PUF-Based Digital Money with Propagation-of-Provenance and Offline Transfers Between Two Parties

BENJAMIN BEAN*, University of New Mexico, United States
CYRUS MINWALLA*, Bank of Canada, Canada
EIRINI ELENI TSIROPOULOU*, University of New Mexico, United States
JIM PLUSQUELLIC*, University of New Mexico, United States

Building on prior concepts of electronic money (eCash), we introduce a digital currency where a physical unclonable function (PUF) engenders devices with the twin properties of being verifiably enrolled as a member of a legitimate set of eCash devices and of possessing a hardware-based root-of-trust. A hardware-obfuscated secure enclave (HOSE) is proposed as a means of enabling a PUF-based propagation-of-provenance (POP) mechanism, which allows eCash tokens (**eCt**) to be securely signed and validated by recipients without incurring any third party dependencies at transfer time. The POP scheme establishes a chain of custody starting with token creation, extending through multiple bilateral in-field transactions, and culminating in redemption at the token-issuing authority. A lightweight mutual-zero-trust (MZT) authentication protocol establishes a secure channel between any two fielded devices. The POP and MZT protocols, in combination with the HOSE, enables transitivity and anonymity of **eCt** transfers between online and offline devices.

CCS Concepts: • **Security and privacy** → **Hardware-based security protocols**.

Additional Key Words and Phrases: electronic money, digital currency, physical unclonable functions

ACM Reference Format:
Benjamin Bean, Cyrus Minwalla, Eirini Eleni Tsiropoulou, and Jim Plusquellic. 2023. PUF-Based Digital Money with Propagation-of-Provenance and Offline Transfers Between Two Parties. *J. ACM* 1, 1, Article 1 (January 2023), 25 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

An electronic money (eCash) ecosystem defines a set of protocols and security properties for enabling the creation and validation of electronic cash tokens (**eCt**), as well as secure transfers between end-users. A typical incarnation consists of a token-issuer (TI), e.g., a central bank, a set of financial institutions (FI), e.g., commercial banks, and a consumer population. The TI defines the root-of-trust for the entire ecosystem, while the FIs provide accounting services to subsets of the consumer population.

Electronic money presents unique challenges not commonly associated with security protocols providing, e.g., a secure communication channel. An inherent property of paper money is anonymity, which hides the identity of the purchaser of goods and services. Although anonymity is a desirable

*Authors contributed equally to this research.

Authors' addresses: Benjamin Bean, bbean@unm.edu, University of New Mexico, P.O. Box 1212, Albuquerque, New Mexico, 43017-6221, United States; Cyrus Minwalla, cminwalla@bank-banque-canada.ca, Bank of Canada, 234 Wellington St., Ottawa, K1A0G9, Canada; Eirini Eleni Tsiropoulou, eirini@unm.edu, University of New Mexico, P.O. Box 1212, Albuquerque, New Mexico, 43017-6221, United States; Jim Plusquellic, jplusq@unm.edu, University of New Mexico, P.O. Box 1212, Albuquerque, New Mexico, 43017-6221, United States.

0004-5411/2023/1-ART1 $15.00
https://doi.org/XXXXXXX.XXXXXXX

feature in eCash, it also substantially reduces the barrier for adversaries to create counterfeits. By contrast, fiat money is naturally resistant to counterfeiting, owing to the cost of the paper and inks, but copying digital representations of **eCt** is trivial and nearly cost-free. Similarly, dishonest users who engage in the double spending, i.e., use the same **eCt** to purchase goods and services from different vendors, is difficult to prevent, particularly for offline value transfer operations where users are not able to consult with a trusted-third party (TTP).

Supplemental to these primary goals of an eCash system is a mechanism for identifying malicious actors, who might engage in attacks on other devices or who may collude to launder money. Yet another desirable property is a mechanism to recover lost funds, which would occur if the device is lost or stolen, or is destroyed. The implementation of these supplemental services requires disclosure of the user's identity, and therefore, eCash protocols must resolve these issues through coordination across multiple TTPs, as a means of preventing abuse by the TTP themselves. Although our protocol stores information that the TI and FI could use for handling malicious actors and fund recovery, we do not describe the message exchanges and actions required to implement them in the protocol diagrams.

In this paper, we propose PUF-Cash, which possesses the following characteristics:

- A token-issuer (TI) provides a PUF-based root-of-trust for the entire system, and is able to extend the root-of-trust to trusted intermediaries and in-field devices using a lightweight mutual-zero-trust (MZT) protocol.
- End-user (in-field) devices incorporate a strong PUF hardware security primitive, encapsulated in hardware-obfuscated secure enclave (HOSE). The HOSE enables the device to perform security functions related to the management and validation of **eCt**. The HOSE is completely independent of the untrusted microprocessor environment which is typically co-located on the same SoC device.
- A propagation-of-provenance (POP) scheme is proposed which enables transactions between entities receiving **eCt** to authenticate the **eCt** of the issuing parties. Moreover, the receiving parties can perform this authentication without the involvement of a TTP. The entire sequence of such authentications establishes provenance back to the TI which created the **eCt**.
- The proposed PUF-Cash system supports unlimited transistivity where Alice can pay Bob, and Bob can pay Charlie without the need for any party to interact with a TTP.

Note that the strong PUF and HOSE provide unique capabilities, and in fact, define orthogonal security components of the system. The strong PUF protects keys and data at rest by eliminating the need to store keys in an NVM, and by encrypting protocol artifacts, i.e., those stored in databases on user devices. The HOSE, on the other hand, leverages the PUF-based keys to protect secrets in motion and in use, namely, payment information, customer credentials and the **eCt** themselves.

The remainder of this paper is organized as follows. Section 2 describes previous related work, and Section 3 summarizes relevant features of the shift-register reconvergent-fanout (SiRF) PUF. Section 4 describes the proposed mutual-zero-trust authentication and key exchange protocol, while Section 5 presents the details of the PUF-Cash message exchange protocol. The experiment setup is described in Section 6, with experimental results given in Section 7. A security analysis is presented in Section 8, and our conclusions in Section 9.

2 PREVIOUS WORK

2.1 Mutual Authentication and Session Key Exchange

Recent efforts around authentication utilize PUFs as a local root of trust to augment or substitute asymmetric key pairs. An extensive literature review on previously proposed lightweight PUF-based authentication protocols is given in [7]. The authors of [12] 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. [17] 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 [16]. 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 [6]. 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 [3], improving upon on the attack resilience and performance overhead of the previous method [2]. 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 [23], 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.

Yu et al. [25] 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 [27], 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. The authors of [9], [18] propose a lightweight edge-device authentication method which uses a weak SRAM PUF to generate a secret ID. A hamming distance-based ID matching scheme is proposed as a lightweight reliability enhancement method. Although XOR and secure hash are used in our protocol as well, their protocol requires the device to store a secret key in an NVM, a vulnerability that we avoid.

2.2 Electronic Money

Electronic money and digital currency are synonymous concepts in the literature. Viable solutions satisfy the properties of provenance, protection against double-spending attacks, and privacy in payments. Chaum, Fiat, and Naor (CFN) explored anonymous payments through blind signatures [4], [5]. CFN is unsuitable for offline transfers, as funds need to be validated at the time of transaction. More recently, blind signatures are incorporated in the GNU Taler system as a privacy-preserving primitive for electronic money [14].

Cash-like anonymity and privacy are defined in [10] as the ability to exchange money without tracing the exchange and without revealing the identities of the payer or payee. In their 2021 paper, they also note that there exist no token-based eCash systems that have all the same peer-to-peer (P2P), offline, and anonymity capabilities as account-based systems, and that offline double-spending cannot yet be prevented.

Up to this point, other offline eCash systems [26] [15] [1] [20] have tried to solve the double-spending problem by tracing eCash in bank and wallet records. The authors of [24] address double-spending by executing transactions in an ARM processor's Trusted Execution Environment (TEE) secured by an SRAM PUF. Similarly, the authors of [19] propose a P2P offline payment scheme that prevents double-spending, preserves privacy and anonymity and is reasonably low-powered.

Their scheme utilizes an mobile phone's TEE to execute a one-time-program within a sandbox, that executes on a single input and then self-destructs.

3 HARDWARE SECURITY PRIMITIVES

The authentication bitstrings, encryption keys and random nonces utilized within the proposed MZT and PUF-Cash protocols are derived from a strong physical unclonable function (PUF) called the shift-register reconvergent-fanout (SiRF) PUF [21]. The SiRF PUF is the physical root-of-trust in the PUF-Cash ecosystem, across TI, the FI and end-user devices.

3.1 SiRF PUF Architecture

The SiRF PUF architecture and algorithm are shown on the left and right sides of Fig. 1, respectively. The SiRF PUF utilizes within-die variations in a set of path delays as a source of entropy. Path delays are measured through an engineered netlist of shift-registers and logic gates constructed with fan-in and fan-out, referred to as reconvergent-fanout, to create an exponentially diverse matrix of signal paths that traverse a rectangular region of the FPGA fabric (22x23 CLBs). The architecture incorporates a mode switch to enable the entropy source and algorithm to be used as a true-random number generator (TRNG), which is able to supply an exponentially large number of random nonces for cryptographic operations [13].

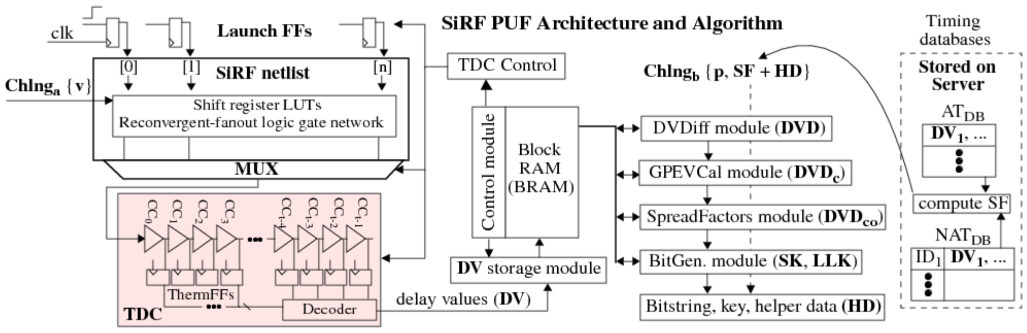


Fig. 1. SiRF PUF Architecture and Algorithm.

AT_{DB}:	Anonymous timing DB	NAT_{DB}:	Non-anon. timing DB	**DV**:	Delay values
DVD:	**DV** differences	$\mathbf{DVD_c}$:	Calibrated **DV**	$\mathbf{DVD_{co}}$:	Entropy optimized **DV**
p:	LFSR seed, range, threshold	**SF**:	SpreadFactors 2048 bytes	**HD**:	helper data 2048 bits
v:	seed specifying Chlng vecs	$\mathbf{Chlng_a}\{\mathbf{v}\}$:	Set of challenge vectors	$\mathbf{Chlng_b}\{\mathbf{p}, \mathbf{SF}\}$:	One set of **p** and **SF**
ID_x:	Device ID	**SK**:	Session key	**LLK**:	Long-lived key

3.2 Challenge and Response Construction

The challenge for the SiRF PUF consists of two components, labeled $\mathbf{Chlng_a}$ and $\mathbf{Chlng_b}$ in Fig. 1. The $\mathbf{Chlng_a}$ component controls the configuration of the paths through the netlist of shift-registers and logic gates [21]. The **v** argument is an 32-bit LFSR seed, that is used to select a sequence of binary vectors from a $Vecs_{DB}$ database stored locally on the device (not shown here but included in the message exchange diagrams in this paper). Paths are timed by the *TDC Control* module, which launches a set of rising transitions into the SiRF netlist using the **Launch FFs**. A path output is selected using the *MUX* shown along the bottom left in the figure, which routes emerging signal transitions to a time-to-digital converter (TDC). The TDC creates high resolution digitized representations (*delay values* or **DVs**) of the path delays. The *Control module* carries out a sequence of path timing operations using a sequence of **v** vectors to produce a set of 2048 rising delay values

($\mathbf{DV_R}$) and a set of 2048 falling delay values ($\mathbf{DV_F}$). The **DV** are stored in a BRAM located within the programmable logic and are used as inputs to a post-processing algorithm for authentication bitstring and encryption key generation.

The remaining components of the challenge specified in $\mathbf{Chlng_b}$ are given as **p**, **SF** and **HD**. The **p** component specifies parameters to the SiRF PUF algorithm, which are used as input to a sequence of mathematical operations applied to the 4096 **DV** values stored in the BRAM. The first operation, carried out by the *DVDiff module*, creates 2048 differences (**DVD**) by subtracting unique pairings of $\mathbf{DV_F}$ from $\mathbf{DV_R}$. Note that there are 2^{22} or 4 million unique **DVD** that can be generated from the sets of rise and fall **DV**. The LFSR seed parameter component of **p** specifies a single unique set of 2048 **DVD** to use in subsequent steps, described in detail in [21].

The **DVD** are then calibrated using the *GPEVCal module* to remove variations in delay introduced by global performance differences, as well as non-nominal temperature and supply voltage environmental conditions, and are designated as $\mathbf{DVD_c}$. The *SpreadFactors module* applies **SF** to remove delay bias introduced by differences in the length of the tested paths, to produce $\mathbf{DVD_{co}}$. The $\mathbf{DVD_{co}}$ are then converted into an authentication bitstring or key by the *BitGen module*. The **HD** component refers to helper data that is needed by the *BitGen module* for regeneration. The components of the challenge, including the **SF** and the helper data **HD** produced during enrollment, are stored in a database to enable the device to regenerate the bitstring or key at any point in the future and potentially under adverse environmental conditions.

3.3 Strong Timing-Based Authentication and Key Generation Protocol Primitives

Mutual authentication (MA) and session key generation (SKG) is accomplished using one of two low-level PUF-based protocols. The timing-based (TB) version requires AT_{DB} and NAT_{DB} timing databases stored on the Token Issuer (TI) (see Fig. 1). The timing databases encapsulate and compress a subset of the challenge-response space of each device, and are used exclusively by the Token Issuer (TI) to carry out MA and SKA with other entities in PUF-Cash ecosystem.

