Mutually Authenticated Key Exchange with Physical Unclonable Functions

Cyrus Minwalla*, Eirini Eleni Tsiropoulou†, Jim Plusquellic†
*Financial Technology Research, Bank of Canada, Ottawa, ON, Canada K1A 0G9
†Dept. of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87131-0001

***Abstract*—Authenticated key exchange is desired in scenarios where two participants must exchange sensitive information over an untrusted channel and do not trust each other at the outset of the exchange. As a unique hardware-based random oracle, PUFs can embed cryptographic hardness and binding properties desired for an interactive authentication system. Building on PUF technology, a light-weight end-to-end mutual authentication and session key generation protocol is proposed that simultaneously provides high levels of trust between authenticating parties and culminates in a shared session key that the parties can use to communicate over an untrusted network. A strong PUF called the Shift-register, Reconvergent, Fanout (SiRF) PUF is utilized and a Key Encryption Key (KEK) primitive implemented by this PUF is presented. The PUF's underlying entropy hardness characteristics and the KEK primitive are leveraged in the construction of a MAKE protocol (termed PUF-MAKE). Key properties of the protocol are a one-time enrollment for unlimited authentication instances, asynchronous connectivity between the issuing authority and the authentication server, and a key refresh mechanism that replaces both the lock (challenge) and the key (response) on every successful authentication. The protocol was experimentally verified using multiple realized FPGA PUF instances, with space and run time performance characterized across multiple configurations. A detailed attack analysis captures the robustness of the protocol. In particular theft and impersonation of credentials are eliminated as risks, which is a distinct improvement over traditional key authentication via asymmetric cryptography.**

***Index Terms*—Physical Unclonable Functions, Light-Weight Mutual Authentication and Key Exchange Protocol.**

I. Introduction

Consumer devices are increasingly represented as embedded systems connected wirelessly to Internet of Things (IoT) infrastructure [1]. The parties must first be mutually authenticated before cooperating to generate a shared secret to encrypt subsequent communications over the network. Mutual authentication protocols that embed a binding property where the shared secret is tied to the authentication process ensures that fraud and malfeasance during the exchange can be traced back to the malicious participant, a crucial component for high-value, private and time sensitive applications such as digital money and identity. Strong physical unclonable functions (PUF) rely on an exponential challenge-response search space while embedding a local source of entropy. Moreover, secrets of a specific instantiation are inextricably tied to the silicon, ergo, credentials cannot be stolen and impersonating a device requires modeling the device itself. Such PUFs can play an important role as a hardware root of trust that is impervious to a variety of typical attack vectors, including supply-side threats, to mitigate large-scale system-wide risks in mission-critical applications.

The twin properties of a hardware random oracle coupled with true random number generation can be leveraged to craft an authentication scheme that captures the property of a one-time pad in an extended-use authentication scheme that is centered around the PUF itself as a local root of trust and the enrollment as a central root of trust, wherein both must be compromised simultaneously to defeat the authentication mechanism.

A. Contributions

In this work, we propose a PUF-based mutual authentication and key exchange (PUF-MAKE) protocol that utilizes a set of hardware-based security primitives derived from a strong physical unclonable function (PUF). The proposed protocol establishes explicit authentication for both parties facilitated by a trusted authority, and constructs a shared session key for the partnered parties in a single round. The system can replace a traditional certificate authority scheme such that fixed long-lived credentials are replaced by a device with mutable credentials.

In addition, the authentication challenge and response are updated on every authentication cycle, to achieve greater levels of security over a traditional certificate-based credential system. Finally, the list of challenges per device can be refreshed *asynchronously* at the server without requiring re-enrollment or the physical availability of the PUF itself. The protocol is light-weight and uses mature cryptography and deterministic O(n) operations for all phases of the protocol. The scheme is well-tailored for mobile and embedded devices, but it can also be used in any type of client-server architecture to implement a hardware-based root-of-trust and secure enclave for communications. Security guarantees are preserved even if one or both participants are compromised, provided the root of trust (IA) remains intact.

II. Relevant Research

Authenticated key exchange (AKE) is a mechanism that serves the dual purpose of (a) authenticating two parties with respect to each other, and (b) building a secure communications channel between two parties over untrusted networks (e.g., the Internet) where active adversaries can be expected. In cases where (a) is satisfied for both parties, the protocol is considered to be mutually authenticated key exchange

(MAKE). Most AKE and MAKE schemes in literature are based on asymmetric cryptography. Early work by Diffie et al. [2] explore authenticated key exchanges as part of a station-to-station (STS) encryption protocol. The work is notable for establishing design principles for a secure protocol, with later work by Bellare and Rogaway [3] further formalizing this approach, with the authors specifically noting that two partners are not restricted to just sharing a session key, extending the approach to arbitrary two-party protocols.

Bellare et al. [4], [5] later proposed an authenticated key exchange protocol based on the standard model of an asymmetric encryption algorithm coupled with a trap-door function, but the scheme proved susceptible to chosen cipher-text attacks. Okamoto [6] improved upon this approach by introducing a pseudo-random function as a replacement for the trap-door function, proving IND-CCA indistinguishability under the standard model. In parallel, Law et al. [7] proposed an efficient protocol titled MQV (the author's initials concatenated) combining Diffie Hellman for forward secrecy and elliptic curve cryptography for efficient key generation. Authentication, however, was implicit as entities were not identified, leaving the scheme vulnerable to impersonation attacks. Security of the protocol was later upgraded by Krawczyk [8] by introducing a Schnorr signature scheme which serves the dual purpose of including identities and a building block to construct a challenge-response sequence for a three-round protocol. LaMacchia et al. [9] improved upon Krawczyk's scheme with the NAXOS protocol by making stronger adversarial assumptions. MAKE is also standardized in TLS 1.3 as per RFC 8446 circa 2018. This standard implements MAKE by relying on certificates from a public key infrastructure for authentication and the Diffie-Hellman protocol for session key generation.

More recent efforts around MAKE constructions utilize PUFs as a local root of trust as a replacement for asymmetric cryptography. An extensive literature review on previously proposed light-weight PUF-based authentication protocols is given in [10]. The authors of [11] propose a lightweight PUF-based mutual authentication and secret message exchange protocol. The protocol only succeeds in authenticating the server, and further, assumes that the router has a soft model of the PUF and can generate a response to any randomly generated challenge during the refresh phase. Mahalat et al. [12] propose a scheme for secure WiFi authentication of IoT devices. This approach was later expanded to a PUF-based authentication and key sharing scheme that utilizes Pedersen's commitment scheme coupled with Shamir's secret sharing [13]. The mutual authentication and key sharing scheme proposed between user (server) and sink nodes can be implemented more easily using challenge-response-pair (CRP) strong PUF-based schemes without the mathematical complexity of the secret sharing schemes [14]. This becomes possible since in both cases the server stores a CRP database that is constructed in a secure environment during provisioning.

A PUF-based authentication and key management protocol for IoT is proposed in [15], improving upon on the attack resilience and performance overhead of the previous method [16]. Elliptic curve cryptography (ECC) is used to create shared keys among IoT nodes with the assistance of a verifier. The scheme requires a trusted setup and tamper-resistant hardware to protect secret keys. Similarly, a controlled PUF that utilizes ECC is proposed as a lightweight authentication and key generation protocol for IoT nodes in [17], relying on zero knowledge proofs for device authentication, however the authentication is one-way, and the server is not authenticated.

A PUF-based El-Gamal algorithm is proposed for message encryption as well as a PUF-based digital signature scheme. In parallel, Yu et al. [18] propose a scheme that is designed to prevent an adversary from obtaining sufficient CRPs to carry out model-building attacks. However, the number of authentications is constrained by the number of CRPs stored in the database, requiring either reuse of entries or re-enrollment of the PUF, a caveat we avoid in our scheme. In [19], the authors propose a crossover ring oscillator (R)O PUF cloning technique that enables a group of IoT devices to all generate the same (shared) key, thereby eliminating the key distribution problem for devices engaging in multi-party shared key encrypted communication.

III. Hardware Security Primitives

The proposed protocol utilizes a novel strong physical unclonable function (PUF) called the Shift-register, Reconvergent-Fanout (SiRF) PUF and a PUF-based protocol primitive called *key-encryption-key* (KEK) [20]. KEK was first described in [21] and later expanded on in [22], in a protocol for PUF-based authentication in resource constrained environments (PARCE). In this paper, we implement and evaluate the SiRF PUF and KEK security primitive in FPGAs experiments and extend their usage in the protocol to mutual, privacy-preserving authentication and ephemeral session key generation.

A. Overview of SiRF PUF

The SiRF PUF [20] is a delay-based, strong PUF that digitizes delay values (DV) into response bitstrings and keys using the SiRF algorithm and a specialized key-encryption-key (KEK) security function. SiRF is capable of providing an exponential number of reproducible, long-lived keys (LLK). SiRF is implemented entirely within the programmable logic of a SoC-based FPGA as a secure enclave, thus, it is hardened against software-based attacks carried out on the embedded SoC microprocessor. The LLK produced by the SiRF PUF (before hashing) are evaluated against the standard statistical quality metrics. The protocol uses SiRF as a strong PUF, and is compatible with any strong PUF possessing similar hardware qualities.

The architecture utilizes a complex network of shift registers, logic gates and MUXs as the source of entropy (left side of Fig. 1). Challenge bits configure the network to create signal paths that pass through the layered structure. The logic gate network is distributed across a 20x20 CLB region of the FPGA as a means of eliminating localized bias that exists within FPGA LUTs and switches.

The application of a challenge involves driving a set of rising or falling input transitions into the logic gate network using *Launch FFs* shown along the top left in Fig. 1. Signals propagate through the network and emerge as output transitions on the MUX inputs. The MUX selects and routes one of the output signal paths to a time-to-digital (TDC) converter (center of Fig. 1). The TDC is used to measure and digitize the delay of the signal path at high resolution, approximately 20 ps. The digitized path delays are stored in a Block RAM (BRAM) and post-processed by set of modules, defined collectively as the SiRF algorithm, to produce random, unique and reliable encryption keys or authentication bitstrings. Important characteristics of the DVDiff, GPEV and Spread Factors modules are captured as follows, with additional details provided in a technical report [20].

