Digital Currency with Seamless Online/Offline Functionality (PUF-Cash 3.0)

A STUDENT*, 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 and e-cash, we introduce a digital currency where physical unclonable function (PUF) devices satisfy the twin properties of being enrolled in a participating set and sufficient security features to act as their own root of trust in bilateral transfers. Presented in this work is an electronic cash ecosystem consisting of device enrollment, token creation and a bilateral secure transfer mechanism that authenticates devices as ones issued by the authority and propagates the token from sender to recipient in a secure fashion without incurring any third party dependencies at transfer time. A Propagation of Provenance (PoP) is proposed to establish a chain of custody from token creation to redemption. Authentication is based on challenge response pairs (CRPs) drawn from labelled and unlabelled databases established at a central authority during enrollment. A mutual self-trust Underlying PUF primitives enable the The central authority can establish that all transactions occurred between enrolled devices without revealing the identity of the interacting devices. Devices perform a pairwise mutual authentication to establish a secure channel, relying on challenge response pairs that only PUFs are familiar with. Only devices enrolled at the central authority can participate in transactions, eliminating counterfeit risk. A thorough

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

Additional Key Words and Phrases: security, electronic money, digital currency, networks

ACM Reference Format:
A Student, Cyrus Minwalla, Eirini Eleni Tsiropoulou, and Jim Plusquellic. 2022. Digital Currency with Seamless Online/Offline Functionality (PUF-Cash 3.0). *J. ACM* 37, 4, Article 111 (August 2022), 15 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

2 ECOSYSTEM

3 SIRF PUF AND HARDWARE SECURITY PRIMITIVES

The authentication bitstrings, encryption keys, random nonces and eCash tokens utilized within the proposed MST and eCash protocols are derived from a physical unclonable function (PUF) called the shift-register reconvergent-fanout (SiRF) PUF [1]. The SiRF PUF serves as the root-of-trust in

*Authors contributed equally to this research.

Authors' addresses: A Student, student@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.

0004-5411/2022/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

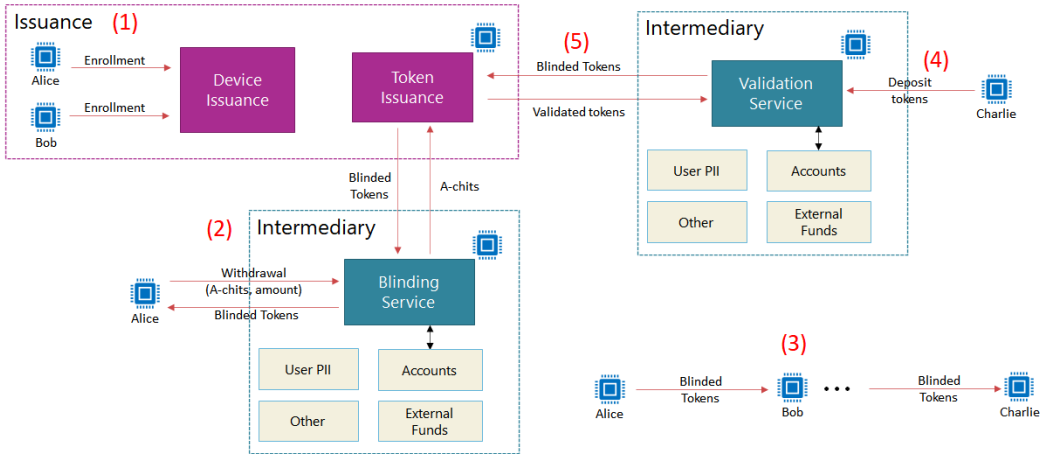

Fig. 1. High level overview of PUF-Cash V3.0

the proposed MST and eCash protocols, and its exponential challenge-response-pair (CRP) space provides the foundation for the strong security and trust properties associated with these protocols.

3.1 SiRF PUF Architecture and Algorithm

The SiRF PUF architecture and algorithm are shown on the left and right sides of Fig. 2, 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-out and fan-in, 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). A set of 32 flip-flips launch a set of transitions onto the path inputs and a time-to-digital converter (TDC) is used to measure the delay of signal transitions emerging on the path end points. The TDC creates high resolution integer representations (*digital values* or *DVs*) of the path delays. The DV are stored in a BRAM and are used as inputs to a post-processing algorithm responsible for generating authentication bitstrings and keys. The SiRF PUF architecture incorporates a mode switch to enable the entropy source and algorithm to be used for true-random number generation (TRNG), which is able to supply an unlimited number of random nonces for cryptographic operations [5].