The TB functions utilized by the TI are annotated in the diagrams as MA_{NA}, MA_{A}, SKG_{NA} and SKG_{A}. The subscript NA refers to non-anonymous mutual authentication, where the authenticating device ID, e.g., ID_x, is derived privately by the TI using the NAT_{DB}. The subscript A refers to an anonymous mutual authentication, where the TI is able to confirm that the device has been provisioned, but is not able to identify the end-user's identity.

The AT_{DB} and NAT_{DB} are built with distinct **DV**, and the ordering of the device timing data sets in the two databases is scrambled to generate anonymity. The SiRF PUF has 16 million unique paths that can be timed, making it possible to select large numbers of unique **DV** per database. For the current experiments, the number of **DV** per device stored in the databases is limited to 10,144 (5072 $\mathbf{DV_R}$ and 5072 $\mathbf{DV_F}$). Lighter-weight versions of MA and SKG (referred to as mutual-zero-trust or MZT) are annotated in the diagrams as MA_{MZT} and SKG_{MZT}. The MZT versions are based on authentication tokens, and are used between pairs of devices and between a device and a FI.

Additional PUF-based security functions leveraged in the proposed MZT and PUF-Cash protocols include true-random-number-generation (TRNG) and long-lived key (**LLK**) generation. As key generation requires reliable reproduction of bitstrings, the functions leverage three reliability enhancing techniques integrated into the SiRF PUF algorithm, namely *GPEVCal*, thresholding and XMR, details of which are available in [21]. To summarize, the cummulative benefits of these lightweight reliability-enhancement techniques reduce the probability of a bit-flip error to $< 1e^{-6}$.

4 MUTUAL-ZERO-TRUST PROTOCOL

An eCash payment system should support trusted payment transactions in any type of environment, including scenarios in which the payer and payee cannot consult with a trusted third party (TTP) to assist with authentication i.e. offline mode. In such cases, trust within the two-party system must be derived from the devices themselves. We use the term mutual-zero-trust (MZT) to describe this scenario, in contrast to environments in which peers (other customers) can be consulted to build a trusted relationship, or a TTP is available to provide authentication credentials for the two entities.

Supporting transactions which might occur between arbitrary pairs of customer devices in an offline context, where neither device has connectivity to a remote service, means that each device must store MZT data about other customer devices. The data composition must be compact to be practical for an embedded system, while simultaneously providing each device with sufficient confidence that the established channel and the counter-party device can be trusted. In this section, we describe a MZT system that meets these requirements.

4.1 MZT Enrollment Operations

The security properties of the MZT protocol are derived from the SiRF PUF primitive, a secure hash function and a symmetric encryption algorithm. The PUF serves as the root-of-trust and is the source of entropy, secure hash provides obfuscation and data integrity while the symmetric encryption algorithm provides confidentiality. The integration of the three primitives provides a highly secure and lightweight mechanism for enabling Alice and Bob to exchange sensitive data.

The protocol requires the SiRF PUF to generate a 256-bit long-lived key (**LLK**) and a 256-bit nonce **n**. These bitstrings are used as input to a hash function to create an authentication token (AT), referred to as **ZHK**, with Z referring to zero-trust, H for secure hash and K for **LLK**. In our implementation, SHA-3 is used as the secure hash function. The **ZHK** authentication tokens are created using the relation given by Eq. 1.

$$\mathbf{ZHK} := \text{Hash}(\mathbf{LLK} \oplus \mathbf{n}) \tag{1}$$

The MZT protocol requires Alice and Bob to carry out an enrollment operation with the Token Issuer (TI). Although shown together here, enrollment is carried out separately and usually at different times by Alice and Bob. The following sequence of operations, corresponding to the message exchange diagram of Fig. 2, defines MZT enrollment.

(1) Alice and Bob authenticate non-anonymously via MA_{NA} and generate session keys $\mathbf{SK_{TA}}$ and $\mathbf{SK_{TB}}$ using SKG_{NA} with the TI. As discussed in Section 3.3, Alice and Bob use the TB versions of these security functions since they are interacting with the TI. Non-anonymous authentication allows the TI to identify Alice and Bob as ID_A and ID_B, respectively. Note that the MA_{NA} protocol is privacy-preserving, i.e., the TI derives the IDs based on the responses provided by Alice and Bob without the need to transmit their IDs openly.
(2) The TI encrypts and then transmits unique challenges to Alice and Bob, labeled $\mathbf{Chlng_{ZTa/b}}$.
(3) Alice and Bob decrypt and apply their respective challenges to their hardware PUFs, HPUF_E, in enrollment mode, to generate a long-lived key, $\mathbf{LLK_{MZT}}$, and helper data **HD**. Alice and Bob store the challenge information in the MZT_LLK_{DB} under challenge number CN_1, which includes the components discussed earlier in reference to Fig. 1, i.e., $\mathbf{v_1}$, $\mathbf{p_1}$, $\mathbf{SF_1}$ and $\mathbf{HD_1}$, to enable regeneration of $\mathbf{LLK_{MZT}}$ later in the field. As noted above, $\mathbf{v_1}$ is a 32-bit LFSR seed used to select binary vectors from the Vec_{DB} stored locally on Alice and Bob's devices.
(4) Once the $\mathbf{LLK_{MZT}}$ is generated, the TI sends a request to Alice and Bob to generate x **ZHKs**.
(5) Alice and Bob construct a set of tuples $\{\mathbf{ZHK_i}, \mathbf{n_i}\}$ by running their PUFs' TRNGs to generate a sequence of nonces, $\mathbf{n_i}$, which are XORed with $\mathbf{LLK_{MZT}}$ and used as the input to the AT creation function given by Eq. 1.
(6) The tuples are encrypted using $\text{SK}_{TA/B}$ and transmitted to the TI. The TI decrypts and stores them along with the field device's identifier $\text{ID}_{A/B}$ in its MZT_AT_{DB}.

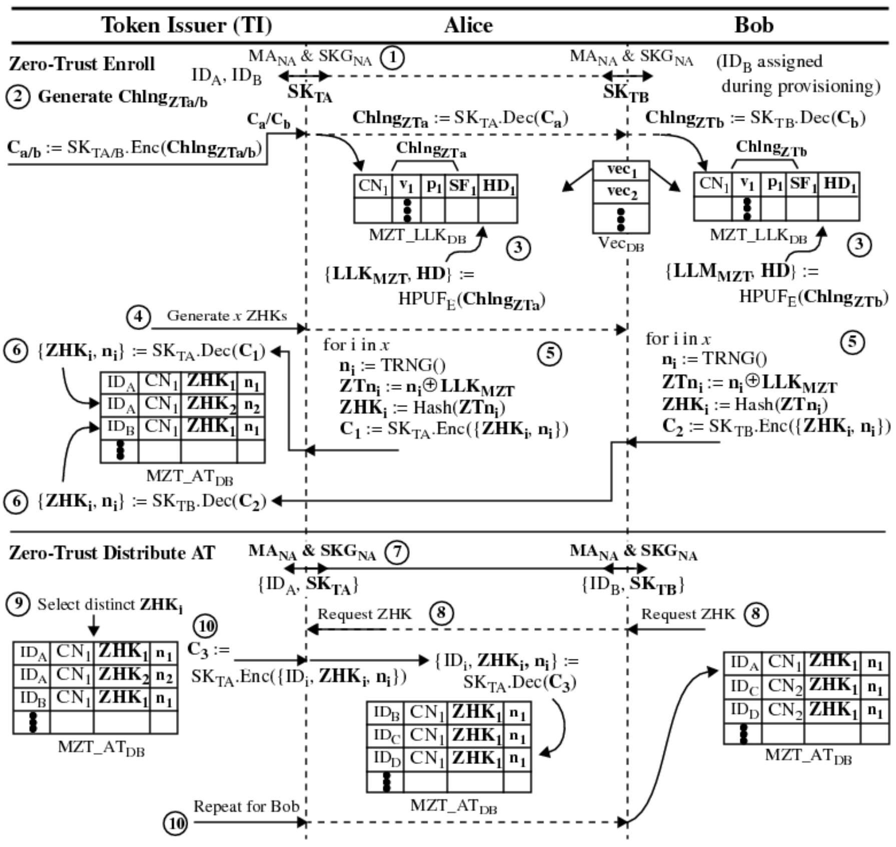

Token Issuer (TI)
Alice
Bob
Zero-Trust Enroll
MA_NA & SKG_NA ①
ID_A, ID_B
SK_TA
SK_TB
(ID_B assigned during provisioning)
② Generate Chlng_ZTa/b
C_a/C_b
C_a/b := SK_TA/B.Enc(Chlng_ZTa/b)
Chlng_ZTa := SK_TA.Dec(C_a)
Chlng_ZTb := SK_TB.Dec(C_b)
CN_1 | v_1 | p_1 | SF_1 | HD_1
MZT_LLK_DB
vec_1
vec_2
Vec_DB
③
{LLK_MZT, HD} := HPUF_E(Chlng_ZTa)
{LLM_MZT, HD} := HPUF_E(Chlng_ZTb)
④ Generate x ZHKs
⑤
for i in x
n_i := TRNG()
ZTn_i := n_i ⊕ LLK_MZT
ZHK_i := Hash(ZTn_i)
C_1 := SK_TA.Enc({ZHK_i, n_i})
C_2 := SK_TB.Enc({ZHK_i, n_i})
⑥ {ZHK_i, n_i} := SK_TA.Dec(C_1)
⑥ {ZHK_i, n_i} := SK_TB.Dec(C_2)
MZT_AT_DB
Zero-Trust Distribute AT
MA_NA & SKG_NA ⑦
{ID_A, SK_TA}
{ID_B, SK_TB}
Request ZHK ⑧
⑨ Select distinct ZHK_i
⑩ C_3 := SK_TA.Enc({ID_i, ZHK_i, n_i})
{ID_i, ZHK_i, n_i} := SK_TA.Dec(C_3)
⑩ Repeat for Bob

Fig. 2. MZT enrollment and authentication process between Token Issuer, Alice and Bob.

MA_{NA}:	Non-anon. mutual authen.	SKG_{NA}:	Session key generation	$ID_{A/B}$:	Alice/Bob ID
$\mathbf{SK_{TA}}$:	TI-Alice 256-bit key	$\mathbf{Chlng_{ZT}}$:	Chlng components {**v**, **p**, **SF**}	$\mathbf{LLK_{MZT}}$:	256-bit long-lived-key
$\mathbf{n_i}$:	nonce bit vector	$\mathbf{ZTn_i}$:	256-bit intermediary	$\mathbf{C_x}$:	Cipher text packet
$\mathbf{ZHK_i}$:	Authen. token	MZT_LLK_{DB}:	Long-lived key DB	MZT_AT_{DB}:	Authen. token DB
$CN_{1/2}$:	Challenge number				

The TI collects $\mathbf{ZHK_i}$ from all field devices to build the MZT_AT_{DB}. Each entry consumes only 72 bytes, consisting of a 4-byte integer for the ID_x and CN_x fields, and 32 bytes for each of the $\mathbf{ZHK_i}$ and $\mathbf{n_i}$ fields. Alice and Bob will retrieve one unique $\mathbf{ZHK_i}$ tuple from the TI for each device during the distribution operations, labeled as steps 7 through 10 along the bottom of Fig. 2, which they store in their MZT_AT_{DB} databases. Note that the protocol and database structure support multiple challenges, e.g., Bob stores $\mathbf{ZHK_i}$ for a second challenge number, CN_2, in his $\mathbf{ZHK_i}$. The MZT_AT_{DB} database can be stored in a standard off-the-shelf NVM, e.g., SD card, where storage capacities of 4 GB or larger are common. Elements within the MZT_AT_{DB} can be encrypted for additional security, and processing may be carried out in the HOSE on the device.

4.2 MZT In-Field Operations

The MZT in-field process is carried out when Alice contacts Bob for goods or services in an environment where a direct communication channel is established between the two parties. The message exchange protocol is shown in Fig. 3, and is described as follows:

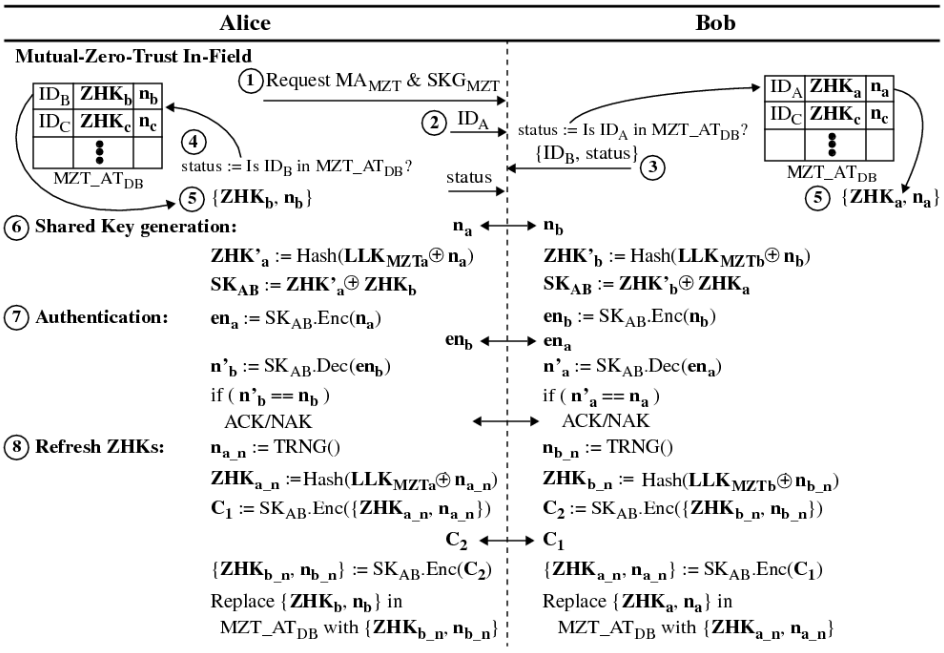

Alice
Bob
Mutual-Zero-Trust In-Field
1 Request MA_MZT & SKG_MZT
2 ID_A
status := Is ID_A in MZT_AT_DB?
{ID_B, status}
3
4
status := Is ID_B in MZT_AT_DB?
status
MZT_AT_DB
ID_B ZHK_b n_b
ID_C ZHK_c n_c
ID_A ZHK_a n_a
5 {ZHK_b, n_b}
5 {ZHK_a, n_a}
6 Shared Key generation:
n_a ↔ n_b
ZHK'_a := Hash(LLK_MZTa ⊕ n_a)
SK_AB := ZHK'_a ⊕ ZHK_b
ZHK'_b := Hash(LLK_MZTb ⊕ n_b)
SK_AB := ZHK'_b ⊕ ZHK_a
7 Authentication:
en_a := SK_AB.Enc(n_a)
en_b := SK_AB.Enc(n_b)
en_b ↔ en_a
n'_b := SK_AB.Dec(en_b)
n'_a := SK_AB.Dec(en_a)
if (n'_b == n_b)
if (n'_a == n_a)
ACK/NAK
ACK/NAK
8 Refresh ZHKs:
n_a_n := TRNG()
n_b_n := TRNG()
ZHK_a_n := Hash(LLK_MZTa ⊕ n_a_n)
ZHK_b_n := Hash(LLK_MZTb ⊕ n_b_n)
C_1 := SK_AB.Enc({ZHK_a_n, n_a_n})
C_2 := SK_AB.Enc({ZHK_b_n, n_b_n})
C_2 ↔ C_1
{ZHK_b_n, n_b_n} := SK_AB.Enc(C_2)
{ZHK_a_n, n_a_n} := SK_AB.Enc(C_1)
Replace {ZHK_b, n_b} in MZT_AT_DB with {ZHK_b_n, n_b_n}
Replace {ZHK_a, n_a} in MZT_AT_DB with {ZHK_a_n, n_a_n}

Fig. 3. Mutual-zero-trust in-field authentication and session key generation between Alice and Bob.