B. Key-Encryption-Key (KEK) SiRF PUF algorithm

Key-encryption-key or KEK is traditionally used in reference to a master key that a device stores and uses to decrypt boot images at start-up, and to generate other keys, e.g., ephemeral session keys and authentication bitstrings during system operation. Given its central role, it defines the root-of-trust in most systems. KEKs also need to be reproducible at any instance in time, potentially over the lifetime of the device, and are therefore also referred to as long-lived keys (LLK). The role of KEK as the root-of-trust also imposes strong security constraints on the system to maintain its secrecy because an attack that is able to extract the KEK compromises the entire system.

A strong PUF is hardened against KEK extraction attacks as it stores only the challenges and helper data needed to reproduce the KEK, not the KEK itself, and can go a step further by harnessing the ability to generate an exponential number of LLKs. The SiRF PUF leverages these benefits to enable Alice and Bob to authenticate and generate a shared session key. In particular, the SiRF long-lived key generation function provides an exponential number of unique KEKs as a means of meeting the one-time use constraint associated with authentication, while simultaneously providing the ability to reliably regenerate any one of its KEKs on-demand.

C. Thresholding and XMR

The SiRF PUF leverages variations in path delays as a source of entropy. As indicated earlier, high resolution measurements of path delays are obtained using a time-to-digital converer or TDC. The digitized representations of path delays produced by the TDC are referred to as *delay values* or DV. DV capture small delay variations that occur uniquely within each device, and therefore, represent the source of entropy for the PUF. The SiRF algorithm shown along the right side of Fig. 1 processes DV into DVD_{cr}, which represent the input to the *BitGen* component of the algorithm. The DVD_{cr} are compensated and randomized delay difference values which significantly improve on the reliability, uniqueness and randomness characteristics of the original DV [20]. Two components of the SiRF algorithm, called thresholding and XMR, are tasked with converting the DVD_{cr} into bitstrings and keys for use in the MAKE protocol.

Fig. 2 illustrates the thresholding and XMR processes carried out by the KEK algorithm. We use the term *first-strong-bit* (FSB) to refer to this operational mode of the KEK algorithm (in contrast to the *secure-key-encoding* (SKE) mode described in [22]). A set of DVD_{cr} measured and processed by an instance of the SiRF PUF are plotted along the x-axis. The output during enrollment is a XMR helper data bitstring and response bitstring. The algorithm used to generate the response bitstring is described in the following.

A thresholding scheme is used to increase the reliability of the SiRF PUF's bitstring generation process. The threshold is tunable with larger values providing greater certainty that the KEK will be reproduced without bit-flip errors during regeneration. From Fig. 2, the DVD_{cr} are partitioned into two regions by the horizontal line at 0, with values above the line assigned a bit value of 1 and values below the line assigned 0. The threshold further partitions each group of DVD_{cr} into strong and weak classes. The two horizontal dotted lines in Fig. 2 represent the threshold, which are placed at equal-distance positive and negative displacements around 0.

The DVD_{cr} within the threshold region are classified as weak because they are closer to 0 and are less reliable, i.e., they have a higher probability of flipping between 0 and 1 and vise versa during regeneration. This is true because of the uncompensated noise and drift that occurs in the measurement and processing of the DVD_{cr}, especially when the device is operated in hot or cold temperature environments and/or when the supply voltage is above or below nominal. In other words, the DVD_{cr} vary (within a limit) during regenerations when the paths are remeasured, making it possible for DVD_{cr} close to 0 to *flip* to the opposite bit value when compared against the bit value generated during enrollment. Weak DVD_{cr} are excluded from the bitstring generation process by labeling them with a 0 in the helper data bitstring. Strong DVD_{cr}, on the other hand, are located above the upper threshold or below the lower threshold, and are labeled with a 1 in the helper data bitstring. Thresholding improves the reliability of the bitstring regeneration process by creating a buffer against environmental disturbances, such as power supply noise or temperature changes.

The XMR algorithm adds a second layer of resiliency to the KEK regeneration process. The annotations in Fig. 2 show the response bitstring generated using TMR (triple modular redundancy or 3MR), although any odd number is suitable for the XMR redundancy scheme with higher levels, e.g. 5MR, providing higher levels of protection against bit flip errors. TMR uses 3 consecutive strong bits to encode one **super-strong** KEK bit. The XMR algorithm scans the DVD_{cr} from left to right searching for the first strong bit indicated by the helper data bitstring. For example, from the figure, the leftmost DVD_{cr} produces a strong 1. A 1 is added to both the response bitstring and the TMR_1 block of bits (which are shown for illustration purposes only). The algorithm continues to scan the DVD_{cr} searching for two more instances of strong 1's, skipping strong 0's and DVD_{cr} previously labeled as weak. Bits in the XMR helper data bitstring corresponding to strong 0's are changed from 1 to 0 during this scan, i.e., they are also labeled as weak. Once three strong 1's are located, a super-

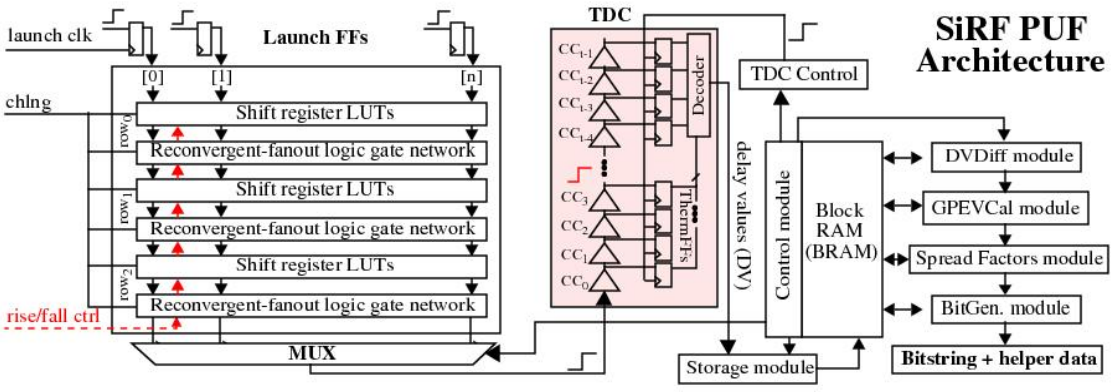


Fig. 1: SiRF PUF architecture illustrating shift register and layered logic gate network which serve as the source of entropy.

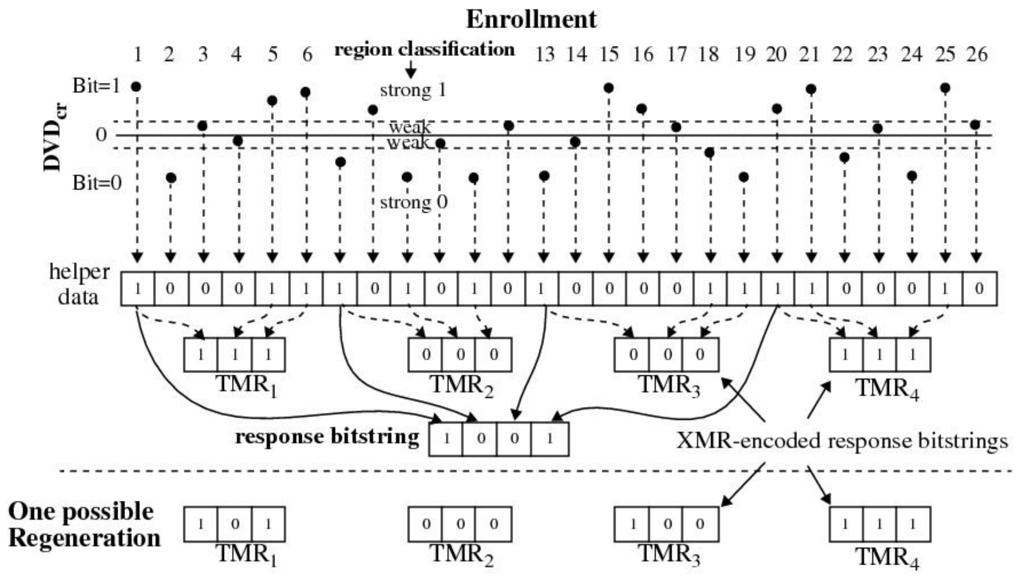


Fig. 2: XMR illustration showing response bitstring generation using FSB mode during enrollment (top) and regeneration (bottom).

strong KEK bit is considered fully encoded and the algorithm searches for the next strong bit of either value to represent the second KEK bit. This occurs at position 7 where a strong 0 is found and the process is repeated.

Regeneration reverses the process associated with the XMR helper data bitstring, in particular, the helper data bitstring generated during enrollment is treated as read-only during regeneration. The 1's in the helper data bitstring indicate which DVD_{cr} to use in the reconstruction of the TMR_x sequences. A KEK bit is generated from each TMR_x sequence using majority vote among the three TMR_x bits. Therefore, the TMR_x sequences are able to correct any single bit-flip error, adding resiliency to the KEK regeneration process. The regeneration process is shown along the bottom of Fig. 2.

IV. The PUF-MAKE Protocol

The proposed protocol is implemented using a codesign methodology, with both software (C code) and hardware (Verilog) components. The SiRF PUF and KEK algorithm are implemented entirely in Verilog, and synthesized to a Xilinx Zynq system-on-chip (SoC) FPGA. The SiRF PUF uses high-speed general-purpose input-output (GPIO) registers to exchange challenges and KEK bitstrings with a C-version of the protocol running on an ARM microprocessor, integrated onto the Xilinx SoC. The linux operating system running on the ARM microprocessor provides a TCP-IP network stack to enable commmunications between protocol entities.