3.2 SiRF PUF Challenge

The challenge for the SiRF PUF consists of several components, annotated as $Chlng_a$ and $Chlng_b$ in Fig. 2. A vector of binary values, v, controls the configuration of the paths through the netlist of shift-registers and logic gates. Although v represents a sequence of netlist configurations vectors that are applied to generate a set of 2048 rising and 2048 falling DV, in our implementation, v actually refers to a 32-bit number that represents a seed to an LFSR, which pseudo-randomly selects the sequence of configuration vectors from a $Vecs_{DB}$ database (not shown in the figure). The remaining components of the challenges are given as p, SF, HD. The p component is used to specify parameters to the SiRF PUF algorithm. The parameters are used by the SiRF PUF algorithm as control inputs to mathematical operations applied to the 4096 DV stored in the BRAM. The first operation involves creating differences in the path delay values, called DVD (*DVDiff module*). The DVD are then compensated to remove variations in delay introduced by global performance differences and operational environments that are not nominal with respect to temperature and supply voltage, to

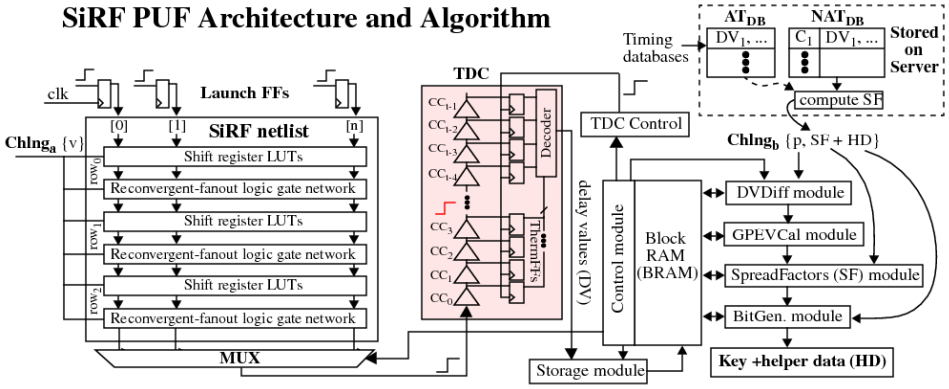

Fig. 2. SiRF PUF Architecture and Algorithm.

produce DVD_c (*GPEVCal module*). A third operation removes delay bias introduced by differences in the length of the tested paths, which is accomplished through the application of SpreadFactors (*SF*) (*SpreadFactors module*). The *HD* component refers to helper data that is needed for bitstring or encryption key regeneration.

The *SF* are computed by a server using timing data (*DV*) collected from a sample of devices during provisioning, and are transmitted to the device when the PUF is generating the bitstring or encryption key for the first time; a process called enrollment. The *SF* are used to optimize the statistical quality of the PUF-generated bitstrings, and are computed using the timing data stored in one of two timing databases, labeled AT_{DB} and NAT_{DB} in Fig. 2. The process used to construct these databases is described in the next section.

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. Although the details of the SiRF PUF algorithm are not presented here, the size of the challenge data is an important criteria when assessing the storage requirements of the MST and PUF-Cash protocols. As discussed in the following sections, the bitstring and keys used in the protocols are 256-bits, which requires approximately 512 bytes for *HD* and 4096 bytes for *SF*. However, the *SF* are derived from and can be applied to any device in the population, which reduces challenge overhead significantly.

3.3 PUF-Based Authentication and Encryption Key Primitives

Mutual authentication (*MA*) and session key generation (*SKG*) is accomplished in the MST and PUF-Cash protocols using one of two PUF-based protocols. The versions described in this section are used between the TI and customer devices, and between the TI and FI, and are referred to as timing-based (TB) versions, because they utilize the AT_{DB} and NAT_{DB} timing databases stored on the TI (see upper right of Fig. 2). The timing databases encode a large set of challenge-response pairs (CRPs) for each device and represent the root-of-trust for the TI. Alternative, lighter-weight versions of *MA* and *SKG*, referred to as challenge-response-based (CRB), utilize the CRPs directly. The CRB versions are used for mutual authentication and encryption key generation between pairs of customer devices and between a customer device and the FI, where the timing databases are not available. Despite these differences, it is always the case that one of the two parties in the engagement utilize a PUF to generate the bitstrings and keys for authentication or session key generation.