(1) The transaction begins with Alice sending Bob a request to authenticate and to generate a shared session key.
(2) Alice sends Bob an identifier, ID_A, that allows Bob to locate the her AT in his MZT_AT$_{DB}$.
(3) Bob responds to Alice with an ACK or NAK (labeled *status*) on whether or not he possesses an AT for Alice. He also transmits his own identifier, ID_B.
(4) Alice determines if she has an AT for Bob in her MZT_AT$_{DB}$ using Bob's ID_B, and transmits a corresponding ACK or NAK to Bob in the *status* message.
(5) Assuming both Alice and Bob have ATs for each other (otherwise the transaction is cancelled), Alice and Bob retrieve the AT for the other party from their MZT_AT$_{DB}$, which is represented by the tuple $\mathbf{ZHK_x}$, $\mathbf{n_x}$ with $x := b$ or a, respectively.
(6) **Shared Key Generation:** Alice and Bob exchange the nonce components, $\mathbf{n_x}$, of the ATs. Both parties regenerate their long-lived keys $\mathbf{LLK_{MZTx}}$ using challenge information stored in their MZT_LLK$_{DB}$ (not shown), and then compute a local version of the $\mathbf{ZHK'_x}$ using Hash($\mathbf{LLK_{MZTx}} \oplus \mathbf{n_x}$). Alice and Bob create a shared key $\mathbf{SK_{AB}}$ by XOR'ing the local copy of $\mathbf{ZHK'_x}$ with the $\mathbf{ZHK_x}$ that they store for the other party in their MZT_AT$_{DB}$.
(7) **Authentication:** Authentication begins with Alice and Bob encrypting the $\mathbf{n_x}$ they received from the other party with the newly created shared key $\mathbf{SK_{AB}}$ to create $\mathbf{en_x}$. Alice and Bob exchange the encrypted nonces $\mathbf{en_x}$, decrypt them using the shared key and then compare the $\mathbf{n_x}$ with the ones they store in their MZT_AT$_{DB}$. The status of the comparison is shared with the other party via an ACK or NAK. At this point, they have authenticated and posses a shared key assuming both have acknowledged that the $\mathbf{n_x}$ match their own local copies.
(8) **Refresh ZHKs:** Alice and Bob generate new nonces $\mathbf{n_{x\_n}}$ by running their TRNGs, and then compute new AT by hashing ($\mathbf{LLK_{MZTx}} \oplus \mathbf{n_{x\_n}}$). They encrypt the new AT as $\mathbf{C_x}$ with their shared session keys $\mathbf{SK_{AB}}$. They exchange the $\mathbf{C_x}$, and then decrypt the $\mathbf{C_x}$ to recover the new AT. They store the new AT in their MZT_AT$_{DB}$, replacing the existing AT just used.

The MZT in-field operations are lightweight using only the PUF, secure hash, and symmetric encryption functions. The refresh operation ensures that future transactions can occur between the two parties without either party needing to return to the TI to obtain additional AT. The new AT are generated by Alice and Bob's PUFs and therefore have the same strong security properties as the original AT that are replaced. The protocol is not subject to desynchronization attacks in which an adversary prevents, e.g., Alice from receiving $\mathbf{C_2}$, because Bob does not record state information that Alice stores in her MZT_AT$_{DB}$. If $\mathbf{C_2}$ is not received, then the refresh operation is not performed. Section 8 expounds further on the protocol's security properties.

5 PUF-CASH PROTOCOL

The operating environment used in the PUF-Cash protocol described here is characterized as offline, requiring Alice and Bob to authenticate using the MZT model. By assuming a bilateral, peer-to-peer trust model, we ensure that two parties can transact regardless of the underlying communications medium. Additionally, transitivity of **eCt** tokens across multiple parties is supported, where transitivity means that a token can be transferred to a new party without requiring synchronization with the token issuer (TI) or any other back-end system. PUF-Cash introduces a novel security primitive called propagation-of-provenance (POP) and a hardware-obfuscated secure enclave (HOSE) as a means of ensuring transitivity in a secure manner. The HOSE prevents Alice from double-spending her own **eCt**. POP and HOSE together enable each party in a chain of **eCt** transfers to authenticate the **eCt** that they receive, and to validate provenance back to the point of origin, namely the TI. Note that POP is a form of remote attestation, where Alice will prove the authenticity of her **eCt** to a remote party, i.e. Bob, and eventually, to the TI.

The HOSE is implemented as a set of state machines embedded in a hard-wired portion of the device, or, in the case of FPGAs, in the programmable logic. The HOSE limits interactions and potential attacks from malicious software applications, including the PUF-Cash software components, through an interface consisting of two 32-bit hardware registers. The HOSE incorporates the SiRF PUF for generating keys on-the-fly as needed. Similarly, all sensitive cryptographic operations, specifically AES encryption/decryption and the SHA-3 hash function, are instantiated in the HOSE.

5.1 PUF-Cash Overview

A high-level model of the PUF-Cash protocol is presented in Fig. 4 to highlight the basic operations. The sequence of numbered operations are described as follows:

(1) The bootstrap operation within the PUF-Cash protocol generates and distributes a set of anonymous POP cryptographic tuples ($\mathbf{POP_x}$) to customer devices, Alice, Bob, Charlie, etc. The set of $\mathbf{POP_x}$ are generated as customer long-lived keys using the TI's AT$_{DB}$, and are therefore anonymous to the TI.

(2) Alice contacts her FI and requests a withdrawal from her account, sending the tuple {*Amt*, ID$_A$, AID$_A$}, which includes both her non-anonymous ID for the FI and her anonymous AID for the TI. The FI checks her balance and responds with an ACK (not shown) if she has sufficient funds or a NAK if she does not.

(3) Assuming Alice has sufficient funds, the FI forwards the *Amt* and AID$_A$ to the TI to generate the eCash tokens ($\mathbf{eCt_A}$). The TI fetches Alice's anonymous $\mathbf{POP_A}$ from the POP$_{DB}$ using her AID$_A$, and adds a withdrawal record to its eCt$_{DB}$ database.

(4) The TI generates the $\mathbf{eCt_A}$ using a TRNG, and creates signed versions, $\mathbf{heCt_A}$, using Alice's $\mathbf{POP_A}$ in a XOR-secure-hash operation similar to Eq. 1. The TI then encrypts both of the $\mathbf{eCt_A}$ and $\mathbf{heCt_A}$ as $\mathbf{ECt_A}$ using a shared session key that is created between Alice and the

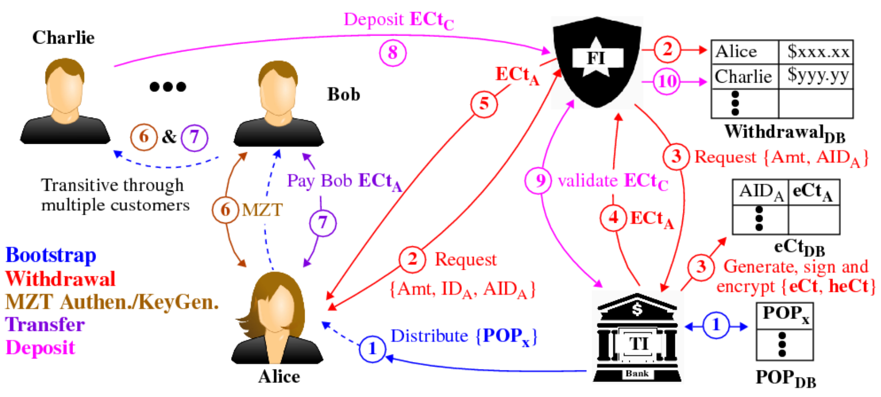


Fig. 4. High level overview of PUF-Cash

TI anonymously (not shown here but included in message exchange diagrams below), and transmits the $\mathbf{ECt_A}$ to the FI.

(5) The FI simply forwards the $\mathbf{ECt_A}$ to Alice. The FI, acting as an intermediary, keeps Alice and her $\mathbf{eCt_A}$ anonymous to the TI. Moreover, her $\mathbf{ECt_A}$ are also anonymous to the FI because they are encrypted with the Alice-TI session key.

(6) Alice contacts Bob for a payment transaction, and then both run the MZT protocol.

(7) Assuming authentication succeeds, Alice pays Bob with (a subset of) her $\mathbf{ECt_A}$ by encrypting them with the Alice-Bob MZT session key. Although not shown here, Alice and Bob use their $\mathbf{POP_x}$ to authenticate and propagate the provenance of the **eCt**. The MZT security functions and $\mathbf{ECt_X}$ transfer operation can be repeated between other pairs of customers, shown here between Bob and Charlie.

(8) At some point in the future, Charlie regains internet connectivity and deposits the $\mathbf{ECt_C}$ to his FI. Note that although Alice and Charlie use the same FI as shown, this is not a requirement.

(9) The FI contacts the TI and asks the TI to validate the $\mathbf{ECt_C}$, as a precursor to the FI accepting the $\mathbf{ECt_C}$ as a valid deposit.

(10) Assuming validation succeeds, the FI credits Charlie's account, and although not shown, the $\mathbf{ECt_B}$ are marked as 'redeemed' in the TI's eCt_{DB}.

5.2 Propagation of Provenance (POP) Qualities

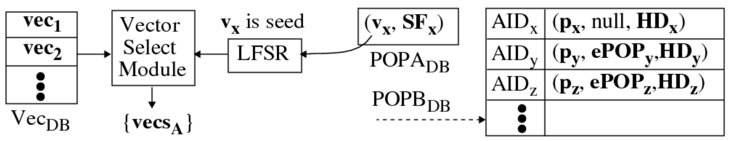


Fig. 5. Propagation-of-provenance (POP) database storage and challenge vector generation scheme.

The TI uses a specialized process to create the entire set of $\mathbf{POP_x}$ for each of the customers' POP_{DB} (distributed in Step 1 of Fig. 4) that significantly reduces the storage requirements for the challenges. The $\mathbf{POP_x}$ transmitted to Alice and Bob's device is composed of two components, a population-based component (PBC) and a device-specific component (DSC), each stored in two different databases labeled POPA_{DB} and POPB_{DB} in Fig. 5. The PBC component is used for all devices in Alice's POPB_{DB}, and is defined as the tuple ($\mathbf{v_x}$, $\mathbf{SF_x}$). The $\mathbf{v_x}$ component is a 32-bit

LFSR seed that selects the actual vectors $\mathbf{vecs_x}$ from a separate database called Vec_{DB} while $\mathbf{SF_x}$ represents a set of SpreadFactors for optimizing POP key generation (from Section 3).

The DSC components are stored in the POPB_{DB}, one element for each device, and are defined as a tuple (AID_x, $\mathbf{p_x}$, $\mathbf{ePOP_x}$, $\mathbf{HD_x}$). The AID_x is a 32-bit integer representing a device's anonymous ID, the $\mathbf{p_x}$ component is a 22-bit nonce used to specify the seeds to the LFSRs in the *DVDiffs* module from Section 3, the $\mathbf{ePOP_x}$ is the encrypted PUF's response to the challenge for device x (256-bits), and $\mathbf{HD_x}$ is the helper data (2048-bits). Therefore, the size of each POPB_{DB} element is 296 bytes, and is larger than the MZT_AT_{DB} which requires only 74 bytes per device entry.

The primary benefit of the larger POP database is related to the strength of the authentication process. The challenge information stored by Alice for, e.g., Bob's device, requires Bob's device to generate the response to Alice's stored challenge, and the challenge that Alice's stores (at least initially) has not been exposed a priori to Bob's device. This is true because the TI provides Alice with Bob's response to this challenge (as the $\mathbf{ePOP_B}$ component) by running a 'soft PUF' version of the SiRF PUF algorithm using the anonymous timing data stored in the AT_{DB}. The soft PUF is able to produce the same response that Bob's device generates for this challenge.

Moreover, Alice is able to refresh Bob's entry in her DB after every engagement with Bob in two different ways. In one scenario, Alice generates a new challenge for Bob's device by changing the $\mathbf{p_B}$ component (LFSR seeds) in POPB_{DB}, and then asks Bob to generate a new response. She then replaces the DSC component of Bob's entry in her POPB_{DB} with the new information. In the second scenario, Alice asks the TI to perform this operation. Although this requires Alice to be online, it prevents Bob from seeing Alice's next challenge for him.