The protocol requires a minimum of four entities, namely an issuing authority (IA), an authentication server(s) (AS), and a pair of devices (Alice and Bob) who wish to establish a secure channel with each other to exchange information. An example application is one of digital currency, where the IA is the central authority, AS is an intermediary and Alice and Bob as devices that exchange electronic funds for goods and services. For example, Alice can be a customer wishing to buy something wirelessly from a brick-and-mortar or on-line store (Bob), or Alice and Bob may be two customer devices communicating through a Bluetooth or IR link to exchange funds. The AS in this scenario can be a trusted intermediary providing value-added services, ranging from a small embedded system in a coffee shop to a bespoke enterprise server installation tailored to high-traffic applications. Notably the

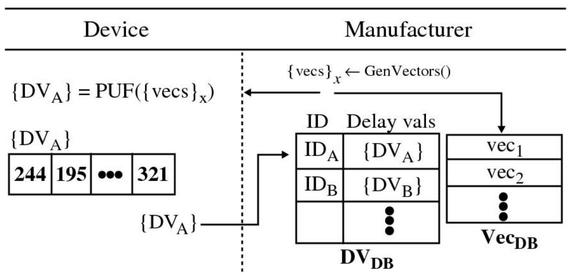


Fig. 3: The provisioning process showing the manufacturer of the customer devices supplying vectors $vecs_x$ to an instance A of the SiRF PUF on a newly manufactured device and recording delay values DV_A into the DV_{DB}.

scale and scope of deployment is not restricted by protocol primitives. The root of trust resides in the IA and is extended to the field via the AS and ultimately to fielded devices.

The implementation presented here consists of the minimum set of entities, namely Alice and Bob's devices, AS and IA. In our implementation, the PUF devices and AS run on Xilinx FPGAs and authenticate and encrypt communications to each other. A small embedded-system version of the AS is cost-effective in scenarios where it may be preferred to distribute multiple independent copies of AS; configuration options are discussed in a later section.

The IA stores meta-data collected during PUF provisioning in two separate databases, labeled DV_{DB} and Vec_{DB} in Figs. 3 and 4. The IA is responsible for carrying out the enrollment process and for providing a fresh supply of Authentication Tokens (AT) in a *refresh* process with AS during fielded operations. IA is typically a multi-threaded application that runs on a multi-core server in a central, secure facility.

The AS can also run as a multi-threaded application (C program), and is initialized and periodically refreshed with a compact representation of the AT, a core component of PUF-MAKE's CRP-based authentication and session key generation process. The left side of Fig. 4 shows the format of the database called $AuTk_{DB}$ that stores AT. An AT is defined as including a customer device ID, e.g., ID_A, a hashed version of a KEK response, HK_A, a SiRF PUF challenge $Chlng_A$ and helper data HD_A. This information enables AS to validate Alice and Bob's devices as enrolled, PUF-instantiated devices via an interactive protocol.

The protocol encapsulates three core properties: First, physical access to the PUF is required to participate in authentication, second challenge-response pairs, in the form of $Chlng_{Ax}$-HK_{Ax}, stored in the $AtTk_{DB}$ are used only once in a single authentication, and third, that each element AT in AS's $AtTk_{DB}$ relies on a successful authentication of the previous pair, with the first pair originating from the IA. The last property employs a sequencing methodology that extends the root-of-trust from IA to AS and then to Alice and Bob.

A. MAKE Provisioning

The provisioning process for MAKE is shown in Fig. 3, where the manufacturer of the device (or IA) is a trusted authority that collects timing data DV and vectors $\{vec\}$ for storage in the DV_{DB} and Vec_{DB} databases. In contrast to other proposed PUF-based authentication schemes, which store response bitstrings instead, the DV enable IA to expand the challenge-response space exponentially. For example, the SiRF algorithm operations shown on the right side of Fig. 1 expand the challenge-KEK CRP space defined by a stored set of n DV to at least n^2. A further combinatorial expansion is realized by selecting x DV from the larger set of n DV during authentication. A more thorough analysis of the SiRF CRP space is provided in the Security Analysis section of this paper. In our experiments, we provision each device and store approximately 8,000 DV in the DV_{DB} database (approx. 3.5 MByte/device) and approximately 1500 challenge vectors in the Vec_{DB} (approx. 1 MByte in size).

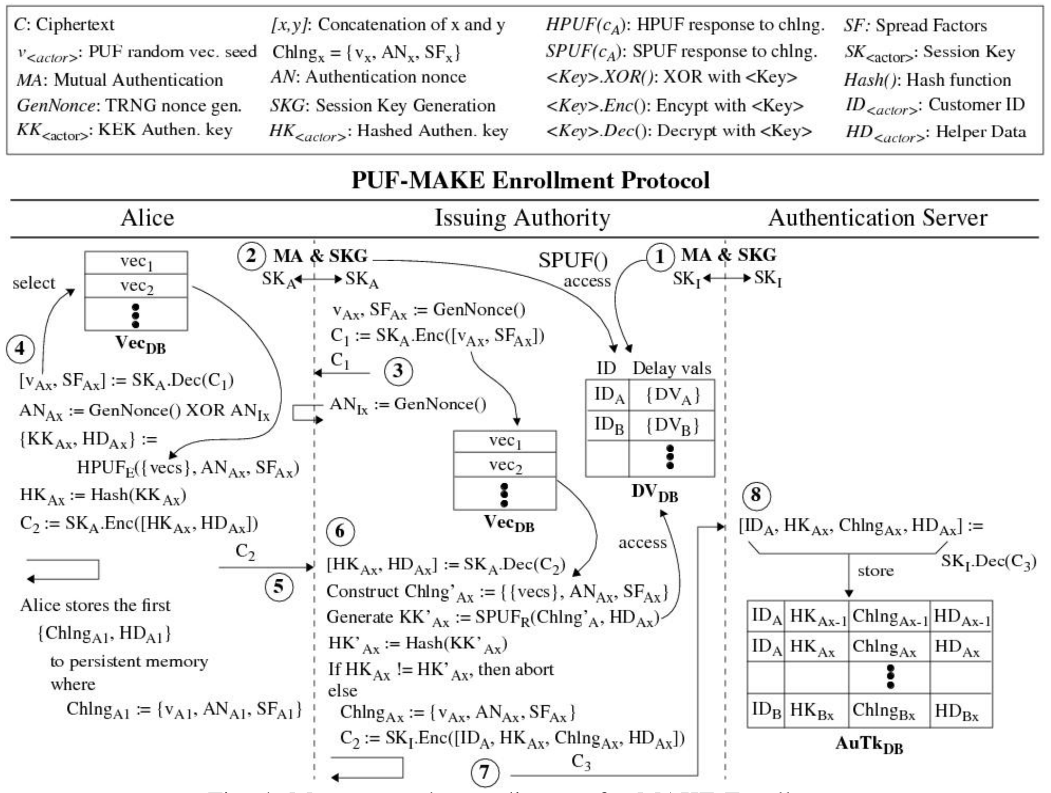
B. MAKE Enrollment

The enrollment process is performed after a new customer device is provisioned, and involves three entities including the device, IA and AS. A close variant is performed periodically between only the IA and AS after the customer device is deployed but only after the number of AT stored by the AS drops below a threshold. We describe the differences between this *refresh* operation and the initial three-party version in the following.

The message exchange diagram for the three-party version is shown in Fig. 4. IA stores provisioning DV data for the device in its DV_{DB} database and interacts directly with the device to create a set of AT. The requirement for the device to be involved in this three-party version prevents certain attacks that are discussed in the Section VI.

The ordered sequence of message exchanges and operations that comprise the enrollment process are described below, in correspondence with the number annotations shown in Fig. 4.

1) The Issuing Authority (IA) and Authentication Server (AS) mutually authenticate (MA) using a privacy-preserving PUF-based protocol and then generate a session key (SKG), called SK_I. The process utilizes the KEK algorithm and is similar to the PARCE authentication protocol described in [22], and therefore, the details are omitted for brevity.
2) The same MA and SKG process is carried out between Alice and the IA to authenticate and generate a shared session key SK_A.
3) The IA runs a TRNG to generate a random seed v_{Ax} which is used to seed a pseudo-random number generator (PRNG). The output of the PRNG is used to select a set of challenge vectors $\{vecs\}$ from the Vec_{DB}. The IA runs a TRNG again to generate a random set of Spread Factors, SF_{Ax}, that are used by the SiRF PUF Spread Factors module (see right side of Fig. 1) as a means of randomizing the KEK response, thereby expanding the CRP space. The v_{Ax} and SF_{Ax} are encrypted using the AES-256 and the ciphertext C_1 is transmitted to Alice.
4) Alice decrypts C_1 to obtain v_{Ax} and SF_{Ax}. She uses v_{Ax} as a seed to a PRNG to select the same set of challenge vectors $\{vecs\}$ (as IA) from her copy of Vec_{DB}. IA sends Alice a TRNG-generated Authentication Nonce, AN_{Ix}, that she XORs with her own SiRF



C: Ciphertext	*[x,y]*: Concatenation of x and y	*HPUF(c_A)*: HPUF response to chlng.	*SF:* Spread Factors
$v_{<actor>}$: PUF random vec. seed	$Chlng_x = \{v_x, AN_x, SF_x\}$	*SPUF(c_A)*: SPUF response to chlng.	$SK_{<actor>}$: Session Key
MA: Mutual Authentication	*AN*: Authentication nonce	*<Key>.XOR()*: XOR with <Key>	*Hash()*: Hash function
GenNonce: TRNG nonce gen.	*SKG*: Session Key Generation	*<Key>.Enc()*: Encypt with <Key>	$ID_{<actor>}$: Customer ID
$KK_{<actor>}$: KEK Authen. key	$HK_{<actor>}$: Hashed Authen. key	*<Key>.Dec()*: Decrypt with <Key>	$HD_{<actor>}$: Helper Data