The timing databases give the TI the ability to exactly reproduce the responses to the challenges presented to the PUF devices. The timing databases are constructed as devices are added to the PUF-Cash ecosystem during provisioning, and represent an alternative to the challenge-response-pair (bitstring) format usually used for PUFs. The digitized values of the path delays enable bitstrings and keys to be constructed on-the-fly by executing a software version of the SiRF PUF algorithm on the server. The parameters p used as input to the algorithm expand the challenge-response space by a factor greater than 1000 over the alternative of storing challenge-response bitstrings directly. The timing values can be reused to generate more than 1000 256-bit authentication bitstrings (or encryption keys), while maintaining high levels of model-building resistance. Experimental results supporting the model-building resistance of a delay PUF similar to SiRF is presented in [3].

The TB operations carried out between the TI and the fielded devices is annotated as MA_{NA} or MA_A (for mutual authentication) and SKG (for session key generation) in the message exchange diagrams shown in this paper. The subscript NA refers to a non-anonymous mutual authentication operation, where the device identifies itself to the TI using it's unique chip number, e.g., C_1. The NAT_{DB} is used for non-anonymous authentication, where the timing data is labeled in the database with the chip number as shown in Fig. 2. The subscript A refers to an anonymous authentication, where the TI is able to confirm that the device has been enrolled, but is not able to identify the customer's name (chip number) during authentication. The AT_{DB} and NAT_{DB} are built with distinct DV, and the ordering of device timing data sets in the two databases is scrambled to provide anonymity. The SiRF PUF has more than 10 million unique paths that can be timed, making it possible to select large numbers of unique DV per database. The number of DV per device in each database in the implementation presented in this paper is approximately 15,000.

The basic PUF-based security functions leveraged in the proposed MST and PUF-Cash protocols also include true-random-number-generation ($TRNG$) and long-lived key (LLK) generation. The SiRF $TRNG$ function uses the same measurement and processing operations shown in Fig. 2 to produce a sequence of nonces, but randomizes the v and p challenge components and eliminates the SF and XMR components [5]. The LLK function, on the other hand, needs to reproduce the bitstring and therefore, implements both enrollment and regeneration modes, where HD is produced when the bitstring is generated for the first time that is later used as input when the bitstring is reproduced. However, unlike MA and SKG which involve two parties, LLK is used when the device needs to reproduce the bitstring in a stand-alone environment. The LLK function plays a central and novel role in the MSP and PUF-Cash protocol operations.

The ability to exactly reproduce bitstrings is a requirement for the MA, SKG and LLK functions, independent of whether bitstring generation occurs between two devices or is carried out at two separate times within a single device. All of these security functions leverage three reliability enhancing techniques that are integrated into the SiRF PUF algorithm. The *GPEVCal* module first compensates the DVD for sources of variation that are not entropy-related. A thresholding scheme is then applied to select DVD_c (after SF are applied) that have a low probability of generating bit flip errors. Last, an XMR technique incorporates redundancy in the bitstring enrollment process that is used to further decrease the chance of bit flip errors during regeneration of the bitstring or key. The details of these techniques and experimental results showing their effectiveness are presented in previous work [6] [4].

4 MUTUAL-SELF-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 other entities, e.g., trusted third parties (TTPs), to assist with authentication and session key generation. In such cases, trust within the two-party system must be derived from the devices themselves. We use the term

mutual-self-trust (MST) 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. Given the access restrictions of the MST environment, devices will necessarily need to obtain and securely store local data that enables them to establish a trusted relationship.

A key challenge to implementing authentication, session key generation and an eCash transaction protocol that meets the security properties described earlier is to accommodate transactions between any pair of customer devices from the population. Given that each device must store MST data on every other device within the population, the composition of the MST data must be compact to be practical for an embedded system, while simultaneously providing each device with sufficient confidence that the relationship can be trusted. In this section, we describe a MST system that meets these requirements, and later describe a PUF-Cash protocol that leverages the MST data format for providing a secure and trusted payment mechanism.

4.1 MST Enrollment Operations

The security properties of the MST protocol are derived from two primitives, one hardware-based, namely the SiRF PUF and one cryptography-based, namely a secure hash function. The PUF serves as the root-of-trust and is the source of entropy while secure hash provides obfuscation, data integrity and authentication. The integration of the two primitives provides a highly secure and lightweight mechanism for enabling Alice and Bob to authenticate and to generate a shared secret for encrypting communications.