5.3 PUF-Cash Protocol Details

The message exchange diagrams for the four PUF-Cash operations are described in this section. The BootStrap operation is carried out after device manufacture, and at any point in the field under the condition that Alice is online, i.e., has internet connectivity. Once Alice has engaged the TI in a bootstrap operation, she can perform any one of the three primary PUF-Cash operations, Withdrawal, Transfer and Deposit (note, her very first transaction must be a Withdrawal).

All transactions with TI are preceded with mutual authentication, e.g., MA_{NA}, and session key generation, e.g., SKG_{NA}. Session key generation produces a shared key $\mathbf{SK_x}$, where x is replaced with the initials of the authenticating parties, e.g., $\mathbf{SK_{TF}}$ refers to the shared session key between the TI and FI. Although the details of TB security functions are presented in previous work [22], the enrollment and regeneration functions for the POPB_{DB} are similar, and are outlined in Section 8.2 for completeness.

5.3.1 BootStrap. The message exchange diagram for Bootstrap is shown in Fig. 6. The sequence of operations performed by Alice and the TI are numbered within circles in the figure. The shaded regions identify operations carried out in the HOSE, i.e., the FPGA's programmable logic region.

(1) **Generate $\mathbf{LLK_{hose}}$**: Alice and the Token Issuer (TI) mutually authenticate and generate a shared session key, $\mathbf{SK_{TA}}$. Mutual authentication is done anonymously. Upon successful authentication, the TI affirms that Alice's device is a genuine SiRF-instantiated PUF device with anonymous ID, AID_A.

(2) The TI generates random values for $\mathbf{v_{hose}}$ and $\mathbf{p_{hose}}$ to enable the TI and Alice to enroll and regenerate, respectively, a long-lived key, $\mathbf{LLK_{hose}}$. The $\mathbf{LLK_{hose}}$ will be used by TI and Alice to encrypt and decrypt POP data. The term $\mathbf{v_{hose}}$ is used as the seed to pseudo-randomly select vectors, e.g. $\mathbf{vecs_x}$ from the Vec_{DB}, as shown in Fig. 5, and $\mathbf{p_{hose}}$ specifies the LFSR seed for the SiRF PUF *DVDiff* module. The anonymous timing data, $\mathbf{DV}$, for a random subset of devices corresponding to paths timed by $\mathbf{vecs_x}$, is extracted from the AT_{DB} and used to

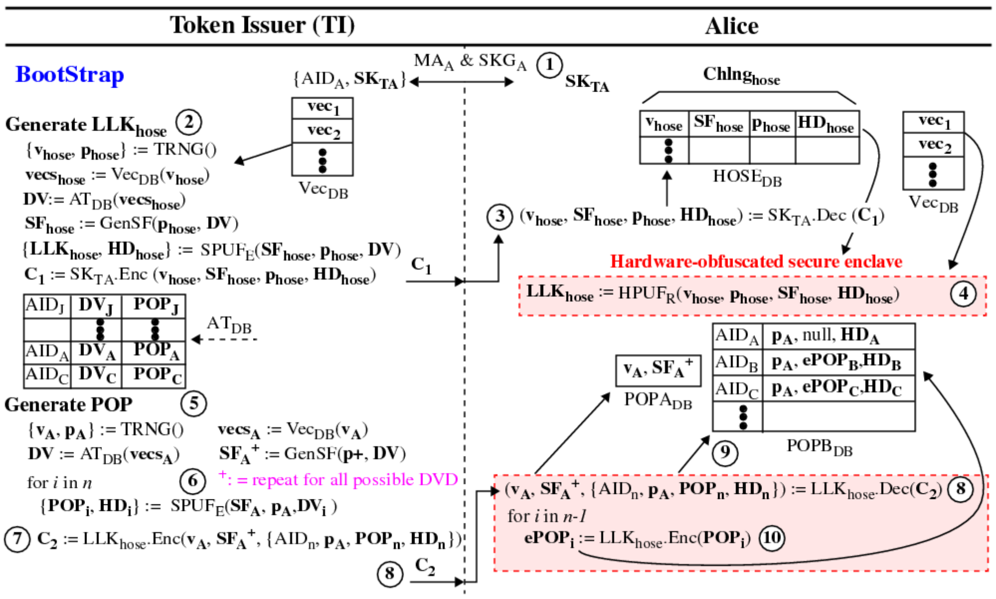

Token Issuer (TI)
Alice
BootStrap
MA_A & SKG_A ①
{AID_A, SK_TA}
SK_TA
Chlng_hose
Generate LLK_hose ②
{v_hose, p_hose} := TRNG()
vecs_hose := Vec_DB(v_hose)
DV := AT_DB(vecs_hose)
SF_hose := GenSF(p_hose, DV)
{LLK_hose, HD_hose} := SPUF_E(SF_hose, p_hose, DV)
C_1 := SK_TA.Enc (v_hose, SF_hose, p_hose, HD_hose)
vec_1
vec_2
Vec_DB
v_hose SF_hose p_hose HD_hose
HOSE_DB
③ (v_hose, SF_hose, p_hose, HD_hose) := SK_TA.Dec (C_1)
C_1
Hardware-obfuscated secure enclave
LLK_hose := HPUF_R(v_hose, p_hose, SF_hose, HD_hose) ④
AID_J DV_J POP_J
AID_A DV_A POP_A
AID_C DV_C POP_C
AT_DB
v_A, SF_A^+
POPA_DB
AID_A p_A, null, HD_A
AID_B p_A, ePOP_B, HD_B
AID_C p_A, ePOP_C, HD_C
POPB_DB
Generate POP ⑤
{v_A, p_A} := TRNG()
DV := AT_DB(vecs_A)
vecs_A := Vec_DB(v_A)
SF_A^+ := GenSF(p+, DV)
for i in n ⑥
+: = repeat for all possible DVD
{POP_i, HD_i} := SPUF_E(SF_A, p_A, DV_i)
⑦ C_2 := LLK_hose.Enc(v_A, SF_A^+, {AID_n, p_A, POP_n, HD_n})
⑧ C_2
⑨
(v_A, SF_A^+, {AID_n, p_A, POP_n, HD_n}) := LLK_hose.Dec(C_2) ⑧
for i in n-1
ePOP_i := LLK_hose.Enc(POP_i) ⑩

Fig. 6. PUF-Cash bootstrap operations between the Token Issuer and Alice.

MA_A:	Anonymous mutual authen.	SKG_A:	Session key generation	AID_x:	Anonymous ID
$\mathbf{SK_{TA}}$:	TI-Alice, etc. 256-bit session keys	$\mathbf{v_x}$:	32-bit vector specifier	$\mathbf{vecs_x}$:	771-bit Chlng vecs
$\mathbf{p_x}$:	22-bit LFSR seed	$\mathbf{SF_x}$:	2048 bytes of SpreadFactors	$\mathbf{DV}$:	Set of 16-bit DV
$\mathbf{LLK_{hose}}$:	HOSE 256-bit LLK	$\mathbf{POP_x}$:	POP **LLK**	$\mathbf{ePOP_x}$:	HOSE encrypted POP **LLK**
$\mathbf{HD_x}$:	2048-bit Helper data	C_x:	Cipher text packet		

generate a set of SpreadFactors, $\mathbf{SF_{hose}}$. The TI runs a software version of the SiRF PUF in enrollment mode, $SPUF_E$, which produces the $\mathbf{LLK_{hose}}$ key and the $\mathbf{HD_{hose}}$ helper data.

(3) The TI encrypts the tuple ($\mathbf{v_{hose}}$, $\mathbf{SF_{hose}}$, $\mathbf{p_{hose}}$, $\mathbf{HD_{hose}}$) with its session key $\mathbf{SK_{TA}}$ as $\mathbf{C_1}$ and transmits it to Alice. Alice decrypts $\mathbf{C_1}$ and inserts the challenge data into her $HOSE_{DB}$.

(4) Alice reads the challenge information from her $HOSE_{DB}$ and runs her hardware PUF in regeneration mode, $HPUF_R$, to produce $\mathbf{LLK_{hose}}$.

(5) **Generate POP**: The TI generates a set of POP elements for a set of devices from the population by first generating random values for $\mathbf{v_A}$ and $\mathbf{p_A}$ and extracting a set of vectors $\mathbf{vecs_A}$. It then extracts the timing values, **DV**, for a random subset of devices from AT_{DB} corresponding to $\mathbf{vecs_A}$. *GenSF* is called to generate the SpreadFactors, $\mathbf{SF_A^+}$, for all possible DVD, that can be created from the selected 2048 DV_R and 2048 DV_F. From the discussion in Section 3.2, the DV_R and DV_F can be paired in 2^{22} possible ways to create unique DVD. Therefore, the size of $\mathbf{SF_A^+}$ is 4 million bytes. This enables Alice to refresh her POP challenge and response information after a successful transfer operation with any device in her POP_{DB}.

(6) The TI generates a set of 256-bit $\mathbf{POP_i}$ responses to the challenge for a set of devices n, using the $\mathbf{DV_i}$ corresponding to each of the devices i. Alice's response, $\mathbf{POP_A}$, is stored in the verifier's AT_{DB} for signing and validating **eCt** during withdrawals and deposits, respectively.

(7) The challenge components, $\mathbf{v_A}$ and $\mathbf{SF_A^+}$, and the tuple sets {AID_n, $\mathbf{p_A}$, $\mathbf{POP_n}$, $\mathbf{HD_n}$} are encrypted with Alice's $\mathbf{LLK_{hose}}$ by the TI. The $\mathbf{POP_A}$ component in Alice's tuple is set to null.

(8) The TI sends the encrypted packet C_2 to Alice, which she decrypts within her HOSE. She stores the $\mathbf{v_A}$ and $\mathbf{SF_A^+}$ in her $POPA_{DB}$. These components of the challenge are used for all customers she interacts with.

(9) She then creates separate records for each of the customers i in the POPB$_{DB}$ database and stores the AID$_i$ and $\mathbf{HD_i}$ components. The $\mathbf{p_A}$ component is replicated in each database record and will be used as the initial value during the first eCash transaction with a customer.

(10) The $\mathbf{POP_i}$ are re-encrypted with Alice's $\mathbf{LLK_{hose}}$ and stored as $\mathbf{ePOP_i}$. This protects the customer responses in the event an adversary gains access to Alice's device and attempts to extract information from the POPB$_{DB}$. Note that Alice's $\mathbf{ePOP_i}$ does not need to be stored (is null) because she can regenerate it with her PUF.

The size of the subset of **DV** extracted from the AT$_{DB}$ and used as input to *GenSF* in Step 5 needs to be large enough to characterize the entire population. In our implementation, we use sets of **DV** corresponding to 30 customers. The subset of the customer population for which the TI creates $\mathbf{POP_i}$ for Alice in Step 6 can be selected by Alice's device from preferences she specifies in advance, or they can be learned over time from online eCash transactions she engages in.

5.3.2 *Withdrawal.* The following series of message exchanges and cryptographic operations are carried out between Alice, the FI and TI to enable Alice to obtain a set of anonymous **eCt** for payment transactions with end-users or commercial vendors. The corresponding message exchange diagram is shown in Fig. 7.

(1) The TI and FI mutually authenticate and generate a session key, $\mathbf{SK_{TF}}$ using the non-anonymous TB protocol primitives.
(2) The FI and Alice mutually authenticate and generate, $\mathbf{SK_{FA}}$, using the MZT protocol primitives.
(3) Alice creates a ciphertext packet, $\mathbf{C_1}$, containing her anonymous ID called AID$_A$, and requested withdrawal amount, *Amt*, and sends the packet to FI.
(4) FI decrypts $\mathbf{C_1}$ with $\mathbf{SK_{FA}}$, checks Alice's balance and responds to Alice with a ACK if she has enough funds in her account, or a NAK if she does not.
(5) If the response is ACK, the transaction continues (otherwise it is cancelled) with FI encrypting AID$_A$ and *Amt* as $\mathbf{C_2}$ and transmitting the packet to TI. TI decrypts $\mathbf{C_2}$.
(6) TI and Alice generate a session key $\mathbf{SK_{TA}}$ using procedure SKG$_A$ (which utilizes the anonymous timing database). The FI serves as a pass-through intermediary. The SKG$_A$ process is nearly identical to the **Generate** $\mathbf{LLK_{hose}}$ (Step 2 of BootStrap) where the TI generates a challenge, runs $\mathbf{SK_{TA}}$:= SPUF$_E$ and sends the challenge and helper data encrypted to FI (not shown). The FI forwards the packet to Alice, and Alice decrypts and regenerates $\mathbf{SK_{TA}}$ by running her hardware PUF in regeneration mode. Note that Alice remains anonymous to TI because the TI draws the challenge and DV from AT$_{DB}$ using her AID$_A$.
(7) The TI extracts Alice's $\mathbf{POP_A}$ from the AT$_{DB}$ using her AID$_A$.
(8) The TI generates a set of $\mathbf{eCt_A}$ using its *GenNonce* function, equivalent to one 256-bit nonce for each 1 cent token of Alice's requested *Amt*, and records the $\mathbf{eCt_A}$ in its eCt$_{DB}$ along with her AID$_A$ and $\mathbf{POP_A}$. The latter two components can be used for recovering lost funds and tracking malicious actors, but are otherwise not needed.
(9) The TI creates signed versions of the $\mathbf{eCt_A}$, labeled as $\mathbf{heCt_A}$, by XORing each $\mathbf{eCt_A}$ with $\mathbf{POP_A}$, and then hashing the result. The $\mathbf{eCt_A}$ and $\mathbf{heCt_A}$ are then encrypted with $\mathbf{POP_A}$ to prevent observation or manipulation attacks from the processor-side. The encrypted versions, $\mathbf{ECt_A}$, are encrypted again with the TI-Alice session key, $\mathbf{SK_{TA}}$, as $\mathbf{C_3}$. The double encryption adds an additional layer of obscurity to the FI and to network packet eavesdropping.
(10) The TI transfers $\mathbf{C_3}$ to the FI. In order to ensure Alice acknowledges receipt of the $\mathbf{ECt_A}$ packet, the FI generates a nonce, $\mathbf{n_{obs}}$, and XOR encrypts $\mathbf{C_3}$ as $\mathbf{C_{3_o}}$ and transmits it to Alice. Once Alice acknowledges receipt, the FI sends $\mathbf{n_{obs}}$ and deducts *Amt* from her balance in Acct$_{DB}$. Alice recovers $\mathbf{C_3}$ by XORing with $\mathbf{n_{obs}}$, decrypts $\mathbf{C_3}$ to recover the $\mathbf{ECt_A}$ and passes them to the HOSE for processing.