PUF-MAKE Enrollment Protocol
Alice
Issuing Authority
Authentication Server
select
vec1
vec2
VecDB
2 MA & SKG
SKA ↔ SKA
SPUF() access
1 MA & SKG
SKI ↔ SKI
vAx, SFAx := GenNonce()
C1 := SKA.Enc([vAx, SFAx])
C1
3
4
[vAx, SFAx] := SKA.Dec(C1)
ANAx := GenNonce() XOR ANIx
ANIx := GenNonce()
{KKAx, HDAx} :=
HPUFE({vecs}, ANAx, SFAx)
HKAx := Hash(KKAx)
C2 := SKA.Enc([HKAx, HDAx])
C2
5
ID Delay vals
IDA {DVA}
IDB {DVB}
DVDB
access
8
[IDA, HKAx, ChlngAx, HDAx] := SKI.Dec(C3)
store
6
[HKAx, HDAx] := SKA.Dec(C2)
Construct Chlng'Ax := {{vecs}, ANAx, SFAx}
Generate KK'Ax := SPUFR(Chlng'A, HDAx)
HK'Ax := Hash(KK'Ax)
If HKAx != HK'Ax, then abort
else
ChlngAx := {vAx, ANAx, SFAx}
C2 := SKI.Enc([IDA, HKAx, ChlngAx, HDAx])
C3
7
Alice stores the first
{ChlngA1, HDA1}
to persistent memory
where
ChlngA1 := {vA1, ANA1, SFA1}
IDA HKAx-1 ChlngAx-1 HDAx-1
IDA HKAx ChlngAx HDAx
IDB HKBx ChlngBx HDBx
AuTkDB

Fig. 4: Message exchange diagram for MAKE Enrollment.

PUF TRNG-generated nonce to create AN_{Ax}, that she then transmits back to IA. AN_{Ax} also expands the CRP space of the SiRF algorithm and is used later in an in-field AT validation step. Alice applies the challenge tuple $[\{vecs\}$, AN_{Ax}, $SF_{Ax}]$ to her hardware HPUF in enrollment mode to generate an authentication key, KK_{Ax} and helper data HD_{Ax}. She hashes KK_{Ax} using a SHA-3 hashing function to produce HK_{Ax}, and encrypts HK_{Ax} and HD_{Ax} with SK_A to produce ciphertext C_2.

5) Alice transmits ciphertext C_2 to IA. On the first iteration of the MAKE Enrollment process, Alice also stores the tuple $[Chlng_{A1}$, $HD_{A1}]$ to persistent memory. She will use this challenge and helper data to reproduce HK_{Ax} during her first in-field authentication, discussed below.
6) IA decrypts C_2, constructs $Chlng'_{Ax}$, matching Alice's challenge, and applies it to its soft PUF algorithm, $SPUF_R$ along with the helper data HD_{Ax} supplied by Alice. The $SPUF_R$ algorithm reads a set of DV corresponding to the Challenge vectors $\{vecs\}$ from the DV_{DB} and runs the KEK algorithm in regeneration mode to produce KK'_{Ax}. IS hashes KK'_{Ax} and compares the output HK'_{Ax} with the decrypted HK_{Ax} received from Alice. If they match then Alice's HK_{Ax} is validated. IA then encrypts packet C_3 with SK_I, containing Alice's ID, ID_A, and Authentication Token components, namely HK_{Ax}, $Chlng_{Ax}$ and HD_{Ax}.
7) IA transmits the C_3 to AS.
8) AS decrypts the C_3 and stores ID_A, HK_{Ax}, $Chlng_{Ax}$ and HD_{Ax} in its $AuTk_{DB}$ database. The AT elements in this table will be used by AS for in-field authentication operations carried out on behalf of Alice (and Bob) and to enable Alice and Bob to generate a shared session key.

A two-party version of the enrollment is used to refresh the $AuTk_{DB}$ database maintained by AS. The absence of Alice in the two-party version requires IA to act as a surrogate for Alice. Since IA stores a subset of Alice's DV, it too can generate KK_{Ax} and helper data HD_{Ax} by running $SPUF_E$, the enrollment version of the KEK algorithm. This change constitutes the primary difference between the two versions. The AN_{Ax} nonce exchange in Step 3 is also omitted, with IA generating the AN_{Ax} nonce by itself, as is the validation operation in Step 6. AS receives new AT for its database in Steps 7 and 8, as shown by the three-party version of the protocol.

C. MAKE In-Field Interactive Authentication Protocol

At the end of enrollment, Alice and Bob each store a challenge that they will use to generate a response, HK_{A1}, for the first authentication request with AS. The in-field version of the protocol is sequenced such that Alice and Bob must first authenticate to AS with their existing challenge, and as part of a successful authentication, receive and store a new challenge for the next authentication cycle, guaranteeing forward secrecy. The $AuTk_{DB}$ maintained by AS is designed to make the protocol light-weight and fast but secure against many types of attacks. Each authentication uses a one-time credential that is replaced at the end of the authentication cycle. The session key is jointly derived by both participants, creating a binding partnership between devices for the duration of the session.

The Interactive Authentication message exchange diagram for MAKE is shown in Fig. 7. The diagram shows the sequence of operations that occur when Alice and Bob wish to establish an authenticated and encrypted channel with each other, e.g., to exchange electronic cash for goods and services.

1) The protocol begins with Alice and Bob sending a request to communicate to each other. Bob responds with an acknowledgement of Alice's request. Both the request and the response embed device IDs (ID_A, ID_B) and nonces (n_T, n'_T). These nonces will later be used to XOR-decrypt a shared session key between Alice and Bob while ensuring the session key remains hidden from AS. In order to accomplish this goal, AS is excluded as a participant in this nonce exchange.
2) Using a stored PRNG seed v_{A1}, Alice extracts the set of vectors $\{vecs\}$ associated with this seed from her Vec_{DB}. The $\{vecs\}$ along with the stored challenge nonces n_{A1} and SF_{A1} are used as the input challenge to SiRF PUF. She runs her hardware instance of the SiRF PUF, $HPUF_R$ in regeneration mode to regenerate the response KK_{A1}. She then calculates a hash HK_{A1} of KK_{A1} using a suitable hash function, for convenience we assume SHA-3.
3) *Shared Session Key Generation*: Alice and Bob generate session key shards, SK_T and SK'_T, using their PUF-based TRNGs. Alice and Bob then encrypt their session key shards with the nonces they generated in Step 1, n_T and n'_T to form C_T and C'_T, respectively.
4) Once Alice constructs C_T, the packet C_1 is assembled by concatenating authentication artifacts (AN_{A1}), Bob's device ID, ID_B and the session shard C_T, which are then AES-encrypted with HK_{A1} as the key. Alice sends the authentication request to AS, along with ID_A and C_1 as metadata. The server matches ID_A to the list of known devices in the database $AuTk_{DB}$ populated by the Issuing Authority (IA) during enrollment. Upon match, the server uses the associated HK_{A1} to decrypt C_1 and disassembles the AN_{A1}, ID_B and C_T fields. Authentication is a success if the extracted AN_{A1} matches the one stored for ID_A in the database, otherwise this process repeats using each of the remaining ID_A elements. If no matches are found, the protocol aborts. Note that both HK_{A1} and AN_{A1} must be correct for authentication to succeed. If HK_{A1} does not match any of those stored in the $AuTk$ database, the packet will not be decrypted correctly and AN_{A1} will be random, causing a mismatch. Alternatively, if HK_{A1} is correct and AN_{A1} is not, the packet will decrypt correctly but AN_{A1} will fail to match. In either case, authentication fails and the server aborts the connection. Bob performs this same set of operations with AS.
5) If Alice's authentication succeeds, AS adds ID_A and C_T to a $SESSION_{DB}$ database. Similarly, if Bob's authentication succeeds, AS adds ID_B and C'_T to the $SESSION_{DB}$. In both cases, AS first searches for a match to ID_B and ID_A supplied by Alice and Bob, respectively, to determine if a row already exists, and, if so, adds Alice or Bob's information to matching row element instead of creating a new row. Once Alice and Bob's ID and C_T are both present, AS proceeds to the next step, otherwise it stalls the thread waiting for Alice or Bob to complete the transaction.
6) AS can now complete the exchange with Alice and Bob to enable them to derive a shared session key S_K in the final step of the protocol. However, AS must ensure that Alice and Bob use a new challenge in their next authentication request. AS first deletes the current AT from its $AuTk_{DB}$ database and then selects two new AT elements. The key update mechanism ensures that Alice and Bob are protected against both impersonation and replay attacks.
7) AS constructs a packet C_A for Alice by encrypting Bob's authentication artifacts ID_B, C'_T and Alice's next challenge $Chlng_{A2}$ and HD_{A2} with Alice's HK_{A1} AES key. AS constructs a packet C_B for Bob in a similar fashion and transmit the packets to Alice and Bob along with an ACK. This completes Alice and Bob's interactive authentication with AS. Note that the Alice's C_T and Bob's C'_T embed their session key shards SK_T and SK'_T, respectively. AS is, however, prevented from learning the shards because they are XOR-encrypted with Alice and Bob's n_T and n'_T.
8) Alice decrypts her C_A with HK_{A1} to obtain Bob's ID ID_B, Bob's C'_T and her next authentication challenge. She replaces the current challenge $Chlng_{A1}$ and helper data HD_{A1} with $Chlng_{A2}$ and HD_{A2}. She extracts Bob's shared session key shard SK'_T by XOR-decrypting C'_T with his n'_T nonce. Bob carries out the same operations. They both XOR the SK_T and SK'_T shards to obtain a shared session key SK, which they can use to encrypt communications between them.