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

$$\mathbf{ZHK} := \mathbf{SHA\text{-}3}(\mathbf{LLK} \oplus \mathbf{n}) \tag{1}$$

The MST scheme requires Alice and Bob's devices to each carry out an enrollment operation with the *Token Issuer* or TI, as shown in the upper portion of Fig. 3. Enrollment is performed by Alice and Bob separately during provisioning, i.e., after device manufacturing, and periodically in the field as needed. The following sequence of operations are carried out by both both Alice and Bob separately during enrollment:

(1) Alice and Bob authenticate non-anonymously via MA_{NA} and generate a session key SK_{TA} using SKG with the TI. As discussed in Section 3.3, the TI uses the TB versions of these security functions given the availability of the timing databases. The non-anonymous authentication allows the TI to identify Alice and Bob as ID_x
(2) The TI transmits unique challenges to Alice and Bob, labeled $Chlng_{ZT}$.
(3) Alice and Bob apply the challenge to their hardware PUFs, $HPUF_E$, in enrollment mode, to generate a long-lived key, ZT_LLK_1, and helper data HD_1. Alice and Bob store the challenge information in the LLK_{DB}, which includes the components discussed earlier in reference to Fig. 2, i.e., v, p, SF and HD, to enable regeneration of ZT_LLK_1 at any point later in the field.
(4) Once the ZT_LLK_1 is generated, the TI sends a request to Alice and Bob to generate num_ZHKs.
(5) Alice and Bob construct a set of tuples $\{ZHK_i, n_i\}$ by running their PUF's TRNGs to generate a sequence of nonces, n_i, which are XOR'ed with ZT_LLK_i and used as the input to the AT creation function given by Eq. 1.

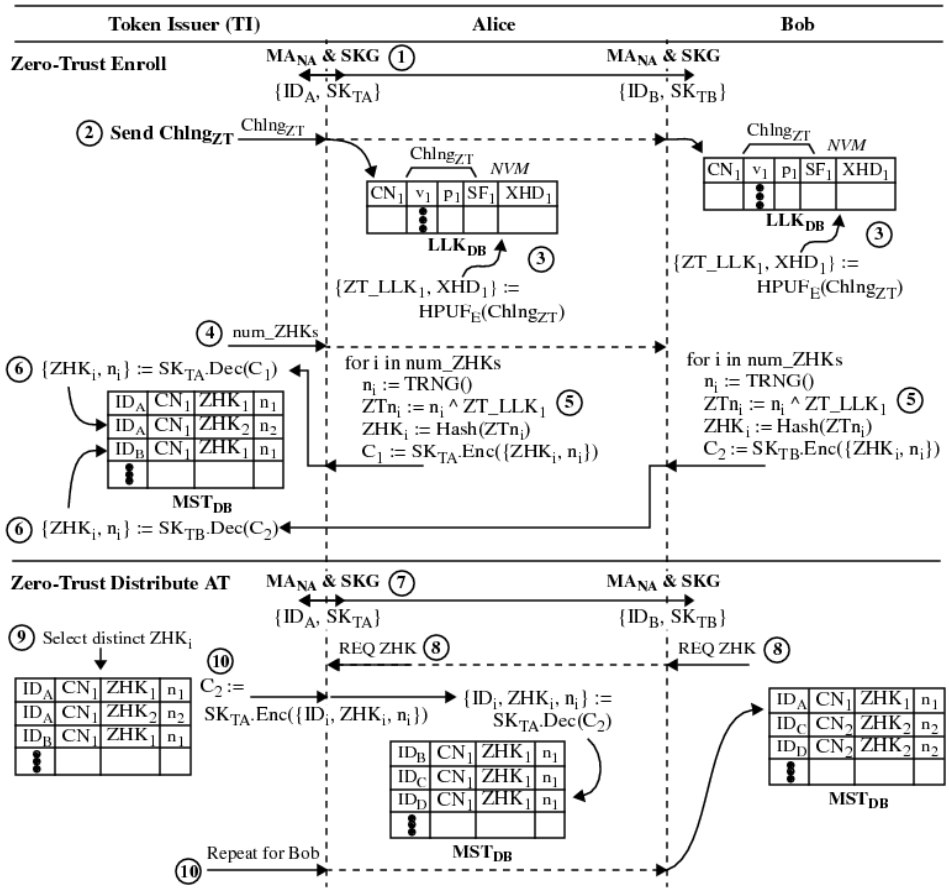

Fig. 3. MST enrollment and authentication token (ZHK_i) exchange process between Token Issuer, Alice and Bob.

(6) The tuples are encrypted using SK_{TA} and transmitted to the TI. The TI decrypts and stores them along with the customer's device identifier ID_x in its MST_{DB}.

The TI collects ZHK_i from all customer devices to build the MST_{DB}. Note that each entry consumes only 72 bytes, assuming 4-byte integers for the ID_x and CN_x fields, and 32 byte for each of the ZHK_i and n_i. Alice and Bob will retrieve one unique ZHK_i tuple from the TI for each customer during the distribution operation shown along the bottom of Fig. 3, which they store in their MST_{DB}.