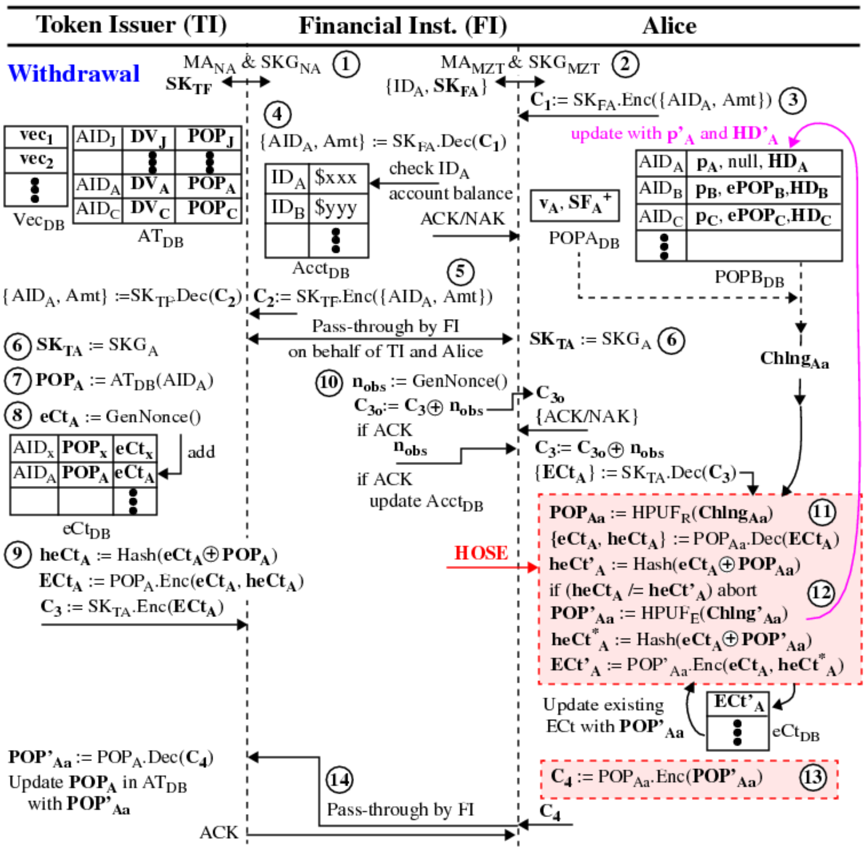

Token Issuer (TI)
Financial Inst. (FI)
Alice
Withdrawal
MA_NA & SKG_NA
SK_TF
MA_MZT & SKG_MZT
{ID_A, SK_FA}
C_1 := SK_FA.Enc({AID_A, Amt})
{AID_A, Amt} := SK_FA.Dec(C_1)
check ID_A account balance
ACK/NAK
update with p'_A and HD'_A
Vec_DB
AT_DB
Acct_DB
POPA_DB
POPB_DB
{AID_A, Amt} := SK_TF.Dec(C_2)
C_2 := SK_TF.Enc({AID_A, Amt})
Pass-through by FI on behalf of TI and Alice
SK_TA := SKG_A
POP_A := AT_DB(AID_A)
eCt_A := GenNonce()
add
eCt_DB
heCt_A := Hash(eCt_A ⊕ POP_A)
ECt_A := POP_A.Enc(eCt_A, heCt_A)
C_3 := SK_TA.Enc(ECt_A)
n_obs := GenNonce()
C_3o := C_3 ⊕ n_obs
if ACK
if ACK update Acct_DB
Chlng_Aa
C_3o {ACK/NAK}
C_3 := C_3o ⊕ n_obs
{ECt_A} := SK_TA.Dec(C_3)
HOSE
POP_Aa := HPUF_R(Chlng_Aa)
{eCt_A, heCt_A} := POP_Aa.Dec(ECt_A)
heCt'_A := Hash(eCt_A ⊕ POP_Aa)
if (heCt_A /= heCt'_A) abort
POP'_Aa := HPUF_E(Chlng'_Aa)
heCt*_A := Hash(eCt_A ⊕ POP'_Aa)
ECt'_A := POP'_Aa.Enc(eCt_A, heCt*_A)
Update existing ECt with POP'_Aa
C_4 := POP_Aa.Enc(POP'_Aa)
POP'_Aa := POP_A.Dec(C_4)
Update POP_A in AT_DB with POP'_Aa
Pass-through by FI
ACK

Fig. 7. PUF-Cash message exchange for withdrawal operation between Alice, the Financial Institution and the Token Issuer.

(11) Alice's HOSE fetches challenge information from the POP DBs as $\mathbf{Chlng_{Aa}}$, and runs Alice's hardware PUF in regeneration mode, HPUF_R, to reproduce $\mathbf{POP_{Aa}}$. The subscript **Aa** should be read as "Challenge for Alice from Alice". The $\mathbf{ECt_A}$ are then decrypted and validated by the HOSE. Validation is accomplished by recreating the signatures, $\mathbf{heCt'_A}$, using the $\mathbf{eCt_A}$ and the locally generated $\mathbf{POP_{Aa}}$, and comparing them with the received versions, $\mathbf{heCt_A}$.

(12) If any of the signature comparisons fail, then the process is aborted. Otherwise, Alice updates her challenge to $\mathbf{Chlng'_{Aa}}$ by incrementing the $\mathbf{p_A}$ component, and then running HPUF_E to generate $\mathbf{POP'_{Aa}}$. She updates the POPB_{DB} with $\mathbf{p'_A}$ and $\mathbf{HD'_A}$, and creates new $\mathbf{heCt^*_A}$ for the $\mathbf{eCt_A}$. She encrypts the $\mathbf{eCt_A}$ and $\mathbf{heCt^*_A}$ and adds the new $\mathbf{ECt'_A}$ to her eCt_{DB}. She also re-encrypts any existing elements with the new $\mathbf{POP'_{Aa}}$. This type of key rolling scheme reduces the likelihood of a successful differential power analysis attack on Alice's device.

(13) If the previous step succeeds, Alice encrypts the new $\mathbf{POP'_{Aa}}$ with the original $\mathbf{POP_{Aa}}$ as $\mathbf{C_4}$.

(14) Alice sends $\mathbf{C_4}$ to the FI, who forwards to TI. The TI decrypts $\mathbf{C_4}$ with $\mathbf{POP_A}$ to recover $\mathbf{POP'_{Aa}}$. The TI updates Alice's POP in its AT_{DB} database with the new version and sends an ACK through FI to Alice. Although not shown, if Alice does not receive the ACK, she retains the previous parameters $\mathbf{p_A}$ and $\mathbf{HD_A}$ for recovery actions during the next withdrawal operation.

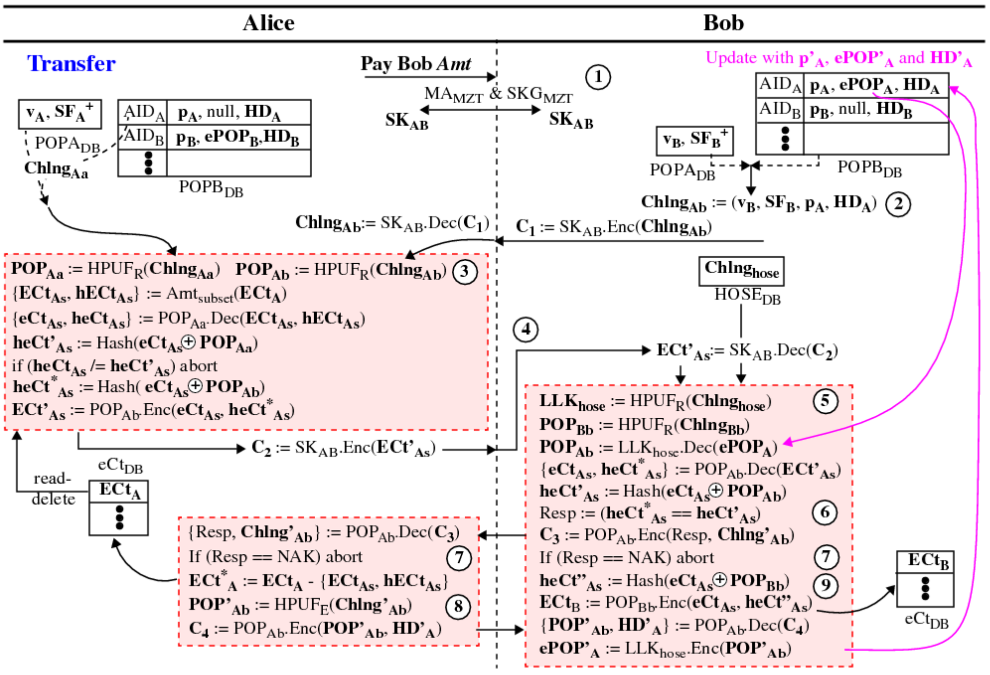

Alice
Bob
Transfer
Pay Bob Amt
MA_MZT & SKG_MZT
SK_AB
SK_AB
Update with p'_A, ePOP'_A and HD'_A
Chlng_Ab := (v_B, SF_B, p_A, HD_A)
Chlng_Ab := SK_AB.Dec(C_1)
C_1 := SK_AB.Enc(Chlng_Ab)
POP_Aa := HPUF_R(Chlng_Aa) POP_Ab := HPUF_R(Chlng_Ab)
{ECt_As, hECt_As} := Amt_subset(ECt_A)
{eCt_As, heCt_As} := POP_Aa.Dec(ECt_As, hECt_As)
heCt'_As := Hash(eCt_As ⊕ POP_Aa)
if (heCt_As /= heCt'_As) abort
heCt*_As := Hash(eCt_As ⊕ POP_Ab)
ECt'_As := POP_Ab.Enc(eCt_As, heCt*_As)
C_2 := SK_AB.Enc(ECt'_As)
ECt'_As := SK_AB.Dec(C_2)
LLK_hose := HPUF_R(Chlng_hose)
POP_Bb := HPUF_R(Chlng_Bb)
POP_Ab := LLK_hose.Dec(ePOP_A)
{eCt_As, heCt*_As} := POP_Ab.Dec(ECt'_As)
heCt'_As := Hash(eCt_As ⊕ POP_Ab)
Resp := (heCt*_As == heCt'_As)
C_3 := POP_Ab.Enc(Resp, Chlng'_Ab)
If (Resp == NAK) abort
heCt''_As := Hash(eCt_As ⊕ POP_Bb)
ECt_B := POP_Bb.Enc(eCt_As, heCt''_As)
{POP'_Ab, HD'_A} := POP_Ab.Dec(C_4)
ePOP'_A := LLK_hose.Enc(POP'_Ab)
{Resp, Chlng'_Ab} := POP_Ab.Dec(C_3)
If (Resp == NAK) abort
ECt*_A := ECt_A - {ECt_As, hECt_As}
POP'_Ab := HPUF_E(Chlng'_Ab)
C_4 := POP_Ab.Enc(POP'_Ab, HD'_A)
v_A, SF_A^+
POPA_DB
Chlng_Aa
AID_A p_A, null, HD_A
AID_B p_B, ePOP_B, HD_B
POPB_DB
v_B, SF_B^+
AID_A p_A, ePOP_A, HD_A
AID_B p_B, null, HD_B
Chlng_hose
HOSE_DB
read-delete
eCt_DB
ECt_A
ECt_B
eCt_DB
1
2
3
4
5
6
7
7
8
9

Fig. 8. PUF-Cash transfer operation between Alice and Bob.

5.3.3 Transfer. The PUF-Cash protocol supports the transfer of **eCt** between offline entities, as illustrated by the message exchange diagram for two devices labeled Alice and Bob in Fig. 8. The transfer protocol is carried out exclusively between Alice and Bob, and is transitive i.e., Bob can pay Charlie, etc.

(1) Alice sends Bob a request to transfer *Amt* to Bob. Bob accepts the transfer request. Alice and Bob mutually authenticate and generate a session key, $\mathbf{SK_{AB}}$, using MZT protocol.
(2) Bob extracts a challenge from his POP DBs, with subscript **Ab** indicating "Challenge for Alice from Bob". Bob encrypts the challenge $\mathbf{Chlng}_{Ab}$ with his session key $\mathbf{SK_{AB}}$ as C_1 and transmits it to Alice.
(3) Alice decrypts $\mathbf{Chlng}_{Ab}$ and passes it into her HOSE. Alice's HOSE fetches $\mathbf{Chlng}_{Aa}$ and regenerates both $\mathbf{POP_{Aa}}$ and $\mathbf{POP_{Ab}}$ by running HPUF_R. Alice then fetches a subset of her encrypted $\mathbf{ECt_A}$ from her eCt_{DB} and decrypts it with her regenerated $\mathbf{POP_{Aa}}$ to recover the tuple $\{\mathbf{eCt_{As}}, \mathbf{heCt_{As}}\}$. Following that, she validates the database stored $\mathbf{eCt_{As}}$ by creating $\mathbf{heCt'_{As}}$ with $\mathbf{POP_{Aa}}$ and compares them with the $\mathbf{heCt_{As}}$ extracted from the database. The process is aborted if a mismatch is detected. If not, the $\mathbf{eCt_{As}}$ subset is signed with $\mathbf{POP_{Ab}}$ as $\mathbf{heCt^*_{As}}$, and the tuple, ECt'_{As}, is encrypted with AES using $\mathbf{POP_{Ab}}$ as the secret key. The re-signing of Alice's **eCt** with a key that Bob stores and only Alice can generate is a key feature of the scheme.
(4) Alice encrypts the $\mathbf{ECt'_{As}}$ with $\mathbf{SK_{AB}}$ as C_2 and transmits it to Bob, who decrypts and transfers it to his HOSE.
(5) Bob regenerates his $\mathbf{LLK_{hose}}$ and $\mathbf{POP_{Bb}}$ using challenges from the HOSE_{DB} and POP DBs, respectively. He uses his $\mathbf{LLK_{hose}}$ to decrypt Alice's $\mathbf{ePOP_A}$ as $\mathbf{POP_{Ab}}$, and then uses $\mathbf{POP_{Ab}}$ to decrypt $\mathbf{ECt'_{As}}$ to recover Alice's $\mathbf{eCt_{As}}$ and $\mathbf{heCt^*_{As}}$. He validates them by creating $\mathbf{heCt'_{As}}$ and comparing with the $\mathbf{heCt^*_{As}}$ received from Alice.