V. Experimental Results

The MAKE Enrollment and Interactive Authentication (In-Field) protocols were implemented and tested on a set of Digilent ZYBO boards [23] and a Dell PowerEdge T440 Server with 32 1.8 GHz processors and 128 GB of main memory. The implementation testbed possesses the following characteristics:

- The programmable logic (PL) component of the Xilinx Zynq 7010 FPGA on the ZYBO Z7 board is programmed with an instance of the SiRF PUF and KEK key generation algorithms. A C program running under the Linux operating system implements the network communication protocol and communicates with the SiRF PUF and KEK hardware instantiations through GPIO registers. A micro-SD card provides persistent storage for sqlite3 database tables, which implement the Vec_{DB} and store Alice/Bob's authentication challenge and helper data.
- The Authentication Server (AS) is also implemented on a ZYBO Z7 board, with PL component programmed with an instance of the SiRF PUF. The C program implementation of AS is a multi-threaded application enabling concurrent communication through sockets with

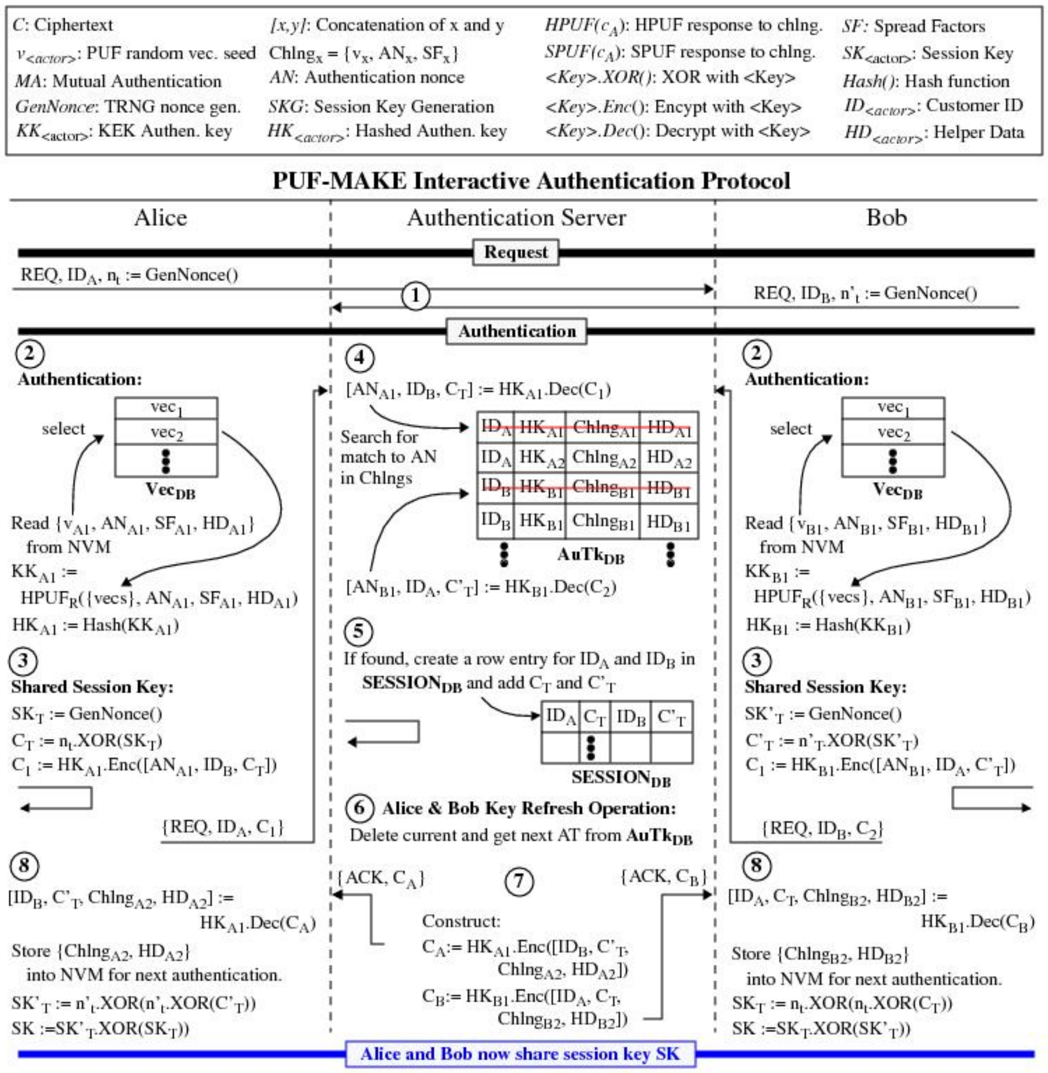

C: Ciphertext
$v_{<actor>}$: PUF random vec. seed
MA: Mutual Authentication
GenNonce: TRNG nonce gen.
$KK_{<actor>}$: KEK Authen. key
[x,y]: Concatenation of x and y
$Chlng_x = \{v_x, AN_x, SF_x\}$
AN: Authentication nonce
SKG: Session Key Generation
$HK_{<actor>}$: Hashed Authen. key
$HPUF(c_A)$: HPUF response to chlng.
$SPUF(c_A)$: SPUF response to chlng.
<Key>.XOR(): XOR with <Key>
<Key>.Enc(): Encypt with <Key>
<Key>.Dec(): Decrypt with <Key>
SF: Spread Factors
$SK_{<actor>}$: Session Key
Hash(): Hash function
$ID_{<actor>}$: Customer ID
$HD_{<actor>}$: Helper Data
PUF-MAKE Interactive Authentication Protocol
Alice
Authentication Server
Bob
Request
REQ, ID_A, n_t := GenNonce()
1
REQ, ID_B, n'_t := GenNonce()
Authentication
2
Authentication:
select
vec_1
vec_2
Vec_{DB}
Read $\{v_{A1}, AN_{A1}, SF_{A1}, HD_{A1}\}$ from NVM
KK_{A1} := $HPUF_R(\{vecs\}, AN_{A1}, SF_{A1}, HD_{A1})$
HK_{A1} := Hash(KK_{A1})
3
Shared Session Key:
SK_T := GenNonce()
$C_T := n_t.XOR(SK_T)$
$C_1 := HK_{A1}.Enc([AN_{A1}, ID_B, C_T])$
$\{REQ, ID_A, C_1\}$
4
$[AN_{A1}, ID_B, C_T] := HK_{A1}.Dec(C_1)$
Search for match to AN in Chlngs
~~ID_A HK_{A1} $Chlng_{A1}$ HD_{A1}~~
ID_A HK_{A2} $Chlng_{A2}$ HD_{A2}
~~ID_B HK_{B1} $Chlng_{B1}$ HD_{B1}~~
ID_B HK_{B1} $Chlng_{B1}$ HD_{B1}
$AuTk_{DB}$
$[AN_{B1}, ID_A, C'_T] := HK_{B1}.Dec(C_2)$
5
If found, create a row entry for ID_A and ID_B in $SESSION_{DB}$ and add C_T and C'_T
ID_A C_T ID_B C'_T
$SESSION_{DB}$
6
Alice & Bob Key Refresh Operation:
Delete current and get next AT from $AuTk_{DB}$
$\{ACK, C_A\}$
7
$\{ACK, C_B\}$
Construct:
$C_A := HK_{A1}.Enc([ID_B, C'_T, Chlng_{A2}, HD_{A2}])$
$C_B := HK_{B1}.Enc([ID_A, C_T, Chlng_{B2}, HD_{B2}])$
2
Authentication:
select
vec_1
vec_2
Vec_{DB}
Read $\{v_{B1}, AN_{B1}, SF_{B1}, HD_{B1}\}$ from NVM
KK_{B1} := $HPUF_R(\{vecs\}, AN_{B1}, SF_{B1}, HD_{B1})$
HK_{B1} := Hash(KK_{B1})
3
Shared Session Key:
SK'_T := GenNonce()
$C'_T := n'_T.XOR(SK'_T)$
$C_1 := HK_{B1}.Enc([AN_{B1}, ID_A, C'_T])$
$\{REQ, ID_B, C_2\}$
8
$[ID_B, C'_T, Chlng_{A2}, HD_{A2}] := HK_{A1}.Dec(C_A)$
Store $\{Chlng_{A2}, HD_{A2}\}$ into NVM for next authentication.
$SK'_T := n'_t.XOR(n'_t.XOR(C'_T))$
$SK := SK'_T.XOR(SK_T)$
8
$[ID_A, C_T, Chlng_{B2}, HD_{B2}] := HK_{B1}.Dec(C_B)$
Store $\{Chlng_{B2}, HD_{B2}\}$ into NVM for next authentication.
$SK_T := n_t.XOR(n_t.XOR(C_T))$
$SK := SK_T.XOR(SK'_T)$
Alice and Bob now share session key SK

Fig. 5: Dual-sided MAKE message exchange diagram for authentication and session key generation between Alice and Bob.

multiple customer devices, and with the Issuing authority (IA) server. The AS also utilizes a sqlite3 database to implement the $AuTk_{DB}$ and $SESSION_{DB}$ components of the MAKE protocol.

- The Issuing authority (IA) is implemented as a multi-threaded application with socket communication channels to customer devices and to AS. It also utilizes a sqlite3 database to implement the Vec_{DB} and DV_{DB} components of the MAKE Enrollment protocol.
- An openSSL implementation of the AES encryption algorithm with a 256-bit key configured in CBC mode is used for all protocol encryption and decryption operations.
- An openSSL implementation of the SHA-3 256-bit hashing algorithm is used for all protocol hashing operations.

A series of experiments were carried out to evaluate the scalability of the MAKE protocol in which Enrollment and InField protocol operations were run with different numbers of Authentication Tokens (AT). The time required to carry out the various steps of the authentication protocol were measured by having Alice and Bob's device perform repeated authentications. Four experiments were performed with the AS $AuTk_{DB}$ database configured with 10, 100, 1000 and 10,000 AT. The SiRF PUF KEK bitstrings were collected over the duration of the run and analyzed to determine the statistical quality of the bitstrings.