The MST_{DB} constructed on Alice and Bob's devices can include millions of elements, one element for each potential customer in the population. Given a customer entry requires 72 bytes within MST_{DB}, it follows that a million customers would require a database of 72 MB. Moreover, the data can be stored in a standard off-the-shelf NVM, e.g., SD card, where storage capacities of 4 GB or larger are common. Further protections can be provided by encrypting the data on the SD card. We propose a secure enclave in PUF-Cash that can be used here for implementing this type of additional layer of security.

4.2 MST In-Field Operations

The MST in-field process is carried out when Alice contacts Bob for goods or services in an environment where connectivity exists only between Alice and Bob. The message exchange protocol is shown in Fig. 4, and is described as a sequence of the following operations:

(1) The transaction begins with Alice sending Bob a request to authenticate and generate a shared session key.
(2) Alice sends Bob an identifier, ID_A, that allows Bob to locate the corresponding AT in his MST_{DB}.
(3) Bob responds to Alice with an Ack or Nak (as *status*) on whether or not he possesses an AT for Alice. He also responds with his own identifier ID_B in cases where he possesses an AT for Alice.
(4) Alice determines if she has an AT for Bob in her MST_{DB} using the Bob's ID_B.
(5) She transmits a corresponding Ack or Nak to Bob.
(6) Alice and Bob retrieve the AT for the other party from their MST_{DB}, which is represented by the tuple ZHK_x, n_x with $x := b$ or a, respectively.
(7) **Shared Key Generation:** Alice and Bob exchange the nonce components, n_x, of the AT.
(8) Both parties regenerate their long-lived keys ZT_LLK using challenge information stored in their LLK_{DB}, and then compute a local version of the ZHK'_x using $Hash(ZT\_LLK_x \oplus n_x)$ (NOTE: XOR operation is annotated as ˆ in the diagram).
(9) Alice and Bob create a shared key SK_{AB} by XOR'ing the local copy of ZHK'_x with the ZHK_x that they store for the other party in their MST_{DB}.
(10) **Authentication:** Authentication begins with Alice and Bob encrypting the n_x they received from the other party with the newly created shared key SK_{AB} to create en_x.
(11) Alice and Bob exchange the encrypted nonces en_x.
(12) Alice and Bob decrypt the en_x using the shared key.
(13) Alice and Bob compare the decrypted n_x with the ones they store in their MST_{DB}.
(14) The status of the comparison is shared with the other party with each transmitting an Ack or Nak. Alice and Bob have authenticated and posses a shared key at this point assuming both have acknowledged that the nonces n_x match their own local copies.
(15) **Refresh ZHKs:** Alice and Bob generate new nonces n_{x_n} by running their TRNGs, and then compute new AT by hashing $(ZT\_LLK_x \oplus n_{x\_n})$.
(16) Alice and Bob encrypt the new AT as C_x with their shared session keys SK_{AB}.
(17) Alice and Bob exchange the C_x.
(18) They both decrypt the C_x to recover the new AT.
(19) They both store the new AT in their MST_{DB}, replacing the existing AT used in this transaction.

The MST in-field operations are light-weight using only PUF and secure hash operations. The refresh operation ensures that future transactions can occur between 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.

5 PUF-CASH PROTOCOL

The PUF-Cash protocol is a eCash system that uses only lightweight security functions, namely PUFs and secure hash, to allow any arbitrary pair of devices to securely exchange funds. The operating environment used in the protocol version described here is characterized as off-line, requiring Alice and Bob to authenticate using the MST model described in the previous section. However, the PUF-Cash protocol also supports a peer-trust model, where limited connectivity