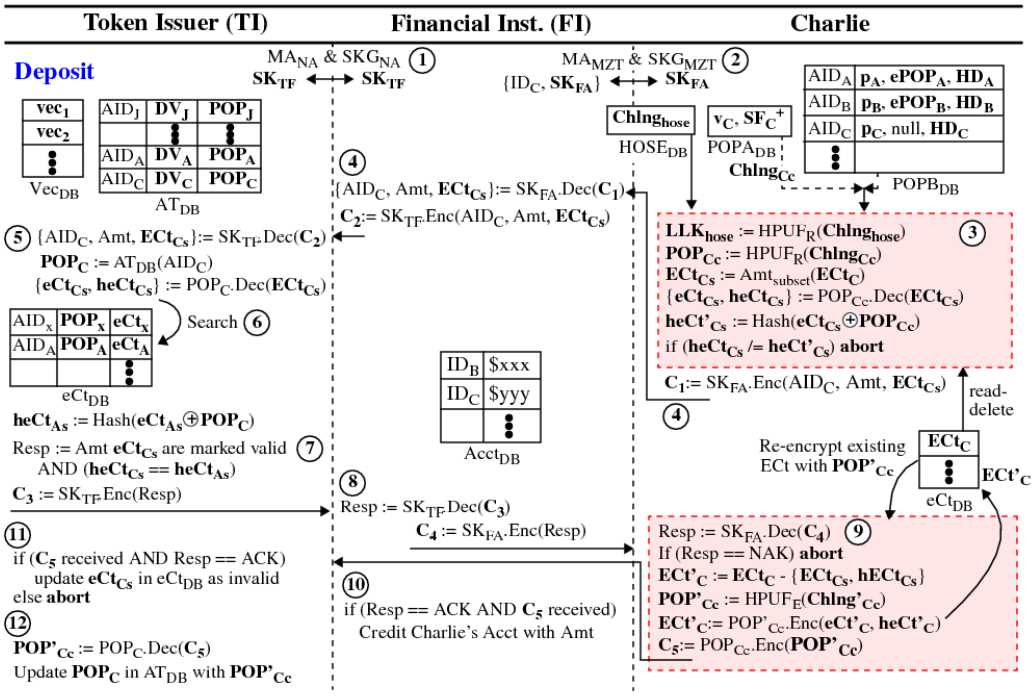


Fig. 9. PUF-Cash deposit operation between Charlie, the Financial Institution and the Token Issuer.

(6) Bob records the result of the validation process as Resp := ACK or NAK. Bob also updates Alice's challenge by randomly modifying the $\mathbf{p_A}$ parameter in his POPB$_{DB}$ to produce a new challenge, $\mathbf{Chlng'_{Ab}}$. He encrypts the Resp and the new challenge with $\mathbf{POP_{Ab}}$ as C_3 and transmits it to Alice.

(7) Alice transfers C_3 into her HOSE, decrypts and aborts if the Resp is NAK. Similarly, Bob aborts if Resp is NAK. Otherwise, Alice removes the spent tuples $\{\mathbf{ECt_{As}}, \mathbf{hECt_{As}}\}$ from $\mathbf{ECt_A}$ and stores the updated $\mathbf{ECt^*_A}$ back to her eCt$_{DB}$.

(8) Alice generates a new response, $\mathbf{POP'_{Ab}}$, using the new challenge, $\mathbf{Chlng'_{Ab}}$, encrypts it along with the new helper data $\mathbf{HD'_A}$ using the original $\mathbf{POP_{Ab}}$ as C_4 and transmits it to Bob.

(9) Bob hashes Alice's $\mathbf{eCt_{As}}$ as $\mathbf{heCt''_{As}}$ and encrypts them with the $\mathbf{eCt_{As}}$ using his $\mathbf{POP_{Bb}}$ as $\mathbf{ECt_B}$, which he stores to his eCt$_{DB}$. He then decrypts C_4 to recover Alice's $\mathbf{POP'_{Ab}}$ and $\mathbf{HD'_A}$. Bob encrypts $\mathbf{POP'_{Ab}}$ with his $\mathbf{LLK_{hose}}$ as $\mathbf{ePOP'_A}$, and updates his POPB$_{DB}$ with the new challenge ($\mathbf{p'_A}$), encrypted response $\mathbf{ePOP'_A}$ and helper data $\mathbf{HD'_A}$.

5.3.4 *Deposit.* The deposit operation involves the TI, FI and an end-user device, e.g., Charlie, where anonymity is preserved between the TI and Charlie. The message exchange sequence is shown in Fig. 9, and is described in the following numbered sequence.

(1) The TI and FI mutually authenticate and generate a session key, $\mathbf{SK_{TF}}$, using the TB protocol.
(2) The FI and Alice mutually authenticate and generate a session key, $\mathbf{SK_{FA}}$, using MZT protocol.
(3) Charlie fetches challenge information and runs his hardware PUF to regenerate $\mathbf{LLK_{hose}}$ and $\mathbf{POP_{Cc}}$. Bob selects a subset of his $\mathbf{ECt_C}$ and then decrypts them using his $\mathbf{POP_{Cc}}$ as $\mathbf{eCt_{Cs}}$ and $\mathbf{heCt_{Cs}}$. Charlie then validates his $\mathbf{eCt}$ and aborts if validation fails.

(4) Charlie constructs a packet consisting of his AID_C, *Amt* and $\mathbf{ECt_{Cs}}$. The packet is encrypted with $\mathbf{SK_{FA}}$ as $\mathbf{C_1}$, and transmitted to the FI. The FI recovers the elements in $\mathbf{C_1}$, re-encrypts them with $\mathbf{SK_{TF}}$ as $\mathbf{C_2}$ and transmits them to the TI.
(5) The TI decrypts $\mathbf{C_2}$ to recover AID_C, *Amt*, and the corresponding tokens $\mathbf{ECt_{Cs}}$. It then reads $\mathbf{POP_C}$ from AT_{DB} using AID_C, and decrypts the $\mathbf{ECt_{Cs}}$ using Charlie's $\mathbf{POP_C}$.
(6) The TI searches for each of the $\mathbf{eCt_{Cs}}$ in the master eCt_{DB} and checks if they are marked *valid*, i.e., still in circulation. If all $\mathbf{eCt_{Cs}}$ are valid, then signatures are created using Charlie's $\mathbf{POP_C}$ as $\mathbf{heCt_{As}}$. For convenience, it is assumed that some of the tokens in Charlie's possession were originally withdrawn by Alice ($\mathbf{eCt_A}$), to show the complete cycle for any given $\mathbf{eCt_A}$.
(7) The TI assigns Resp := ACK if all (*Amt*) of the **eCt** and **heCt** validation processes succeed, and NAK otherwise. The TI encrypts Resp as $\mathbf{C_3}$ with $\mathbf{SK_{TF}}$ and transmits it to the FI.
(8) FI decrypts $\mathbf{C_3}$, records Resp, re-encrypts Resp with $\mathbf{SK_{FA}}$ as $\mathbf{C_4}$, and transmits it to Charlie.
(9) Charlie's HOSE decrypts $\mathbf{C_4}$ and aborts if Resp is a NAK. Otherwise, he deducts the deposited subset of **eCt** and **heCt** from his database and generates a new challenge and corresponding $\mathbf{POP'}_{Cc}$. He encrypts and stores the new $\mathbf{ECt'_C}$ element to his eCt_{DB}, and re-encrypts other elements in his database with the new $\mathbf{POP'}_{Cc}$ if any exist. He encrypts $\mathbf{POP'}_{Cc}$ as $\mathbf{C_5}$ using the original $\mathbf{POP}_{Cc}$ and transmits it to FI.
(10) Once FI receives $\mathbf{C_5}$, if Resp == ACK, it credits Charlie's account with *Amt*.
(11) Once TI receives $\mathbf{C_5}$ and Resp == ACK, it marks all of Charlie's **eCt** as invalid (redeemed) in the eCt_{DB}. Otherwise it aborts.
(12) If Resp != NAK, the TI decrypts $\mathbf{C_5}$ and updates Charlie's POP in its AT_{DB}.

6 EXPERIMENT SETUP

We first present the hardware and software overheads associated with the implementation of the PUF-Cash protocol on our test bed and then present a run time analysis.

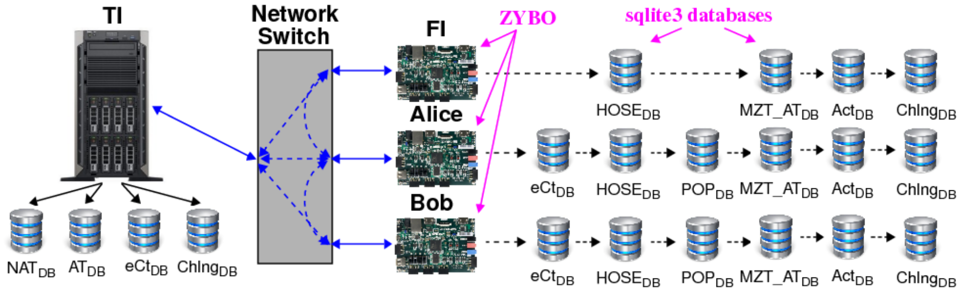


Fig. 10. Experiment set for the evaluation of PUF-Cash.

6.1 Device Resource Utilization

The test bed consists of one server and three devices as shown in Fig. 10. The token issuer (TI) is implemented on a Dell PowerEdge T440 Server with 32 1.8 GHz processors and 128 GB of main memory. The financial institution (FI), Alice and Bob run on a set of Digilent ZYBO-Z7 boards, which utilize Xilinx Zynq 7010 SoCs [8]. The processor system (PS) incorporates a 667 MHz dual-core Cortex-A9 processor, which has access to 1 GB of DRAM and a 1 Gbit/sec ethernet port. The programmable logic side (PL) has 17,600 LUTs and 240 KB of PL-side block RAM (BRAM).

The TI and FI applications are implemented as multi-threaded C programs, while customer device applications are single-threaded. The FI and device implementations are co-design applications, where network and database functions are implemented in C on the PS, while encryption, secure hash, SiRF PUF authentication bitstring and key generation functions are implemented as state machines in the PL. The TI emulates the SiRF PUF in software. The server and devices are connected to a network switch with 1 Gbit/sec of available bandwidth.

Table 1. PL-side resource utilization on the Zynq 7010.

		SiRF			**Crypto**			
Resource	**Engine**	**Netlist**	**Total**	**AES**	**SHA-3**	**Total**	**Available**	**Overall**
LUT	5842	796	37.72%	3128	3809	39.41%	17,600	77.13%
LUTRAM	60	96	2.60%	-	-		6000	2.60%
FF	4377	32	12.53%	2992	2244	14.88%	35,200	27.40%
BRAM blocks	5 (20 KB)	-	8.33%	-	-		60 (240 KB)	8.33%
DSP	2	-	2.50%	-	-		80	2.50%
BUFG	2	-	6.25%	-	-		32	6.25%

The PL resources used by the SiRF PUF Engine and SiRF PUF netlist (Entropy source) are given in Table 1. The percentage of resources used by all SiRF PUF components is shown in the 4th column. The overheads of other components of the HOSE, i.e., AES and SHA-3 cryptographic functions, are also presented in columns 5 and 6. The LUT row shows that SiRF PUF utilization is 37.72%, while utilization of the cryptographic functions is slightly larger at 39.41%.

6.2 Database Size Overhead

The TI, FI and field-device applications utilize the sqlite3 database management software [11]. The database size overheads for each of the databases shown in Fig. 10 are given in Table 2, with CD used as an abbreviation for the customer device. The TI stores SiRF provisioning data in the NAT$_{DB}$ and AT$_{DB}$ for 140 ZYBO devices, each with 5072 DV$_R$ and 5072 DV$_F$ records (Rec), which supports a challenge-response space of more than 2^{24} bits per device (Note that the entire CRP space is 2^{35} bits per device using the SiRF netlist configuration shown in Fig. 1, which is discussed further in Section 8.1). Although each DV can be represented as a 16-bit fixed point number with 4 digits of precision, indexing and other book-keeping within the database engine increase the size to 60 bytes per DV record. Therefore, with 10,144 DV per device, the total storage is 594.3 KB per device. The right-most column shows the total size of these databases after populating them with provisioning data from 140 FPGAs.

The Challenge$_{DB}$, utilized by all entities in the PUF-Cash system, is 1 MB in size. Individual record sizes within the MZT_AT$_{DB}$ is given as 80 bytes per entry. From Fig. 2, a record consists of a 32-byte **ZHK** and **nonce**, and two 4-byte integers, leaving 8 bytes for database overhead. Assuming each device stores a record for 140 other devices in the population, the overhead is 10.9 KB.

The POP$_{DB}$ is composed of two component tables. The POPA$_{DB}$ stores 4 MB of **SF**, which is used for all customers, and is therefore fixed in size. The POPB$_{DB}$ record size is 296 bytes, leading to a size overhead of 40.5 KB assuming each device stores data for 140 other devices. Combined with the fixed overhead of the POPA$_{DB}$, the storage per device is 4.04 MB. The size of the challenges stored in the MZT and HOSE DBs is small at 4.6 KB per device. The total overhead for a CD is the sum of the last four rows, and is slightly larger than 5 MB.