The Entropy and MinEntropy statistics associated with the KEK bitstrings produced during the execution of the '10,000 $AuTk$ experiment' are plotted as a function of time in Fig. 6. The results for the 10,000 KEK bitstrings generated by Alice and Bob are plotted as two superimposed curves, and are analyzed in groups according to the number generated during each 1 minute time interval (x-axis) over the duration of the run. The duration of the protocol run is approx. 270 minutes. Therefore, each group includes approx. 37 KEK bitstrings. The equations for Entropy and MinEntropy are given by Eq. 1 and Eq. 2, where NB = 256, the number of bits in the KEK bitstrings.

$$\mathbf{H}(\mathbf{X}) = \sum_{i=1}^{NB} \sum_{j=0}^{1} p_{i,j} log_2(p_{i,j}) \tag{1}$$

$$\mathbf{H}_\infty(\mathbf{X}) = \sum_{i=1}^{NB} -log_2(max(p_{i,j})) \tag{2}$$

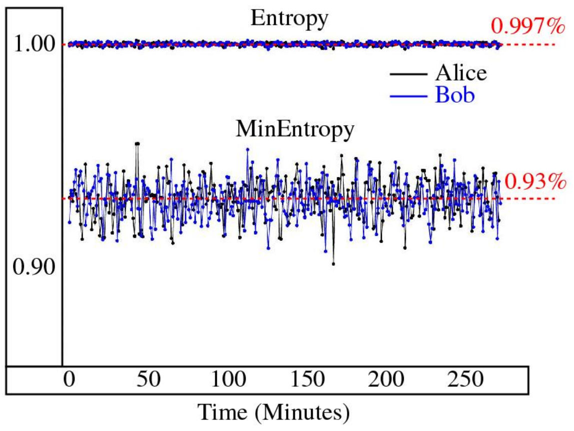


Fig. 6: Entropy and MinEntropy of the Authentication Tokens, KK_A, that are generated over a 4.5 hour run of the MAKE protocol, which produced 10,000 KK_A.

The ideal value for Entropy and MinEntropy is 1.00, which is nearly achieved for Entropy with an average value of 0.997, computed using all 10,000 256-bit bitstrings concatenated as a 2,560,000-bit bitstring. MinEntropy varies over the range of 0.92 to 0.95, which indicates that in the worst case, each KEK bit generates on average 0.93 bits of Entropy. According to the literature, the Entropy and MinEntropy statistics obtained in these experiments indicate the KEK bitstrings are of cryptographic quality.

The Interactive Authentication Bitstring Hamming Distance (HD_{IAB}) of the KEK bitstrings are plotted in Fig. 7, also as a function of one minute time intervals. The sequences of LLKs produced by Alice and Bob are analyzed separately. HD_{IAB} is computed by pairing the bitstrings within each group from the same device under all combinations. The number of bit-wise differences are summed across all pairing combinations and then divided by the total number of bits in the bitstrings from the group. Eq. 3 gives the expression for HD_{IAB}, converted to a percentage as shown in the figure. Here, NBS represents the number of bitstrings (approx. 37 per group), NB the number of bits per bitstring (256), TNB the total number of bits in each bitstring group and BS the bitstrings themselves.

$$\mathbf{HD_{IAB}} = \frac{\sum_{i=1}^{NBS} \sum_{j=i+1}^{NBS} \sum_{k=1}^{NB} (BS_{i,k} \oplus BS_{j,k})}{TNB} * 100 \qquad (3)$$

The HD_{IAB} values are very close to the ideal value of 50%, with the overall mean value across all bitstring pairing combinations (10,000*9,999/2 pairings) given as 49.997%. The $3\sigma_{IAB}$ values for each group are also depicted in the lower portion of the graph. The expected value is given by the binomial expression in Eq. 4. With NB = 256, the expected value is 9.375%. The data plotted in the figure is a very good match to the expected value. Alice and Bob's 256-bit bitstrings are concatenated to form bitstrings of length larger than 50,000 bits and evaluated using the NIST statistical test suite. All applicable NIST tests for bitstrings of this size passed as well as the p-value-of-the-p-value tests.

$$\mathbf{3\sigma_{IAB}} = \frac{3 \times \sqrt{NB * 0.25}}{NB} * 100 \qquad (4)$$

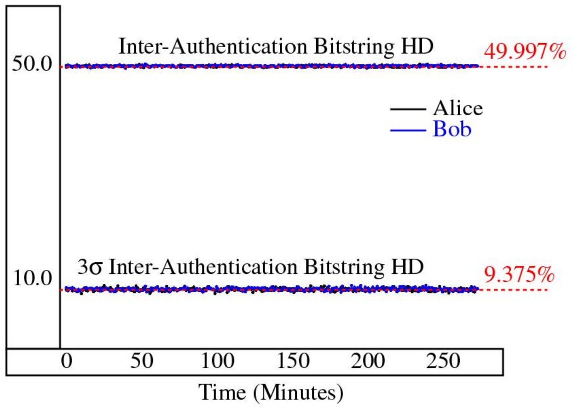


Fig. 7: Intra-chip Hamming Distance statistics for 10,000 Authentication Tokens and KK_A, produced in one minute time intervals over a 4.5 hour duration.

A separate set of experiments was conducted on a set of 120 SiRF instances on FPGAs in which the KEK regeneration was evaluated across extended industrial range temperatures, from -40°C to 100°C. The results obtained indicate that the probability of a bit-flip error is less than $1e^{-}8$, i.e, one chance in 100 million, using XMR with X set to 5.

The MAKE enrollment operations take approx. 1.75 seconds per AT generation. For example, the total amount of time to generate 10,000 AT for both Alice and Bob, and for the AT to be transmitted and stored in the $AuTk_{DB}$ on the AS is 4.87 hours.

The run times for the MAKE In-Field protocol operations are given in 8, partitioned into the 7 steps as given by the legend and plotted along the x-axis. The ordering of the steps given here is not identical, but is consistent with the sequence of operations described earlier in reference to Fig. 5. A set of bars gives the run times in seconds for each of the 4 experiments, with each carrying out different numbers of Alice-Bob authentications as shown along the y-axis. The run times are nearly constant at approx. 1.8 seconds per authentication for the first three experiments, with the $AuTk_{DB}$ database enrolled with 10 through 1000 ATs respectively. The run time increases to approx. 2.2 seconds for the 10,000 AT experiment. From the bar graph, the increase is attributed to operations carried out in Step 4. Alice and Bob are waiting for AS to return the C_A/C_B package in Step 4, so the additional time is introduced by AS, and in particular, to the database search that AS carries out during this step in the protocol. With the $AuTk_{DB}$ database populated with 10,000 AT, the search process adds to approx. 0.4 seconds to the overall runtime. In summary, the SiRF PUF meets all cryptographic bitstring quality standards with run times that enable Alice and Bob to mutually authenticate and generate a shared session key in approximately 2 seconds.

VI. Security Analysis

The protocol can be shown to be secure under strong adversarial conditions. We assume a powerful adversary with computationally infinite resources that is fully aware of the protocol and capable of spying on every channel in the system.

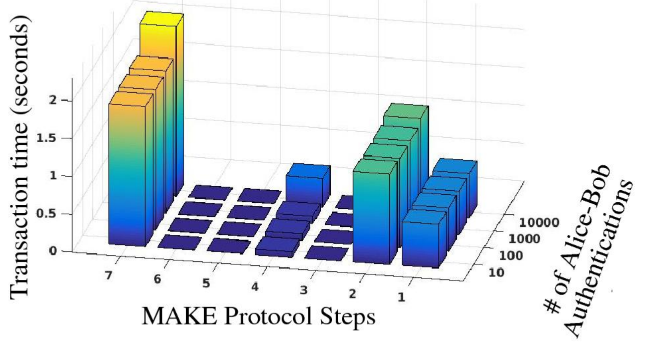


Fig. 8: Transaction times associated with various steps in the PUF-MAKE protocol: 1) n_1 exchange, generate session keys, read-out AT from DB, 2) SiRF PUF KK_A generation, hash KK_A to HK_A, 3) Create/transmit authentication packet for AS, 4) Wait for C_A/C_B from AS, 5) Decrypt C_A/C_B, XOR-decrypt C'_T, create SK, 6) Encrypt-transmit-decrypt test message with SK, and 7) Total authentication time.

A wide range of attack scenarios are considered here as well as the countermeasure defenses provided by the SiRF PUF and the proposed approach.

A. Core Properties

1) Challenge Response Search-Space: The underlying guarantees of the protocol stem from the position that the PUF is a physical random oracle [24], such that the entropy pool of the challenge-response space is vast, yet the response is reproducible at the bit level irrespective of the number of times a challenge is presented.

The SiRF PUF is synthesized with 8 million paths, each of which can be tested with a rising or falling transition, of which the algorithm selects and measures 2048 rise delays and fall delays to create 2048 differences, DVD. Therefore, the total number of such differences is $(10^7)^2$, for a total of $10^{14} = 2^{47}$ raw bits. The GPEV component of the algorithm further expands the CRP space, increasing the number of raw bits to 2^{57}. However, it is neither practical nor necessary to store 20 million DV in the DV_{DB} on IA. Instead, a smaller set of approximately 8,000 DV are stored for each device, primarily for efficiency. For each unique 256-bit KEK key, 4096 valid paths are selected from the set of 10,000 in each enrollment/regeneration cycle, which are processed by two iterations of the SiRF algorithm to generate a unique 4096-bit challenge. Note that we define the challenge as DV for convenience, which are in fact the timing values measured during the application of challenge vectors.

The response search space in terms of bits is governed by the duration of the SiRF algorithm, which is dominated by the path timing operation and as such is unbounded. Longer duration generates more bits but at the expense of time duration, representing a trade-off between security and performance. For present purposes, the path-timing operation is restricted to a duration of 1 sec. Recall that not all bits generated during enrollment will be strong bits and the number will fluctuate with each enrollment, which are corrected to a reproducible set of bits by the thresholding and XMR schemes (Fig. 2).