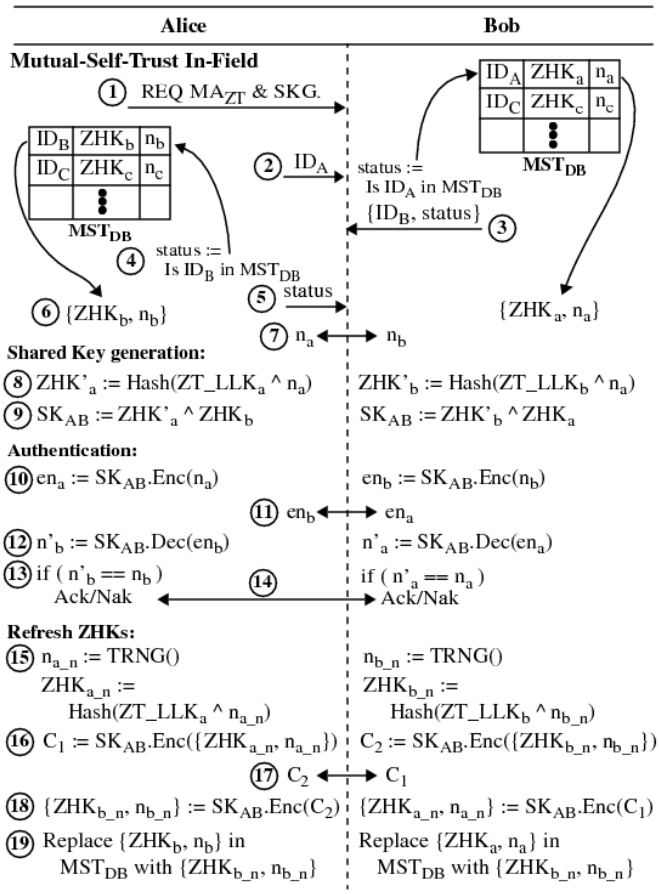

Fig. 4. Mutual-self-trust in-field authentication and session key generation between Alice and Bob.

exists that allows other devices to be consulted during authentication, and a full-trust model for cases in which a trusted-third-party (TTP) can be consulted. The peer-trust and full-trust models will be presented in future work.

PUF-Cash also supports unlimited transitivity of eCash tokens (*eCt*) through multiple parties, e.g., Alice pays Bob, and then Bob pays Charlie with the same eCt, without first requiring Bob to validate his eCt with the financial institution (FI) and/or token issuers (TI). PUF-Cash introduces a novel security primitive called propagation-of-provenance (POP) and a hardware-oriented secure enclave (HOSE) as a means of engendering eCash transitivity with important security properties. In particular, POP and HOSE enable each party in a chain of eCash value transfers to authenticate the eCt that they receive, and to validate provenance back to the point of origin, namely the FI and TI. Furthermore, the HOSE is designed to prevent Alice from double spending her eCt.

The HOSE is implemented as a set of state machines embedded in a hardwired portion of the device, or, in the case of FPGAs, in the programmable logic. HOSE limits interactions and potential attacks with software applications, including the PUF-Cash software components, through an interface that provides only two 32-bit hardware registers. HOSE incorporates the SiRF PUF to serve as the root-of-trust for generating keys on-the-fly as needed. The proposed POP scheme leverages the PUF keys, and the capability of the SiRF PUF to produce a large number of them,

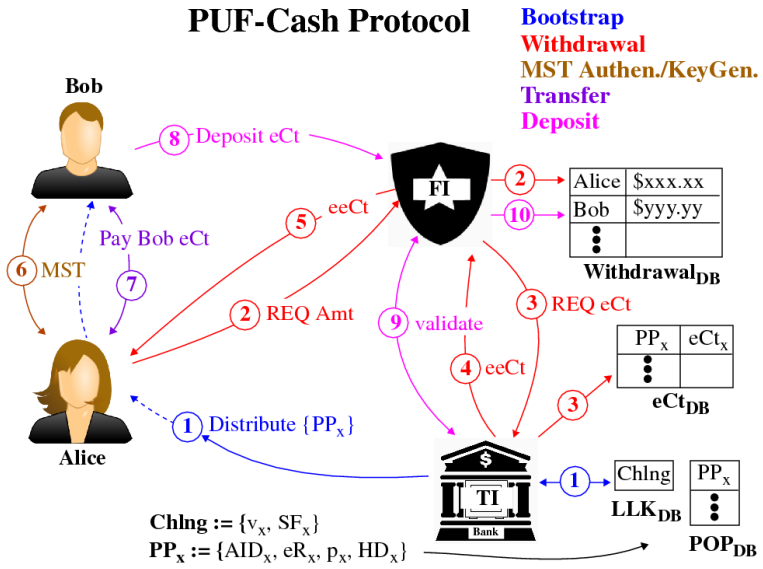

Fig. 5. High level overview of PUF-Cash V3.0

to allow Bob to confirm the provenance of Alice's eCt, and to enable the provenance validation property to be propagated to Charlie, Ted, etc.

5.1 PUF-Cash Overview

In this section, we present the PUF-Cash protocol at a high-level of abstraction to emphasize the sequence of operations as well as the security properties associated with the HOSE and POP scheme. The sequence of operations are numbered 1 through 10 in Fig. 5.

