real computation of $\mathcal{E}_R^{N\|0,A}(Y)$. Upon receiving $\langle C, T\rangle$, $\mathcal{B}$ continues the computation as the protocol specification and outputs $\langle H, N, C^1, T^1\rangle$ as the device's response. When the attacker sends $\langle H^*, N^*, C^{1^*}, T^{1^*}\rangle$, $\mathcal{B}$ issues challenge A and nonce N^* to the oracle oracle and obtains either Y or the distinguished symbol INVALID.

If $\mathcal{B}$ accesses the real encryption and decryption algorithms $\langle \mathcal{E}, \mathcal{D}\rangle$, this simulation is equivalent to **Game** 2-j-2. Otherwise, the oracle query issued by $\mathcal{B}$ is completely random and this distribution is equivalent to **Game** 2-j-3. Thus we have $|S_{2\text{-}j\text{-}2} - S_{2\text{-}j\text{-}3}| \leq \mathbf{Adv}_{\mathcal{A}}^{\Pi}(k)$. ◇

Lemma 5 (Authentication). $\forall\ 0 \leq j < l$, $|S_{2\text{-}j\text{-}3} - S_{2\text{-}j\text{-}4}| \leq 2 \cdot \mathbf{Adv}_{\mathcal{A}}^{\Pi}(k)$ *for a probabilistic polynomial time algorithm* $\mathcal{B}$.

Subproof. Consider an algorithm $\mathcal{B}$ which interacts with the encryption algorithm $\mathcal{E}_R^{N\|1,A}(\ \cdot\ )$ and truly random function **RF**. $\mathcal{B}$ runs the setup procedure and simulates the protocol up to the n-th session. Similarly to the subproof of Lemma 4 when the attacker invokes the n-th session $\mathcal{B}$ sends the uniformly random distributed challenge $A \leftarrow \{0,1\}^k$ as the output of the server. $\mathcal{B}$ continues the computation as the protocol specification and outputs $\langle H, N, C^1, T^1\rangle$ as the device's response. If the attacker $\mathcal{A}$ has sent the challenge A^* to a device $\mathbf{Dev}_i$, $\mathcal{B}$ randomly selects nonce N and issues this to the oracle instead of the real computation $\mathcal{E}_R^{N\|1,A}(\ \cdot\ )$. When the attacker sends $\langle H^*, N^*, C^{1^*}, T^{1^*}\rangle$, $\mathcal{B}$ issues challenge A and nonce N^* to the oracle and obtains T^2.

If $\mathcal{B}$ accesses the real encryption algorithm $\mathcal{E}$, this simulation is equivalent to **Game** 2-j-3. Otherwise, the oracle query issued by $\mathcal{B}$ is completely random and this distribution is equivalent to **Game** 2-j-4. Thus we have $|S_{2\text{-}j\text{-}3} - S_{2\text{-}j\text{-}4}| \leq \mathbf{Adv}_{\mathcal{A}}^{\Pi}(k)$. ◇

When we transform **Game** 0 to **Game** 2-l-4, there is no advantage of the attacker to violate the security. Given the fact that the attacker knows the PUF challenge X from the device's non-volatile memory, the attacker cannot produce a corresponding PUF response. This results in the fact that the attacker cannot produce a key R which matches any of the recovered keys from the server's database. This means that the cryptogram produced by an attacker will never be accepted by the decryption algorithm of the AEAD-scheme in the server. Additionally, changing the authenticator T^2 will only prevent the device from updating its PUF challenge, this is why the server also performs an exhaustive search over the old $(j-1)$ PUF responses.