7 EXPERIMENTAL RESULTS

Hardware experiments are used measure the performance of the PUF-Cash protocol operations. In each experiment, the timing information is collected as the protocol executes the withdraw-transfer-deposit sequence using an increasing number of **eCts** of sizes 1, 5, 10, 50, 100, up to 500,000, for a total of 12 sequences. In each sequence, the entire set of tokens are processed through all operations,

Table 2. Database Size Overhead.

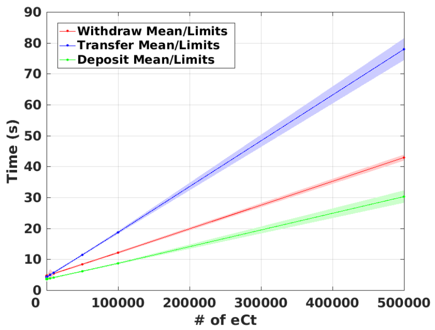
DB name	Used By	Base (MB)	Rec (B)	Per Device (KB)	All Devices (MB)
NAT$_{DB}$	TI	0	60	594.3	81.2
AT$_{DB}$	TI	0	60	594.3	81.2
Chlng$_{DB}$	TI/FI/CD	1	-	-	1
MZT_AT$_{DB}$	FI/CD	0	80	10.9	0.011
POP$_{DB}$	FI/CD	4	296	40.5	4.040
HOSE/MZT$_{DB}$	FI/CD	0	2372	4.6	0.005

thereby tokens are created, transacted and deleted, leaving the database in an empty state at the beginning of the next sequence. In addition, the POP **LLKs** generated in each experiment are stored to a file to allow an assessment of their statistical quality. The experiment as described is performed 40 times for statistical significance and to determine the limits on the transfer times.

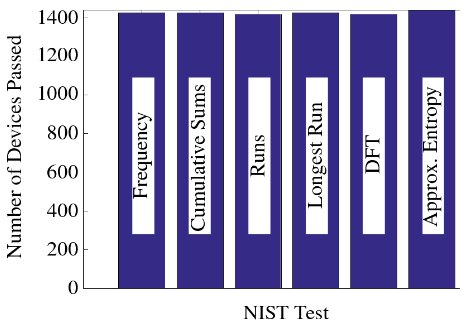
7.1 Run Time Analysis

The processing and transfer times associated with the withdrawal, transfer and deposit operations within the PUF-Cash protocol are plotted in Fig. 11a. The processing times, and 3σ limits, are plotted as a function of the number of **eCt** processed. The run times include the time taken to complete all operations specified in the message exchange diagrams, including the network transfer times between devices.

The processing time of all three operations is linear with respect to the number of tokens. The transfer operation, which occurs exclusively between Alice and Bob's devices, possesses the largest processing time overhead. This is due largely to the limited processing capability of the devices. The processing times associated with the smaller value transfer operations, e.g., from 1 cent to 10,000 ($100), are upper bounded at approximately 6 seconds. Processing times for amounts larger than 10,000 begin to diverge, but remain less than 20 seconds up to 100,000 **eCt** ($1000). These processing times are competitive with existing state of the art payment systems where settlement occurs at the end of the transaction.



(a)



(b)

Fig. 11. (a) Run times (y-axis) plotted against the number of **eCt** transferred (x-axis) for Withdrawal, Transfer and Deposit operations, as measured on the devices. The number of **eCt** processed varies from 1 (1 cent) to 500,000 ($5,000). Three-$\sigma$ variations in the measurements are shown as shaded regions around the mean values. (b) NIST test results using 1,400 POP LLKs collected during execution of the protocol.

Table 3 gives mean run times and 3σ limits for the timing-based (TB) authentication and key generation operations. The mean time for each operation is less than 1 second for all operations

Table 3. SiRF PUF Primitive Run Time Analysis. Mean and $\pm 3\sigma$ values are given. All times in seconds (s).

DA	VA	SE	LLK Enroll	LLK Regen
0.764 ± 0.064	0.618 ± 0.057	0.715 ± 0.071	1.400 ± 0.025	0.919 ± 0.025

Table 4. PUF-Cash Primitive Run Time Analysis. Average transfer times per 100 elements plus fixed overhead time. All times in seconds (s).

BootStrap (Total for 100 POP LLKs)	MZT Enroll (Total for 100 MZT ATs)
5.9 per 100 + 3.5	0.025 per 100 + 3.0

except LLK Enroll. Table 4 gives the run time for the BootStrap operation carried out at device startup (see Fig. 6). The value given corresponds to the total transfer time for 100 $\mathbf{POP_x}$ elements, which includes a fixed overhead of 3.5 seconds. The transfer times correspond to approximately 17 $\mathbf{POP_x}$ per second. MZT enrollment time (also performed during BootStrap) translates to approximately 4000 MZT authentication tokens per second, with a fixed overhead of 3.0 seconds. A typical sequence performed at startup includes DA, VA, SE, two LLK regeneration operations and the BootStrap and MZT Enroll operations. The authentication and key generation operations (DA, VA, SE and LLK Regen) take approximately 4 seconds. For 100 POP **LLKs** and MZT **ATs**, the total startup time is 4 + 9.4 + 3.025 = 16.36 seconds.

7.2 POP LLK Statistical Analysis

In this section, we assess the statistical quality of the **LLKs** generated during runs of the PUF-Cash protocol. The MZT protocol utilizes only one **LLK** in the protocol, and therefore nearly all of the **LLK** analyzed are produced by the POP protocol. The message exchange diagrams for withdrawal, transfer and deposit shown in Figs. 7, 8 and 9 incorporate **LLK** refresh operations, which are annotated as $\mathbf{POP'}_{xy}$. In each withdraw-transfer-deposit sequence, Alice generates a new **LLK** for herself during withdrawals and a second new **LLK** on behalf of Bob during transfers, and Bob generates a new **LLK** during deposits. Therefore, during the execution of the 12 sequences in each experiment, 36 **LLKs** are generated. The 40 repeated runs of the experiment yields a total of 1440 256-bit **LLKs**.

The 1440 **LLKs** are used as input to the NIST statistical test suite. Given the limited size of the bitstrings (256 bits), only six of the NIST statistical tests are applicable, as shown in Fig. 11b. The minimum pass threshold for a set size of 1440 bitstrings, and with α set to the default value 0.01, is 1414. All six tests passed, with the Runs test representing the worst case, at 1418 passing bitstrings.

Inter-bitstring Hamming distance (Inter-HD) is commonly used to measure uniqueness among the bitstrings. Inter-HD is computed by pairing bitstrings under all combinations and then counting the number of bits that differ in each pairing. The ideal result occurs when half of the bits in any pairing of the bitstrings differ.

$$\text{Inter-HD}_{i,j} = \frac{\sum_{k=1}^{|bs|} bs_{i,k} \oplus bs_{j,k}}{|bs|} \tag{2}$$

Eq. 2 gives the expression for Inter-HD for a pair of devices (i, j) of length $|bs|$ = 256 bits. Inter-HD is computed across all possible pairings of 1440 bitstrings, i.e., 1440*1439/2 = 1,036,080 combinations, and averaged. The mean Inter-HD is 49.9992%, which is very close to the ideal value of 50%. No failures of any type were observed in **LLK** regeneration over the 8+ hour run of the experiments.

8 SECURITY ANALYSIS

The TB authentication, session key generation and long-lived key generation PUF primitives act as the root-of-trust for the entire system, and as such, represent the primary targets of an attack. The TI extends the root-of-trust to customer devices using two different lighter-weight mechanisms. The security properties of the MZT and POP protocols, characterized as single **LLK** and challenge-response-based multi-**LLK**, respectively, draw from the root-of-trust by virtue of the amount and type of security related information transferred and stored on the devices. From the message-exchange diagrams, the security-sensitive data utilized by the MZT and POP protocols always originates from the TI, and is authenticated and encrypted in transit by the TB primitives. This requires adversaries to focus their attack on the PUF-based authentication and session key generation functions localized to customer devices. We perform an exhaustive analysis of various attack vectors at multiple levels of abstraction, from the physical device layer to protocol operations.

8.1 CRP Space Analysis

The TB security properties are rooted in the SiRF PUF's physical source of entropy, which are discussed at length in [21]. The size of the CRP space when utilizing DV sets of size 5072 (the number of $\mathbf{DV_R}$ and $\mathbf{DV_F}$ stored in each of the NAT$_{DB}$ and AT$_{DB}$ databases in our experiments) is given by $5072^2 * 2^{23}/5072$, yielding a total CRP space size of more than 2^{35} possibilities in cases where the TI stores all possible sets in its timing databases.

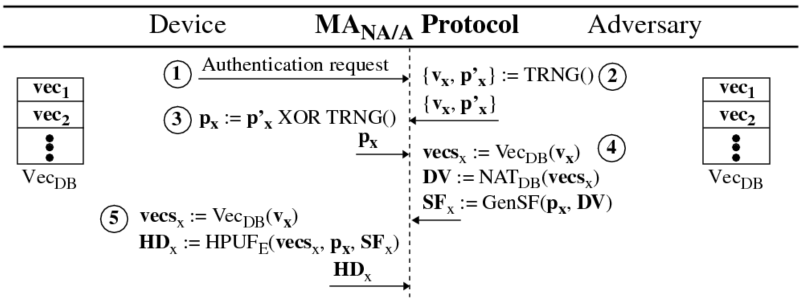


Fig. 12. Adversarial attack model to read out device responses.

8.2 Model-Building Attack Analysis

With the CRP space defined, we next describe an attack scenario where the adversary collects CRPs from a device-under-attack (DUA), as a precursor to building a machine-learning model. The message exchange diagram shown in Fig. 12 gives the sequence of operations carried out when the device requests authentication, which occurs as the first action in mutual authentication (MA$_{NA/A}$) in Figs. 2, 6, 7 and 9. Note that LLK generation and transmission is gated and encrypted by the MA$_{NA/A}$ and SKG$_{NA/A}$ functions. Therefore, the adversary must first defeat these security functions to extract response-based bitstrings, namely LLKs, from the PUF.

The following sequence describes the attack:

(1) The device controls the start of the transaction by sending an *Authentication request* message to the adversary on the right. The adversary is assumed to be in possession of the device to initiate this request.
(2) The adversary then specifies two parameters, v_x and p'_x. The v_x parameter specifies a 32-bit seed to an LFSR, that is used to pseudo-randomly select a set of vectors from the Vec$_{DB}$. We assume the adversary is in possession of a copy of the Vec$_{DB}$ and algorithm used by the DUA. The v_x and p'_x parameters are sent to the DUA.
(3) The device defines p_x by XOR'ing the adversary-generated p'_x with the output of its TRNG. The p_x parameter is used by both the DUA and adversary to seed another LFSR in the *DVDiffs module* that defines the pairing combinations of $\mathbf{DV_R}$ and $\mathbf{DV_F}$ used to create the **DVD**. We

assume the adversary knows the LFSR primitive used by the device to create the **DVD**. The adversary cannot, however, control the final value of p_x. The p_x parameter is transmitted to the adversary.

(4) The adversary extracts the vector sequence from the Vec_{DB} using a copy of the algorithm that the device uses. The adversary accesses an instance of the NAT_{DB} to extract timing data for the paths tested by the vectors. Since the adversary does not have access to the TI version of this database, he/she must create a version using simulation experiments, which assumes he/she has layout information related to the SiRF PUF instantiated on the DUA. The adversary constructs a 2048-byte array of SpreadFactors, $\mathbf{SF_x}$, from the data extracted from the NAT_{DB}. If SiRF PUF layout information is available, the $\mathbf{SF_x}$ may represent good approximations to what the TI provides to the device during a genuine exchange, otherwise the $\mathbf{SF_x}$ are random guesses. The adversary transmits the $\mathbf{SF_x}$ to the device.

(5) The device selects $\mathbf{vecs_x}$ and runs the SiRF PUF in enrollment mode, HPUF_E, to produce helper data $\mathbf{HD_x}$. The $\mathbf{HD_x}$ is a bitstring of length 2048 bits, which reflects which of the device-computed $\mathbf{DVD_{co}}$ are strong (1), and which are weak (0). The $\mathbf{HD_x}$ is transmitted to the adversary for storage and analysis.

From this description, the adversary controls the challenge sequence parameter, v_x, and optimization parameters, $\mathbf{SF_x}$, and receives a helper data bitstring, $\mathbf{HD_x}$, as the response. The v_x can be manipulated by the adversary to force specific paths to be timed while the $\mathbf{SF_x}$ can be manipulated incrementally to force weak bits to become strong, and vise versa. However, the actual response bits are not revealed (or even computed) by the SiRF PUF. Also note that the device can limit the rate of authentication attempts, which would limit the amount of data the adversary can collect over a given period of time.

It is the adversary's goal to produce an $\mathbf{HD_x}$ bitstring that is highly correlated to the $\mathbf{HD_x}$ that would be produced by the device under any arbitrary challenge, as a means of spoofing the identity of the genuine device to the TI. This requires the adversary to send accurate estimates of the $\mathbf{SF_x}$ to the device during the model-building phase, to enable it to learn the bit classification, weak or strong, corresponding to each of the 2048 $\mathbf{DVD}_{co}$.

The effectiveness of the model-building attack can be evaluated during the verifier authentication stage, where the device compares the $\mathbf{HD_x}$ bitstring input by the adversary with the version it creates internally. The adversary again controls the v_x and $\mathbf{SF_x}$ transmissions to the device but must now provide an $\mathbf{HD_x}$ that is highly correlated to $\mathbf{HD_x}$ produced internally by the device. To date, we have been unsuccessful in attempts to carry out this type of attack.

Session key generation, $\text{SKG}_{NA/A}$, is gated by the success of verifier authentication and is not performed unless the adversary succeeds in convincing the device it is communicating with an authentic server. Although response bitstrings are sent in the clear over the network for $\text{SKG}_{NA/A}$, the gating of this operation by $\text{MA}_{NA/A}$, the size of the CRP space and the lack of control over the parameter p_x will make it difficult to gather sufficient information to build an accurate model. If the adversary instead just eavesdrops on genuine device-TI authentications, the adversary will be tasked with identifying data communicated to/from a specific device because the $\text{MA}_{NA/A}$ functions are privacy-preserving, and therefore, they do not reveal the identity of the device in the exchanged messages.