(1) Similar to the MST protocol, the bootstrap (enrollment) operation associated with the PUF-Cash protocol generates and distributes a set of POP cryptographic tuples ($POPcycle_x$) to Alice and Bob's devices (which is represented as a circled 1 and is abbreviated as PP_x in the figure).
(2) Alice contacts here financial institution (FI) and requests a withdrawal from her account. 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 her request to the token issuer (TI) to generate e-Cash tokens (eCt_x). The TI records her anonymous PP_x (also part of the REQ transaction) in its eCt_{DB} database.
(4) The TI encrypts the eCt_x as $eeCt_x$ using a shared session key that is created between Alice and the TI anonymously (see message exchange diagram below for details), and transmits the $eeCt_x$ to FI.
(5) The FI simply forwards the $eeCt_x$ to Alice. This feature of using the FI as an intermediary provides anonymity for Alice to the TI. Moreover, her eCt_x are also anonymous to the FI because they are encrypted with a session key that only Alice and the TI know.
(6) Alice contacts Bob for a payment transaction, and then run the ZeroTrust (ZT) mutual authentication and session key generation algorithm described previously.
(7) Assuming authentication succeeds, Alice pays Bob with her eCt_x by encrypting them with the Alice-Bob ZT session key.

(8) Bob at some point in the future regains internet connectivity (gets on-line) and deposits the eCt_x to his FI. Note that although Alice and Bob use the same FI as shown, this is not a requirement. Moreover, Bob may use his eCt_x to pay another party, and choose not to engage in a deposit transaction, i.e., transitivity is supported in this POP scheme.
(9) The FI contacts the TI and asks the TI to validate the eCt_x, as a precursor to the FI accepting the eCt_x as a valid deposit.
(10) Assuming validation succeeds, the FI credits Bob's account and the eCt_x are discarded.

5.1.1 POP Unique Characteristics. As shown in Step 1 of Fig. 5, the *TI* generates the POP cryptographic tuples, PP_x, in contrast with the MST scheme where Alice and Bob devices' generate their own *AT* using their *ZT_LLK*. Moreover, the *TI* generates the PP_x using the anonymous timing database (AT_{DB}), which will be leveraged in PUF-Cash to provide anonymity for Alice and Bob's eCash tokens. A third distinction of the POP scheme is that the *TI* generates a unique challenge for creating Alice and Bob's PP_x, and the POP_{DB} stores the challenge and encrypted responses, while the MST_{DB} stores only a nonce-hashed version of the responses.

The *TI* uses a specialized process to create the entire set of PP_x for each of Alice or Bob's POP_{DB} that significantly reduces the storage requirements for the challenges. The PP_x transmitted to Alice and Bob's device is composed of two components, a population-based component (PBC) and a customer-based component (CBC), each stored in two different databases labeled LLK_{DB} and POP_{DB} in Fig. 5, respectively. The PBC component of the PP_x labeled *Chlng* in the LLK_{DB} is used for all customers in Alice's POP_{DB}, and is defined as the tuple v_x, SF_x (the components of the SiRF challenge were described earlier in Section 3). The CBC components of the PP_x are stored in the POP_{DB}, one element for each customer, and are defined as a tuple AID_x, eR_x, p_x, HD_x. The AID_x is a 4-byte integer representing a customer's anonymous ID, the eR_x is the encrypted PUF's response to the challenge (32 bytes), the p_x component is a 4-byte nonce used to select parameters to the SiRF algorithm and HD_x is the helper data (512 bytes). Therefore, the size of each POP database element is 552 bytes, which is quite a bit larger than the MST_{DB} at more than 1/2 GB for 1 million customers. However, the benefits of the larger database are significant as we describe next.

One immediate benefit of the POP scheme over the MST scheme is related to the distribution operation. The POP scheme is able to deliver a complete set of PP_x to Alice and Bob's devices during the BootStrap operation, in contrast to the MST scheme where the set delivered to Alice and Bob's devices is composed of only those devices that have enrolled beforehand. Here, complete refers to the set of PP_x that includes all registered PUF devices.

Second, the challenge information stored by Alice for, e.g., Bob's device, allows a stronger form of authentication whereby Bob is required 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 (in the PP_x tuple) by running a 'soft PUF' version of the SiRF 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.

Last, Alice is able to refresh Bob's PP_x after every engagement with Bob in two different ways. In one scenario, Alice generates a new challenge for Bob's device by changing the p_x component, and then asks Bob to generate a new response (which she receives encrypted using the session key she generated using the initial PP_x). She then replaces the PBC component of Bob's PP_x in her POP_{DB} with the new information. In the second scenario, Alice authenticates with the *TI* and asks the *TI* to perform this operation. Although this requires Alice to be on-line, it allows Alice to leverage the stronger form of authentication using a trusted-third party.