Therefore, no attacker can successfully mount the man-in-the-middle attack in our proposed protocol. □

Theorem 2 (Privacy). *Let PUF instance* $\mathbf{puf}^* \leftarrow \mathcal{P}$ *be a* $\langle n, l, d, h, \epsilon_1 \rangle$*-secure PUF, FE be a* $\langle d, h, \epsilon_2 \rangle$*-secure FE and the AEAD-scheme be an* ϵ_3*-secure AEAD-scheme. Then our protocol* Ψ *holds forward and backward privacy.*

Proof. This proof is similar to the proof of Theorem 1. However, we remark that it is important to assume that our protocol satisfies security first for privacy to hold. This is because if security does not hold, a malicious attacker might be able to desynchronize the PUF response Y of device $\mathbf{Dev}^*$ to a chosen one. In that case, even if the attacker honestly transfers the communication message between $\mathcal{I}(\mathbf{Dev}^*)$ and the server in the challenging phase the authentication result is always $B_0 = 0$ and the adversary can observe whether device $\mathbf{Dev}^*$ was selected as the challenge device.

Based on the same game transformation that was describes in the proof of Theorem 1, we continuously change the communication messages for the device $\mathbf{Dev}^*$, however, we now call this device $\mathbf{Dev}_1^*$. We do a similar game transformation for a second target device $\mathbf{Dev}_2^*$. In **Game** 1, the attacker can guess which device will be chosen by the challenger in the privacy game with probability of at least $1/n^2$. Upon continuing, the game transformation in **Game** 2 is applied to the sessions related to device $\mathbf{Dev}_1^*$ and device $\mathbf{Dev}_2^*$. Then, all the message transcripts of the **Game** transformations are changed to random variables and no biased information which identifies the challenger's coin is leaked. The information stored in the non-volatile memory of devices $\mathbf{Dev}_1^*$ and $\mathbf{Dev}_2^*$ will not disclose any information because these are updated from random sources.

Therefore, no attacker can distinguish any two devices with probability higher than $1/n^2$, hence, the proposed protocol satisfies the forward and backward privacy. □

5 Proof of Concept

In this section we present a proof of concept with security level $k = 128$ bits.

Figure 1 illustrates the system architecture of the device and server. The device is implemented on a Zedboard [1] which contains a Xilinx Zynq-7000 All Programmable System on Chip (SoC) XC7Z020-CLG484-1 [17]. The server is implemented on a Linux PC. We design the system architecture using Xilinx Vivado and Xilinx Vivado SDK.

The Zynq SoC is composed of 28 nm programmable logic and a processing system, which can both be programmed through the USB JTAG. Apart from other components, the processing system contains two ARM Cortex-A9 cores, of which only one is used to: (i) control the communication between the device and the server by reading and writing AXI-addresses from the device and sending and receiving serial data through the UART; (ii) update the PUF challenge on the device non-volatile memory by re-writing to a SD-card pugged into the Zedboard. The central communication travels through a bus, the Central Interconnect (CI), which is connected with the components on the Zedboard. Communication between the logic and the ARM-core is supported with a 32-bit AXI.

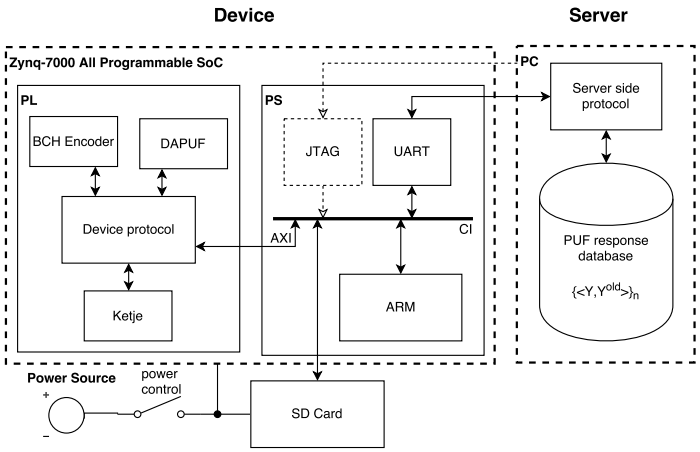

Fig. 1. System architecture of the device and server.

5.1 3-1 Double Arbiter PUF

The type of PUF used in the protocol will motivate most of the other design parameters for the rest of the protocol. For example, depending on the bit-error-probability p_e of a PUF response-bit, the inter- and intra-distances of the PUF responses, the entropy of the PUF responses ρ and the desired maximum for the failure rate of the authentications p_{fail}, both the number of PUF responses as well as the type and size of error-correcting codes is motivated.