A probability distribution constructed from 100,000 runs of the PUF CRP results in a mean value of 152 bits and a 3-sigma deviation of 17 bits per iteration. Batch mode processing implemented by the SiRF algorithm requires two iterations to generate the required 256-bits, with 48 bits of the total 304 bits discarded on average. As such, the PUF will consume two 2048-bit challenges (sets of DV) to produce two 128-bit responses which will form a 256-bit KEK in 1 second. Note that 4096-to-304 bits represents a 13.5-to-1 mapping from the challenge search space to the response search space, and that the the key is perfectly reproducible regardless of the number of regenerations performed. Larger key sizes are possible with the same challenges at the cost of increased regeneration time. From this analysis, we conclude that the number of possible challenges is 2^{4096} and the number of possible responses is 2^{304}, of which only 2^{256} are used during authentication.

2) Model-Building: The authors of [25] have shown that a strong PUF can have its responses modeled and predicted using machine learning techniques. While a variety of designs have been compromised with this approach, a strong PUF with similar design principles to the SiRF PUF (called HELP) has been shown to be resistant to such attempts [14].

Nearly all model-building attacks to date require access to both the challenge and response bitstrings. The PUF- protocol as described herein reveals only the hash HK_A. The actual response KK_A is not revealed and discarded once the hash is generated. This processing step can be further hardened by computing the hash in a trusted environment (ARM Trust-Zone) or in hardware, eliminating all possibility of recovery. In addition, the input challenges are hidden within the construction, as challenges are specified using a 32-bit random PRNG seed v_A in PUF-MAKE. Finally, the determination of which of the 4096 measured DV are weak (and discarded) and which are strong is probabilistic, based not only on uncompensated temperature-voltage variation in the measured DV but also on the randomly-selected Spread Factors (SF). The combination of these individual factors results in the production of different KK_A even when the exact same input challenge is used. Collectively, these attributes of the protocol and the underlying hardware will make it exceedingly difficult, if not impossible, to conduct model-building attacks.

3) Unlinkability and Quantum Hardness: Building on the assumption that the PUF is a physical random oracle, key generation is a random oracle input-response operation, guaranteeing that the key is completely independent of every other key. The extremely large number of possible challenges (2^4096) and response (2^304) bitstrings ensure that Alice can generate a unique key for every authentication cycle without repetition and without collision with any other device in the system for the lifecycle of the protocol. A combination of the PUF's physical properties, the cryptographic properties of AES and SHA-3, and one-time XOR based encryption to provide operational security in the proposed approach. All of the functional blocks and cryptographic primitives in use are considered to be resistant to quantum attacks at the time of publication. Furthermore, AES is not integral to the protocol and can be substituted for any symmetric encryption scheme, similarly SHA-3 can be replaced by any cryptographic one-way function with suitable properties, guaranteeing cryptographic agility in the proposed approach.

B. Physical Attacks

1) Fault Injection: An attacker could attempt to flip one or more bits during sensitive computations to affect the outcome of the protocol. The microprocessor component of System-on-Chip (SoC) FPGAs is synthesized onto standard silicon processes and lacks specific tamper-resistant features. In the present approach, single bit-flip faults are thwarted by implementing protocol steps at the FPGA synthesis level as Verilog state machines. In doing so, the attacker would be forced to inject faults at multiple sites simultaneously to perform the attack, with a high probability risk of the attack causing the hardware to lock up. Execution of cryptographic primitives (AES, SHA-3) can be ported to hardware as well, and other necessary software functions (e.g. database query) can be transferred to a secure processing environment such as ARM TrustZone.

In terms of the core PUF primitives, fault injection is mitigated by the natural tamper-evident property of the PUF itself. Unlike cryptographic primitives, in which fault injection can be used to weaken the security, for instance by reducing the number of rounds in an AES encryption operation, fault injection performed during the SiRF PUF's delay-based timing operations would only prevent the key material from being correctly generated or reproduced. Similarly, fault injection on the state machine implementations of the SiRF algorithm post-processing operations may also result in preventing correct key reproduction. In both cases the impact is limited to a failed authentication attempt. Furthermore, the reliability enhancing techniques employed within the SiRF PUF algorithm, such as environment compensation, thresholding and XMR, protect against a portion of purposely introduced bit-flips.

2) Cloning: An adversary could attempt to create a functional clone of a PUF device. Although some types of memory-based weak PUFs have been shown to be susceptible to cloning, the same is not true of strong PUFs to date. The entropy source within the SiRF PUF is a complex network of wires and logic gates, with $\sim$ 20 million distinct paths, each producing a unique delay value (DV). Moreover, the FPGA bitstream used for provisioning is distinct from enrollment/interactive operations and therefore, it is not possible to read out the DV even in cases where the adversary has possession of the device for a short period of time. As noted previously, the AS is also a device with an instance of the SiRF PUF and is therefore afforded the same protections. It is, however, noted that the IA stores sufficient timing information that if compromised, would allow any PUF in the system to be cloned, at least until it is discovered that such an attack occurred and before the DV_{DB} is replenished with new sets of DV. For this reason, the IA would be kept in a highly secure environment with explicit network segmentation and permission controls to restrict access.

3) Exfiltration of Secrets: An attacker could attempt to extract sensitive information stored on the PUF either at rest or during execution of the protocol. Timing and side-channel attacks have been exploited effectively against PUFs in the past [26]. All devices in the system (AS, Alice and Bob) except for the IA are PUF devices. Each PUF device must be powered, activated and presented an explicit challenge to elicit a corresponding response. Given the size of the path selection space, it is impossible to ascertain the response by examining the silicon. Information stored in local non-volatile memory is encrypted at rest via the KEK key, which is an encryption key derived from the response to a challenge, ergo the key itself is derived from the same device. An attacker could attempt to exfiltrate the AES secret key during encryption/decryption operations via single or differential power analysis (SPA/DPA) techniques. While this is possible, it is noted that the window of opportunity to exfiltrate the key and make use of it in a single transaction is small, and the entire process will need to be repeated for the next authentication cycle, since each key is only used once before replacement.

An attacker could manipulate the communications channel to force Alice to make repeated authentication attempts with the same HK_A. For the present work, we used the AES version implemented in the OpenSSL library, which is susceptible to side-channel attacks such as differential power analysis [27] on common processors including ARM platforms. This attack can be mitigated by switching to a hardened hardware implementation of AES, although care must be taken during implementation to prevent leakage [28]. Note that the overall impact of revealing a key is limited to a single authentication cycle, as the AS responds with a new challenge, which does not reveal the new key to the attacker unless they manage to model the PUF itself.

4) Bitstream Manipulation: An attacker could steal the device and attempt to read out all the DV by repeating the provisioning process. This attack is prevented by disabling the provisioning operation in the bitstream of fielded devices. An adversary can attempt to reverse engineer the in-field bitstream and re-enable the read-out functionality. However, the FPGA bitstream is encrypted in persistent storage on the device and therefore the adversary would need to extract the bitstream decryption key from the FPGA to obtain an unencrypted bitstream and then go through a reverse engineering process to make changes to enable provisioning. Although possible, the process is designed to be very difficult by the FPGA manufacturer.

C. Man-in-the-Middle (MITM) Attacks

A powerful adversary may decide to intercept and modify information in the message exchange of the protocol. We can consider various points where the adversary could make such an attack and highlight how the system thwarts such attacks.

1) Enrollment: A local adversary could spy on the MAKE enrollment process. During initial enrollment, the customer device and AS carry out a PUF-based CRP mutual authentication and session key generation process with IA that is similar to the process described between the customer device and AS in the MAKE protocol. Therefore, it is not possible for an adversary to collect and manipulate plaintext information, or insert new information, between the customer device and IA. The same process is carried out between IA and AS during the refresh operation, and the same protections afforded against MITM.

2) One-time nonce: In Step 1, the attacker could intercept and replace the one-time nonces n_T and n'_T with locally generated versions, with the goal to manipulate the selection of Alice and Bob's shared session key SK. It is noted that n_T and n'_T only represent half of the secrets used to derive SK, and since both SK_T and SK'_T are shared over an AES-256 encrypted channel in Step 4, the attacker would have to break AES-256-CBC to uncover or manipulate the final session key. This is not considered possible at this time as AES-256-CBC is impervious to classical and quantum computing attacks. Ergo, replacing nonces in Step 1 constitutes at best a denial-of-service attack, where Alice and Bob receive incorrect information and cannot derive a common shared session key.

3) In-field authentication: In Step 4, Alice and Bob transmit an initial in-field authentication packet to the AS. An attacker intercepting packets on either channel could attempt to replace Alice or Bob's initial transmission with a custom payload. Note that the AS relies on a unique ID, AN, HK tuple for each PUF. If the PUF ID does not match the appropriate AN and HK_A in the database, the decryption will fail for C_1, subsequently prompting AS to abort the authentication. C_1 itself cannot be duplicated or manipulated as it is encrypted by a 256 bit AES key. Similarly, the AS's response to both Alice and Bob is also encrypted, preventing manipulation. Replacing the packet C_A in Step 7 by anything other than one encrypted by HK_A will result in Alice receiving an incomprehensible packet, forcing her to stop the protocol, with the same being true for Bob. Since neither HK_A nor HK_B appear on the channel at any point, it is impossible for the MITM attacker to derive either secret.

4) AS Spoofing and Impersonation: An adversary could attempt to spoof the AS to manipulate Alice and Bob transactions. This attack will fail at Step 4, as without the right key, the AS would not be able to decrypt C_T or C'_T. Furthermore, if the attacker transmits a fake C_A packet, Alice would know instantly as it would not be encrypted by the correct key. The AS itself is a standard infrastructure component and would be protected by standard approaches such as network segmentation and access control. To properly impersonate AS would require the adversary to break into the infrastructure and extract the AT from the database, this is considered unlikely for a suitably configured installation. Furthermore, the $AtTk_{DB}$ database is encrypted in persistent storage by a long-lived-key (LLK) produced by the SiRF PUF at boot-up. Therefore, the adversary would need to extract an in-memory copy of the LLK encryption key to obtain a plaintext copy of $AtTk_{DB}$ as PUFs do not store keys in persistent storage.