8.3 Protocol Attacks

The security properties of MZT enrollment operation (Fig. 2), as well as all communication between customer devices and the TI during the bootstrap, withdrawal and deposit transactions utilize

$\mathrm{MA}_{NA/A}$, and therefore, they inherit the security properties of the TB functions, which were covered above in the context of model-building attacks.

8.3.1 Process Integrity and Side-channel Attacks. The HOSE provides a secure environment in the programmable logic (PL-side) of the FPGA which acts as a trusted execution environment (TEE). As such, the HOSE embeds secrets that the untrusted processor-side cannot access and serves as the primary mechanism to prevent duplication and double-spending of **eCt**. Access is managed through two 32-bit memory-mapped registers that expose a limited API to the ARM CPU, where only high-level protocol relevant methods are exposed, eliminating side-channel attacks. Unlike a typical TEE where asymmetric keys are used, remote attestation in the HOSE relies on material injected by the TI after the successful negotiation of strong PUF-based authentication. In particular, entries in the POP_{DB} are created by the TI and cannot be modified arbitrarily by other sources except by the HOSE itself in a prescribed manner dictated by the protocol execution steps.

8.3.2 Exfiltration of Secrets. Both the $\mathrm{MZT\_AT}_{DB}$ stored with the ARM processor and the POP_{DB} stored in the HOSE could be subject to key exfiltration attacks. Both databases are protected by encrypting the key material with a single-use PUF response. In Step 6 of the MZT protocol, a shared key is created by XOR-combining a $\mathbf{ZHK}_y$ key stored in the $\mathrm{MZT\_AT}_{DB}$ for customer y and a SiRF PUF regenerated key ($\mathbf{LLK}_{\mathbf{MZTx}}$), which is unique for every entry in the database. The HOSE is similarly configured, with a unique LLK_x generated from a unique challenge $Chlng_{Xx}$ per database entry. The POP_{DB} within the HOSE is additionally protected via API restrictions, which do not contain software calls to read or replace keys, thus limiting the threat landscape to device physical attacks.

8.3.3 Double-Spend Attacks. Alice is tempted to double-spend in an offline context. Two scenarios are possible, one where Alice sends an identical token to Bob twice, and the other where Alice sends an identical token to Bob and Charlie. The design and use of the HOSE as described in Section 5.3.3 and Fig. 8 guards against both attacks. Here, we rely on the integrity of the HOSE, which is equivalent to a trusted execution environment where the ordering of operations and their trusted execution is guaranteed. In Step 7 of Fig. 8, Alice securely deletes **eCt** (by writing zeros to memory) upon receiving an ACK from Bob. This ensures that tokens are destroyed upon transfer and cannot be used in a later transaction. In future, we can consider certain PUF properties to further strengthen this protection.

8.3.4 Replay Attacks. Replay attacks are mitigated via a key refresh mechanism embedded within the protocol. In Step 6 of the protocol (Fig. 8), Bob encrypts and transmits a new challenge to Alice, C_3. In Steps 7 and 8, Alice decrypts the challenge, generates a new response and associated helper data, POP'_{Ab}, HD_A and transmits an encrypted version (encrypted with the original POP_{Ab}) back to Bob, who stores an encrypted version in his POP database (POP_{DB}). Both Alice and Bob will use POP'_{Ab} as the session key for the next transaction. An adversary attempting to replay the transcript of a prior transaction would not know the new session key and would not be able to encrypt or decrypt packets.

8.3.5 Man-in-the-Middle (MITM) Attacks. The protocol relies on pre-shared keys to transfer sensitive information. We can observe that all messages transferred over the insecure channel are encrypted, and each stage of the protocol, be it withdrawal, transfer or deposit, begins with a mutual authentication and session key negotiation step. Unless an adversary has prior knowledge of the keys gleaned from other attacks described herein, it cannot extract any new information from Alice or Bob. Ergo, the adversary can only perform a denial of service (DoS) attack at best in which case the protocol simply aborts.

8.3.6 Device Attacks. Certain physical attacks may still be possible with the current implementation. One possible vector is a NAND mirroring attack, whereby a copy of the NVM eCt$_{DB}$ is made while the device is at rest. After a transaction, the copy is restored, reflecting the prior token state. This malicious operation is equivalent to double-spending because it restores **eCt** that Alice has just transferred to Bob. While the double-spend attempt will eventually be detected during a deposit sequence, and further only one device will succeed in making a deposit, devices in the field participating in offline, transitive payments will still be defrauded. A second possible vector is a fault injection attack, whereby a fault is injected into Alice's device to prevent deletion of **eCts** after the transfer to Bob is complete. While it may be difficult to pinpoint the precise fault location in the FPGA fabric, a fault could be injected in the NVM controller itself. In both cases, the root cause stems from the inability of the device to maintain a trusted state across power cycles. We aim to investigate possible solutions for this attack in future work.

9 SUMMARY AND CONCLUSIONS

A PUF-Cash protocol is proposed, and assessed in FPGA hardware experiments, that leverages two novel security protocols, called propagation-of-provenance (POP) and mutual-zero-trust (MZT), both of which can take place exclusively between two untrusted parties and both constructed to eliminate the need to interact with a trusted authority. POP is applied in this paper to secure the propagation of eCash tokens (**eCt**) from one customer to another. It leverages the exponential challenge-response space of a strong PUF, called the SiRF PUF, to allow recipients of **eCt** to authenticate their origin back to the token issuing authority. Successive hand-offs between devices utilize the combination of the payer's SiRF-instantiated device and the recipient's stored PUF responses to authenticate **eCt** signatures.

A hardware-obfuscated secure enclave (HOSE) is introduced as a means of alleviating software security issues which are difficult to address in microprocessor environments, and as a means of preventing end-users from attempting to double-spend the **eCt** which they store. The HOSE is implemented entirely in the programmable logic of an FPGA embedded within a system-on-chip (SoC) device. A PUF-Cash protocol implementing bootstrap, withdrawal, transfer and deposit transactions is built on top of the HOSE, and POP and MZT protocols. Future work will investigate database-cloning attacks, network attack scenarios, and will further explore the model-building resistance of the SiRF PUF.

REFERENCES

[1] Nur Arifin Akbar, Amgad Muneer, Narmine ElHakim, and Suliman Mohamed Fati. 2021. Distributed Hybrid Double-Spending Attack Prevention Mechanism for Proof-of-Work and Proof-of-Stake Blockchain Consensuses. *Future Internet* 11 (2021). https://doi.org/10.3390/fi13110285

[2] Urbi Chatterjee, Rajat Subhra Chakraborty, and Debdeep Mukhopadhyay. 2017. A PUF-based secure communication protocol for IoT. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 3 (2017), 1–25.

[3] Urbi Chatterjee, Vidya Govindan, Rajat Sadhukhan, Debdeep Mukhopadhyay, Rajat Subhra Chakraborty, Debashis Mahata, and Mukesh M. Prabhu. 2019. Building PUF Based Authentication and Key Exchange Protocol for IoT Without Explicit CRPs in Verifier Database. *IEEE Transactions on Dependable and Secure Computing* 16, 3 (2019), 424–437. https://doi.org/10.1109/TDSC.2018.2832201

[4] David Chaum. 1983. Blind Signatures for Untraceable Payments. In *Advances in Cryptology*. Springer US, Boston, MA, 199–203.

[5] David Chaum, Amos Fiat, and Moni Naor. 1990. Untraceable Electronic Cash. In *Advances in Cryptology*, Shafi Goldwasser (Ed.). Springer, 319–327.

[6] Wenjie Che, Mitchell Martin, Goutham Pocklassery, Venkata K. Kajuluri, Fareena Saqib, and Jim Plusquellic. 2017. A Privacy-Preserving, Mutual PUF-Based Authentication Protocol. *Cryptography* 1, 1 (2017).

[7] Jeroen Delvaux, Roel Peeters, Dawu Gu, and Ingrid Verbauwhede. 2015. A Survey on Lightweight Entity Authentication with Strong PUFs. *ACM Comput. Surv.* 48, 2, Article 26 (oct 2015), 42 pages. https://doi.org/10.1145/2818186

[8] Digilent Corporation 2023. *ZYBO-Z7 Reference Manual.* Digilent Corporation, Pullman, WA. https://digilent.com/reference/programmable-logic/zybo-z7/reference-manual?redirect=1

[9] Ujjwal Guin, Adit Singh, Mahabubul Alam, Janice Cañedo, and Anthony Skjellum. 2018. A Secure Low-Cost Edge Device Authentication Scheme for the Internet of Things. In *2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID).* 85–90. https://doi.org/10.1109/VLSID.2018.42

[10] Isaiah Hull Hanna Armelius, Carl Andreas Claussen. 2021. *On the possibility of a cash-like CBDC.* Technical Report. Sveriges Riksbank Payments Department and Research Division. https://www.riksbank.se/globalassets/media/rapporter/staff-memo/engelska/2021/on-the-possibility-of-a-cash-like-cbdc.pdf

[11] Hipp, Wyrick And Company, Inc 2023. *Sqlite3 webpage.* Hipp, Wyrick And Company, Inc, Charlotte, NC. https://www.sqlite.org/index.html

[12] Tarek Idriss and Magdy Bayoumi. 2017. Lightweight highly secure PUF protocol for mutual authentication and secret message exchange. In *2017 IEEE Int. Conf. on RFID Technology Application.* 214–219. https://doi.org/10.1109/RFID-TA.2017.8098893

[13] N. Irtija, E.E. Tsiropoulou, C. Minwalla, and J. Plusquellic. 2022. True Random Number Generation with the Shift-register Reconvergent-Fanout (SiRF) PUF. In *2022 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST).*

[14] Özgür Kesim, Christian Grothoff, Florian Dold, and Martin Schanzenbach. 2022. Zero-Knowledge Age Restriction for GNU Taler. Springer-Verlag, Berlin, Heidelberg, 110–129. https://doi.org/10.1007/978-3-031-17140-6_6

[15] Bin Lian, Gongliang Chen, and Jianhua Li. 2014. Provably secure E-cash system with practical and efficient complete tracing. *International Journal of Information Security* 13 (2014). https://link.springer.com/article/10.1007/s10207-014-0240-2

[16] Mahabub Hasan Mahalat, Dipankar Karmakar, Anindan Mondal, and Bibhash Sen. 2021. PUF Based Secure and Lightweight Authentication and Key-Sharing Scheme for Wireless Sensor Network. *Journal of Emerging Technologies in Computer Systems* 18, 1, Article 9 (Sep 2021), 23 pages.

[17] Mahabub Hasan Mahalat, Shreya Saha, Anindan Mondal, and Bibhash Sen. 2018. A PUF based Light Weight Protocol for Secure WiFi Authentication of IoT devices. In *2018 8th Int. Symp. on Embed. Comp. and System Design.* 183–187. https://doi.org/10.1109/ISED.2018.8703993

[18] Md Jubayer al Mahmod and Ujjwal Guin. 2020. A Robust, Low-Cost and Secure Authentication Scheme for IoT Applications. *Cryptography* 4, 1 (2020). https://doi.org/10.3390/cryptography4010008

[19] Luca Mainetti, Matteo Aprile, Emanuele Mele, and Roberto Vergallo. 2023. A Sustainable Approach to Delivering Programmable Peer-to-Peer Offline Payments. *Sensors* 23, 3 (2023). https://doi.org/10.3390/s23031336

[20] Jianbing Ni, Man Ho Au, Wei Wu, Xiapu Luo, Xiaodong Lin, and Xuemin Sherman Shen. 2023. Dual-Anonymous Off-Line Electronic Cash for Mobile Payment. *IEEE Transactions on Mobile Computing* 22, 6 (2023), 3303–3317. https://doi.org/10.1109/TMC.2021.3135301

[21] Jim Plusquellic. 2022. Shift Register, Reconvergent-Fanout (SiRF) PUF Implementation on an FPGA. *Cryptography* 6, 4 (2022). https://doi.org/10.3390/cryptography6040059

[22] Jim Plusquellic, Eirini Eleni Tsiropoulou, and Cyrus Minwalla. 2023. Privacy-Preserving Authentication Protocols for IoT Devices Using the SiRF PUF. *IEEE Transactions on Emerging Topics in Computing* (2023), 1–16. https://doi.org/10.1109/TETC.2023.3296016

[23] John Ross Wallrabenstein. 2016. Practical and Secure IoT Device Authentication Using Physical Unclonable Functions. In *2016 IEEE 4th Int. Conference on Future Internet of Things and Cloud.* 99–106. https://doi.org/10.1109/FiCloud.2016.22

[24] Bo Yang, Kang Yang, Zhenfeng Zhang, Yu Qin, and Dengguo Feng. 2016. AEP-M: Practical Anonymous E-Payment for Mobile Devices Using ARM TrustZone and Divisible E-Cash. In *Information Security*, Anderson C A Nascimento Matt Bishop (Ed.). Springer International Publishing, Cham, 130–146. https://link.springer.com/chapter/10.1007/978-3-319-45871-7_9

[25] Meng-Day Yu, Matthias Hiller, Jeroen Delvaux, Richard Sowell, Srinivas Devadas, and Ingrid Verbauwhede. 2016. A Lockdown Technique to Prevent Machine Learning on PUFs for Lightweight Authentication. *IEEE Transactions on Multi-Scale Computing Systems* 2, 3 (2016), 146–159. https://doi.org/10.1109/TMSCS.2016.2553027

[26] Jingliang Zhang, Lizhen Ma, and Yumin Wang. 2007. Fair E-Cash System without Trustees for Multiple Banks. In *2007 Intl.1 Conf. on Computational Intelligence and Security Workshops (CISW 2007).* 585–587. https://doi.org/10.1109/CISW.2007.4425563

[27] Jiliang Zhang and Gang Qu. 2020. Physical Unclonable Function-Based Key Sharing via Machine Learning for IoT Security. *IEEE Transactions on Industrial Electronics* 67, 8 (2020), 7025–7033. https://doi.org/10.1109/TIE.2019.2938462