Figure 2 illustrates the 3-1 Double Arbiter PUF (DAPUF, $\mathcal{P}_{3\text{-}1}$) as proposed by Machida et al. [12], which we implement because its characteristics are promising for the parameters of our protocol. As an example, the authors state the prediction rate is approximately 57%, which approximates a random guess (i.e. 50%). This is a considerable improvement for arbiter PUF constructions because the prediction rate of conventional arbiter PUFs is 86% [12, p. 8]. In the figure, a selector chain composes of 64 switch blocks that, depending on the input challenge bit, can switch signals from the two paths. The DAPUF is composed of three of these selector chains all acting on the same challenge X. Using an 'enable' signal E (E_L and E_R), the competition is started between the left signals E_L and the right signals E_R. For each of the combination of left- or right signals an arbiter is used to measure which path arrived first at the arbiter. After measuring these race conditions, the results are XORed to collect the 1-bit PUF response Y. By challenging the DAPUF with n different challenges, we obtain a n-bit PUF response.

In order to design a FE that produces a key with sufficient entropy, we analyze the performance parameters of the DAPUF. The authors have based the performance results on a Xilinx Virtex-7 device. Because the architecture of our SoC is similar to the Xilinx 7-series Field Programmable Gate Arrays (FPGAs),

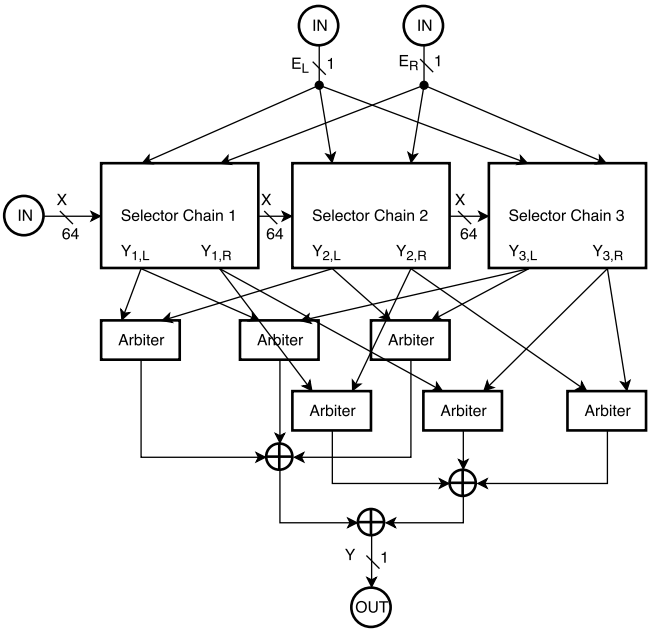

Fig. 2. 3-1 DAPUF as proposed by Machida et al. [12]. $\bigoplus$ denotes a bitwise XOR, (IN) denotes the input of the DAPUF and (OUT) denotes the output of the DAPUF.

we take their performance as a starting point for our design. According to [12], the steadiness is approximately 12%, which means that the bit-error-probability p_e is 0.12 [12]. The average uniqueness of $\mathcal{P}_{3\text{-}1}$ is approximately 50%, which is close to ideal. Finally, the authors achieved a randomness of approximately 54%, meaning that the probability that a response bit is '1' is $\mathbf{Pr}(Y_i = 1) = 0.54$. Using Formula 1 we calculate the entropy of the PUF responses ρ :

$$\rho = -0.54 \log_2(0.54) - 0.46 \log_2(0.46) = 0.9954 \tag{4}$$

5.2 Reverse Fuzzy Extractor

In order to be able to recover the PUF responses, we use a concatenation of error-correcting codes as introduced by Bosch et al. [5], which is a technique to increase the correction rate while minimizing the computational overhead. Our proposed reverse FE uses a concatenation code of a repetition code and a BCH code. The aim is to construct a 128-bit key from the DAPUF responses with quality $\langle p_e = 0.12, \rho = 0.9954 \rangle$. Also, we aim for a fail rate of $p_{\text{fail}} = 10^{-6}$, which is considered an acceptable fail rate for standard performance levels [13].