5) Interception: An attacker could deliberately attempt to disrupt the ordered sequence of AT usage during in-field operations, which could result in Alice and AS agreeing on different challenge material for the next authentication cycle. One practical attack is interception of AS's response to Alice, such that Alice does not receive the challenge information needed to generate HK_{A2}. This can also be upgraded to a known-key attack if coupled with a key exfiltration attack. Alice and the AS are now out of synchronization, since Alice will generate HK_{A1} at the next authentication whereas the AS expects HK_{A2}. It is noted that a loss of synchronization is also possible for entirely non-malicious reasons, for example if connectivity is lost between Alice and AS, or either the Alice or AS suffers a loss of power.

A common failure mode can capture both malice and connection loss, ensuring that (a) Alice and AS can resume when connectivity is restored and (b) A malicious adversary cannot take advantage of this event. At a low level, this failure mode is dealt with by relying exclusively on TCP connections, where an acknowledgement packet is received for every sent packet, such that when the AS sends the C_A to Alice in Step 6, it will wait for an return acknowledgement before deleting the preceding entry in $AuTk_{DB}$. If Alice fails to store the authentication information for some reason, due to a power loss or non-volatile memory failure on Alice's device, the AS will supply past challenges to Alice in reverse order leading back to the very first challenge supplied by the IA. If Alice can successfully authenticate against any one of the previous entries, the AS will supply the latest challenge again for authentication. Forcing the device to use the latest challenge to successfully authenticate prevents impersonating devices from performing a replay attack on the system.

D. Channel Attacks

1) Information Leakage: A passive observer could attempt to gather secret information by observing multiple rounds of the protocol. If the protocol leaks sufficient information, a passive attacker could attempt to impersonate Alice, Bob or the Authentication Server. Steps 2-8 in the protocol are encrypted, therefore the attacker learns zero information about the devices. In step 1, the attacker could keep track of nonces related to Alice and Bob, however, since these are one-time nonces generated by relying on the TRNG properties of the PUF, the attacker learns minimal information about Alice and Bob.

2) Replay: The key update mechanism relies on a unique $(AN, HK, Chlng)$ tuple, which is used only once per authentication and renewed on every authentication, mitigating all replay attacks. This use-only-once implementation practice is also utilized during enrollment (MA and SKG operations), and one-time nonces are used at critical points throughout the protocol to ensure that all aspects of the in-field authentication are impervious to replay attacks.

VII. Conclusions

In this paper, we propose and evaluate a PUF-based lightweight, mutual authentication and key exchange (MAKE) protocol. The protocol is composed of three distinct processes, namely provisioning, enrollment and in-field interactive authentication. The evaluation is performed on a set of three FPGAs that incorporate programmable logic instantiations of the SiRF PUF and KEK algorithms, and a desktop server.

The KEK response bitstrings collected over runs of the enrollment protocol are shown to be of cryptographic quality. Run times of the interactive in-field authentication operations are upper bounded at 2.2 seconds with an authentication token database constructed with up to 20,000 authentication tokens. A wide range of attack scenarios are considered, from physical attacks, e.g., side-channel, probing etc. through many types of network-based attacks. The PUF-MAKE protocol is constructed with built-in mitigations for nearly all attacks considered, and is light-weight, making the PUF-MAKE protocol a good candidate for resource-constrained devices and applications that require elevated levels of security and trust, including those that involve the exchange of digital currency for goods and services.

References

[1] J. Wurm, K. Hoang, O. Arias, A.-R. Sadeghi, and Y. Jin, "Security analysis on consumer and industrial iot devices," in *2016 21st Asia and South Pacific Design Automation Conference*, 2016, pp. 519–524.

[2] W. Diffie, P. C. Van Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes and Cryptography*, vol. 2, pp. 107–125, 1992.

[3] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Advances in Cryptology - CRYPTO' 93*. Springer, 1993, pp. 232–249.

[4] ——, "Provably secure session key distribution: The three party case," in *Twenty-Seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '95. Association for Computing Machinery, 1995, pp. 57—-66.

[5] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *Advances in Cryptology — EUROCRYPT 2000*. Springerg, 2000, pp. 232–249.

[6] T. Okamoto, "Authenticated key exchange and key encapsulation in the standard model," in *Advances in Cryptology – ASIACRYPT 2007*. Springer, 2007, pp. 474–484.

[7] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, "An efficient protocol for authenticated key agreement," *Designs, Codes and Cryptography*, vol. 28, pp. 119–134, 1995.

[8] H. Krawczyk, "Hmqv: A high performance secure diffie-hellman protocol," in *Advances in Cryptology – CRYPTO 2005*. Springer, 2005, pp. 546–566.

[9] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *International Conference on Provable Security*. Springer, 2007, pp. 1–16.

[10] J. Delvaux, R. Peeters, D. Gu, and I. Verbauwhede, "A survey on lightweight entity authentication with strong pufs," *ACM Comput. Surv.*, vol. 48, no. 2, oct 2015. [Online]. Available: https://doi.org/10.1145/2818186

[11] T. Idriss and M. Bayoumi, "Lightweight highly secure puf protocol for mutual authentication and secret message exchange," in *2017 IEEE Int. Conf. on RFID Technology Application*, 2017, pp. 214–219.

[12] M. H. Mahalat, S. Saha, A. Mondal, and B. Sen, "A puf based light weight protocol for secure wifi authentication of iot devices," in *2018 8th Int. Symp. on Embed. Comp. and System Design*, 2018, pp. 183–187.

[13] M. H. Mahalat, D. Karmakar, A. Mondal, and B. Sen, "Puf based secure and lightweight authentication and key-sharing scheme for wireless sensor network," *Journal of Emerging Technologies in Computer Systems*, vol. 18, no. 1, Sep 2021.

[14] W. Che, M. Martin, G. Pocklassery, V. K. Kajuluri, F. Saqib, and J. Plusquellic, "A privacy-preserving, mutual puf-based authentication protocol," *Cryptography*, vol. 1, no. 1, 2017.

[15] 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 Transactions on Dependable and Secure Computing*, vol. 16, no. 3, pp. 424–437, 2019.

[16] U. Chatterjee, R. S. Chakraborty, and D. Mukhopadhyay, "A puf-based secure communication protocol for iot," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 3, pp. 1–25, 2017.

[17] J. R. Wallrabenstein, "Practical and secure iot device authentication using physical unclonable functions," in *2016 IEEE 4th Int. Conference on Future Internet of Things and Cloud*, 2016, pp. 99–106.

[18] M.-D. Yu, M. Hiller, J. Delvaux, R. Sowell, S. Devadas, and I. Verbauwhede, "A lockdown technique to prevent machine learning on pufs for lightweight authentication," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 146–159, 2016.

[19] J. Zhang and G. Qu, "Physical unclonable function-based key sharing via machine learning for iot security," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 8, pp. 7025–7033, 2020.

[20] (2021) Ic-safety, llc. [Online]. Available: http://ic-safety.com

[21] J. Ju, R. Chakraborty, C. Lamech, and J. Plusquellic, "Stability analysis of a physical unclonable function based on metal resistance variations," in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 143–150.

[22] D. Heeger and J. Plusquellic, "Analysis of iot authentication over lora," in *2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2020, pp. 458–465.

[23] *ZYBO-Z7 Reference Manual*, Digilent Corporation, Pullman, WA. [Online]. Available: https://digilent.com/shop/zybo-z7-zynq-7000-arm-fpga-soc-development-board/

[24] M. van Dijk and U. Rührmair, "Physical unclonable functions in cryptographic protocols: Security proofs and impossibility results," Cryptology ePrint Archive, Report 2012/228, 2012, https://ia.cr/2012/228.

[25] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proc. ACM Conf. on Computer and Communications Security*, 2010, pp. 237–249.

[26] 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 *Cryptographic Hardware and Embedded Systems*. Springer Berlin Heidelberg, 2014, pp. 476–492.

[27] C. Ramsay and J. Lohuis, "TEMPEST Attacks against AES," Hardwear.IO, 2017. Accessed Nov 16, 2021. [Online]. Available: https://hardwear.io/document/slides-craig-ramsay.pdf

[28] D. Das and S. Sen, "Electromagnetic and power side-channel analysis: Advanced attacks and low-overhead generic countermeasures through white-box approach," *Cryptography*, vol. 4, no. 4, 2020.

Cyrus Minwalla is a Technical Researcher and Security Lead within the Financial Technology Research group at the Bank of Canada. His research interests include digital currencies, cryptography, embedded devices, and Internet-of-Things. He received his Ph.D. degree in Computer Engineering from York University in Canada. Cyrus was selected as NRC's Top Scientist Under 40 in 2017 and received the Bank of Canada's Award of Excellence in 2020.

Eirini Eleni Tsiropoulou is currently an Assistant Professor at the Department of Electrical and Computer Engineering, University of New Mexico. Her main research interests lie in the area of cyber-physical social systems and wireless heterogeneous networks, with emphasis on network modeling and optimization, resource orchestration in interdependent systems, reinforcement learning, game theory, network economics, and Internet of Things. Five of her papers received the Best Paper Award at IEEE WCNC in 2012, ADHOCNETS in 2015, IEEE/IFIP WMNC 2019, INFOCOM 2019 by the IEEE ComSoc Technical Committee on Communications Systems Integration and Modeling, and IEEE/ACM BRAINS 2020. She was selected by the IEEE Communication Society - N2Women - as one of the top ten Rising Stars of 2017 in the communications and networking field. She received the NSF CRII Award in 2019 and the Early Career Award by the IEEE Communications Society Internet Technical Committee in 2019.

Jim Plusquellic is a Professor in Electrical and Computer Engineering at the University of New Mexico. He received both his M.S. and Ph.D. degrees in Computer Science from the University of Pittsburgh. Professor Plusquellic received an "Outstanding Contribution Award" from IEEE Computer Society in 2012 and 2017 for co-founding and for his contributions to the Symposium on Hardware-Oriented Security and Trust (HOST).