The probability that a received codeword of n bits has more than t errors is given by [5, 8]:

$$\mathbf{Pr}(\text{“}{>}t \text{ errors”}) = 1 - \sum_{i=0}^{t} \binom{n}{i} p_e^i (1-p_e)^{n-i}, \tag{5}$$

where p_e is the bit-error-probability. When using a $C_{\text{REP}}(5, 1, 2)$ repetition code, we can decrease the bit-error-probability $p_e = 0.12$ to $p_{e,\text{REP}} = 0.01432$ ($t = 2$, $n = 5$). Using a $C_{\text{BCH}}(255, 139, 15)$ BCH code on top of that further decreases the bit-error-probability $p_{e,\text{REP}} = 0.01432$ to a fail rate $p_{\text{fail}} = 1.176 \cdot 10^{-6}$ ($t = 15$, $n = 255$), which we consider sufficient. As a result, we use 1275 PUF responses on 64-bit PUF challenges, of which 40 bits are used for the challenge that is stored in the device non-volatile memory, 12 bits are used to obtain the 1275 unique PUF responses and 12 bits are used to produce random numbers, including a seed that is updated at the beginning of every authentication. In order to obtain these responses, we diffuse both sets of 12 challenge bits over the challenge space such that one set is updated every clock cycle using a linear-feedback shift register (LFSR) and the other is fixed to a constant value. As a consequence, every unique 40-bit (stored) challenge produces unique 64-bit PUF challenges, and thus produces unpredictable PUF responses. For the 1275 unique PUF responses we start the LFSR with a fixed value each authentication, whereas for the random number responses we start the LFSR with the updated (random) seed.

In order to analyze whether this construction leaves enough entropy in the key, we calculate the entropy losses in the communicated helper data. When using a $C_{\text{REP}}(5, 1, 2)$ repetition code on 5-bit words of the 1275-bit PUF response, 4 bits per word are disclosed as helper data. As a result, the entropy loss of using the repetition code is $\mathbf{H}_{\text{REP loss}} = 4 \cdot 255 = 1020$ bits. The entropy loss of the $C_{\text{BCH}}(255, 139, 15)$ BCH code is introduced by the random string that is needed to construct the code. As a result, the entropy loss of using this BCH code is $\mathbf{H}_{\text{BCH loss}} = n - k = 255 - 139 = 116$ bits. Hence, the total entropy loss of the 1275-bit PUF response by disclosing the helper data is $\mathbf{H}_{\text{loss}} = 1020 + 116 = 1136$ bits. This leaves $(1275 - 1136) \cdot \rho = 139 \cdot 0.9958 = 138$ bits of entropy left in the 255 bits of the BCH codeword.

These 255 bits will be compressed in a 128-bit key. This method is similar to the constructions in [10,14]. An advantage is that the AEAD-scheme can be used for this construction, minimizing the number of primitives that need to be implemented on the device.

5.3 AEAD-scheme

In our implementation of the protocol we use the AEAD-scheme KETJE [4] with security level $k = 128$, one of the 56 candidates of the CAESAR competition [3]. We use the AEAD-scheme KETJE for the key construction in the reverse FE, the encryption and decryption of the second PUF response Y^2 and the computation of the authenticator T^2. KETJE relies on nonce uniqueness to be secure, which we have taken into account when designing the implementation.

6 Results

In this section we present the results of the proposed protocol. First, we analyze the PUF responses. Second, we present the hardware and software performance of the proof of concept. Following, we give the benchmark results. Finally, we present the comparison of this work to other, similar authentication protocols.

6.1 PUF Response Analysis

Although we assumed that all the PUF response bits are independent, we found out this is not the case. To illustrate this, take two challenges that have a low Hamming distance. The probability that the responses of both challenges differ is only small because the majority of the travelled paths will match in both measurements. A possible reason why this is not reflected in [12] is that the authors challenge the PUF instances with random challenges. Moreover, the machine learning algorithm is trained with only $1,000$ training samples, which means that the probability of having two challenges with low Hamming distance is small. This characteristic means that the 12 bits that are used to retrieve the PUF responses, and the 12 bits that are used to retrieve the random variables, need to be diffused throughout the challenge space such that the highest probability of having different data paths is achieved.

As an experiment we implemented three PUF instances using this construction. These three PUFs were implemented on the same SoC at different locations, which gives us a good approximation of the PUF response quality on distinct SoCs. The metrics are calculated similarly as [12]. However, these results have been achieved by challenging three PUF instances with 40-bit challenges multiple times, obtaining multiple 1275-bit responses. More specifically, steadiness is calculated by challenging the PUF a number of $m = 1275$ times with a set of $n = 128$ equal challenges, averaging the Hamming distances between two arbitrary responses. Uniqueness is calculated by challenging two PUF instances a number of $m = 1275$ times with a set of $n = 500$ randomly chosen challenges, averaging the Hamming distances of each pair. Finally, randomness is calculated by challenging a PUF instance a number of $m = 1275$ times with a set of $n = 500$ randomly chosen challenges, averaging the Hamming weight of the responses. We find an average steadiness of 5.64%, an average uniqueness of 45.19% and an average randomness of 66.41%.

From this experiment, we see that for the DAPUF in our SoC the measure for steadiness is lower (6% versus 12%), which means that the responses in our implementation have a higher reproducibility. However, the randomness of our implementation is higher (66% versus 54%), meaning that the probability of a response bit being '1' is higher. In order to find out whether the output from our FE still provides enough entropy, we recalculate ρ. Using Formula 1 we find $\rho = 0.9208$. Thus, $139 \cdot 0.9248 = 128$ bits of entropy is left to accumulate the 255-bit BCH codeword, which is just enough to construct the 128-bit key. As a result, our implementation can be considered secure and thus privacy-preserving.

Next, we recalculate the fail rate using Formula 5. We find $p_{\text{fail}} = 8.438 \cdot 10^{-15}$, which is a considerable improvement.

6.2 Hardware Performance

The results have been generated by Vivado without the use of BRAM or DSPs and without optimization of the DAPUF design. Synthesis settings are set at `Default` and optimization settings at `Area`. Furthermore, we allow race conditions to occur due to the nature of the DAPUF.

Because of the long paths the signals have to travel through the DAPUF, the path delay is high. In the worst case scenario, the data path delay is 76.509 ns which means that the maximum frequency of the SoC is 12 MHz. The authentication phase of the device takes $8,205$ clock cycles, which on the frequency of 12 MHz takes 0.63 ms. As a result, our proof of concept might be applicable to devices in the IoT and in RFID systems.

In total, our proof of concept utilizes $8,305$ LUTs. The controller utilizes $5,464$ LUTs, Ketje $2,630$ LUTs, the DAPUF 195 LUTs and the BCH encoder 16 LUTs. Similar to the timing results, these utilization results are suboptimal. In this case the registers take a lot of area because of the long variables in the protocol.

6.3 Software Performance

The computation time of the server-side protocol increases linear in the number of devices in the database due to the exhaustive search. In our naive software implementation the execution time of the server-side protocol is approximately $0.05 \cdot n$ seconds. In a real world scenario, the server would be implemented in hardware which substantially decreases the execution time.

6.4 Benchmark Analysis

We analyze our protocol using the recently proposed benchmark for PUF-based authentication protocols [6]. Our device uses a PUF, TRNG, FE **Gen** procedure, cryptographic primitive (AEAD-scheme) and a one-time interface. Our PUF is a so-called strong PUF, indicating that the number of challenge-response pairs (CRPs) is at most 2^l, where l is the number of bits in the challenge. The amount of CRPs for n authentications is $n + 1$ because we use a one-time interface for the setup. The protocol supports server authenticity, device authenticity, device privacy, and memory disclosure. The protocol can support d-authentications for a perfect privacy use-case and ∞-authentications without device anonymity. Our PUF is noise-robust because of the error correction and modeling-robust because of the entropy accumulator in the FE. Mutual authentication provides both server and device authenticity. There is no internal synchronization which means that our implementation is not susceptible to DoS attacks. The execution time of the server per authentication is linear in the amount of devices.

6.5 Protocol Comparison

Table 1 summarizes the comparison between the proposed protocol and the protocols by Moriyama et al. [15] and Aysu et al. [2].

Table 1. Comparison with previous work.

Reference	Moriyama [15]	Aysu [2]	This work
Proofs for security and privacy	✓	✓	✓
Implemented parties	✗	device, server	device, server
Security flaws	✗	✓[2]	✗
Reconfiguration method	✗	modify SW, update microcode	follow generic approach, modify HW and SW
Demonstrator	✗	FPGA, PC	SoC, PC
Security-level	k	64-bit/128-bit	128-bit
Memory	PUF challenge & key	PUF challenge & key	PUF challenge
Device FE procedure	**Rec**	**Gen**	**Gen**
PUF type	✗	weak PUF	strong PUF
PUF instance	✗	SRAM	DAPUF
Hardware platform	✗	XC5VLX30	XC7Z020
Execution time (clock cycles)	✗	18,597	8,205
Logic cost (w/o PUF)	✗	1,221 LUTs	8,110 LUTs

The characteristic that all these protocols have in common is that they are all provably secure PUF-based privacy-preserving protocols. However, [15] only provides a theoretical basis for the proposed protocol, instead of also giving a proof of concept. As a result, no sensible answer can be given to the question whether the protocol is practical or not. On the other hand, [2] uses [15] as a basis, but is vulnerable to linear equation analysis of the FE output [2, p. 12]. As a consequence, this protocol does not provide a secure and privacy-preserving implementation. The performance results would highly likely be worsened because the FE needs to be redesigned. Correspondingly, most likely they need more PUF response bits to meet the failure rate requirements. Moreover, the implementation stores a key in non-volatile memory that does not increase the unpredictability of the communication messages. This overhead is eliminated in our protocol. Finally, with a different PUF (or a weak PUF) our results can be improved substantially, decreasing the execution time and logic cost.

[2] Due to a vulnerability in the implemented FE [2, p. 12].

7 Conclusion

In this paper we have proposed a new PUF-based privacy-preserving authentication protocol. In the process, we have presented proofs for security and privacy preservation, and an implementation serving as a proof of concept. We have seen that in comparison to other similar authentication protocols our protocol does not need a key stored in the non-volatile memory of the devices and is simpler in its design. Although our implementation is slower and consumes more resources in relation to [2], we claim to have an implementation that is both secure and privacy-preserving. On top of that, the performance results of [2] would highly likely be worsened because the FE needs to be redesigned because of the security flaw [2, p. 12].

Although we have presented a functional implementation, a faster and smaller proof of concept is possible. This is mainly due to the implemented PUF which defines the design of the FE and the variable sizes in the protocol. Moreover, the authentication time of the server is linear in the number of devices in the database, which could make the protocol impractical with a substantially large number of devices. A sound design of the server can settle this issue.

The design of our protocol might be optimized further, similar to what this research has achieved in relation to [2]. Mainly, future research has to be carried out towards strong PUF implementations, because these form the basis of our protocol. A strong PUF that has better quality of PUF responses can substantially reduce the consumption of the device. However, although our protocol is based on a strong PUF, a weak PUF can be used decreasing the maximum amount of authentications per device. Also, biometric data and a single PUF fingerprint can be used at the cost of device anonymity.

Acknowledgments I would like to thank Lejla Batina, Joan Daemen, Gergely Alpár and Antonio de la Piedra of the Digital Security Group at the Radboud University for their guidance and support which lead to the publication of this work.

References

1. Avnet Inc.: ZedBoard. http://zedboard.org/product/zedboard (2016), accessed: 2016-08-19
2. Aysu, A., Gulcan, E., Moriyama, D., Schaumont, P., Yung, M.: Cryptographic Hardware and Embedded Systems – CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings, chap. End-To-End Design of a PUF-Based Privacy Preserving Authentication Protocol, pp. 556–576. Springer Berlin Heidelberg, Berlin, Heidelberg (2015), http://dx.doi.org/10.1007/978-3-662-48324-4_28
3. Bernstein, D., et al.: CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness (2016), http://competitions.cr.yp.to/caesar.html
4. Bertoni, G., Daemen, J., Peeters, M., Van Asche, G., Van Keer, R.: CAESAR submission: Ketje v1 (March 2014), http://ketje.noekeon.org/Ketje-1.1.pdf

5. Bösch, C., Guajardo, J., Sadeghi, A.R., Shokrollahi, J., Tuyls, P.: Cryptographic Hardware and Embedded Systems – CHES 2008: 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings, chap. Efficient Helper Data Key Extractor on FPGAs, pp. 181–197. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-85053-3_12
6. Delvaux, J., Peeters, R., Gu, D., Verbauwhede, I.: A survey on lightweight entity authentication with strong pufs. ACM Comput. Surv. 48(2), 26:1–26:42 (Oct 2015), http://doi.acm.org/10.1145/2818186
7. Dodis, Y., Reyzin, L., Smith, A.: Advances in Cryptology - EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings, chap. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data, pp. 523–540. Springer Berlin Heidelberg, Berlin, Heidelberg (2004), http://dx.doi.org/10.1007/978-3-540-24676-3_31
8. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: Physical Unclonable Functions and Public-Key Crypto for FPGA IP Protection. In: 2007 International Conference on Field Programmable Logic and Applications. pp. 189–195 (Aug 2007)
9. Herrewege, A., Katzenbeisser, S., Maes, R., Peeters, R., Sadeghi, A.R., Verbauwhede, I., Wachsmann, C.: Financial Cryptography and Data Security: 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers, chap. Reverse Fuzzy Extractors: Enabling Lightweight Mutual Authentication for PUF-Enabled RFIDs, pp. 374–389. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-32946-3_27
10. Kelsey, J., Schneier, B., Ferguson, N.: Selected Areas in Cryptography: 6th Annual International Workshop, SAC'99 Kingston, Ontario, Canada, August 9–10, 1999 Proceedings, chap. Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator, pp. 13–33. Springer Berlin Heidelberg, Berlin, Heidelberg (2000), http://dx.doi.org/10.1007/3-540-46513-8_2
11. Lee, M.Z., Dunn, A.M., Waters, B., Witchel, E., Katz, J.: Anon-pass: Practical anonymous subscriptions. In: Security and Privacy (SP), 2013 IEEE Symposium on. pp. 319–333 (May 2013)
12. Machida, T., Yamamoto, D., Iwamoto, M., Sakiyama, K.: A New Arbiter PUF for Enhancing Unpredictability on FPGA. The Scientific World Journal (August 2015), http://dx.doi.org/10.1155/2015/864812
13. Maes, R.: Physically unclonable functions: Constructions, properties and applications. Ph.D. thesis, Ph. D. thesis, Dissertation, University of KU Leuven (2012)
14. Maes, R., Herrewege, A., Verbauwhede, I.: Cryptographic Hardware and Embedded Systems – CHES 2012: 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings, chap. PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator, pp. 302–319. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-33027-8_18
15. Moriyama, D., Matsuo, S., Yung, M.: PUF-Based RFID Authentication Secure and Private under Memory Leakage. Cryptology ePrint Archive, Report 2013/712 (2013), http://eprint.iacr.org/2013/712.pdf
16. Rogaway, P.: Authenticated-encryption with associated-data. In: Proceedings of the 9th ACM Conference on Computer and Communications Security. pp. 98–107. CCS '02, ACM, New York, NY, USA (2002), http://doi.acm.org/10.1145/586110.586125
17. Xilinx Inc.: Zynq-7000 All Programmable SoC Overview, Product Specification DS190 (v1.9). http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf (January 2016), accessed: 2016-08-19