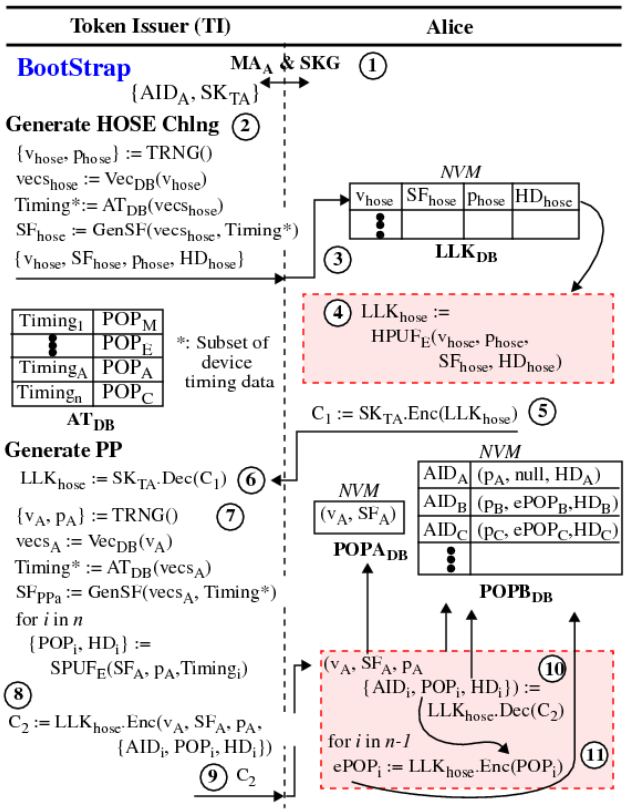

Fig. 6. PUF-Cash V3.0 bootstrap operations between the Token Issuer and Alice.

5.2 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 on-line, 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 of Alice's transactions with the *TI*, as well as transactions between the *FI* and *TI* are preceded with a PUF-based mutual authentication operation, annotated as MA_{NA} or MA_A for mutual authentication using the non-anonymous (NAT_{DB}) or anonymous (AT_{DB}) timing database, respectively, and with a PUF-based session key generation operation, annotated as *SKG*. The session key generation operation produces a shared key SK_x, where x is replaced with the initials of the authenticating parties, e.g., SK_{TF} refers to the shared session key between the *TI* and *FI*. Although the details of these strong, timing-database-oriented security functions are presented in previous work [2][7][1][5], the PUF-Cash challenge-response protocol operations related to the generation of PP_x utilize the same fundamental operations and are described herein for completeness.

5.2.1 PUF-Cash BootStrap. The message exchange diagram for Bootstrap is shown in Fig. 6. The message exchange and operations performed by Alice and the *TI* are described in a series of numbered steps, annotated with circles in the figure. We define the term hardware-obfuscated secure enclave (HOSE) to refer to the programmable logic region of Alice's device, which can

maintain confidential information, e.g., PUF keys, securely from the outside world and even the processor side of Alice's device. As discussed below, the HOSE is accessible only through a well-defined interface.

(1) **Generate HOSE Chlng**: Alice and the Token Issuer (TI) mutually authenticate and generate a shared session key, SK_{TA}. Mutual authentication is done anonymously, i.e., the TI uses its AT_{DB} which contains timing data obtained during provisioning in a secure facility. Upon successful authentication, the TI affirms that Alice's device is a genuine SiRF-instantiated PUF device, with anonymous ID AID_A. The AID_A is a unique record number associated with her timing data in the AT_{DB}.
(2) The TI generates random values $\{v_{hose}, p_{hose}\}$ for the generation of Alice's long-lived key LLK_A, to be used in her hardware security module $HOSE$ for encrypting POP and PUF-Cash data. v_{hose} is used to pseudo-randomly select a set of vectors $vecs$ from the Vec_{DB}, while p_{hose} is used to specify parameters for the SiRF algorithm. The anonymous timing data for a subset of devices $Timing*$ corresponding to paths timed by the selected vectors is extracted from the AT_{DB} and used to generate SpreadFactors SF_{hose}. The SF_{hose} optimizes uniqueness and randomness in the responses of all PUF devices that utilize the challenge. The SF_{hose}, p_{hose} and the timing data for Alice's device $Timing_A$ are used as input to a software version of the SiRF PUF enrollment algorithm to generate the LLK_A and helper data HD_A. The LLK_A is stored in the AT_{DB} as a field associated with her timing data record.
(3) The challenge components $\{v_{hose}, SF_{hose}, p_{hose}\}$ and helper data HD_{hose} are sent to Alice's device, which she stores in her LLK_{DB}. The LLK_{DB} stores long-lived challenges for Alice, which she uses for various functions in POP, PUF-Cash and other security related functions beyond those discussed here.
(4) Alice's $HOSE$ is tasked with regenerating the LLK_A for encrypting POP data received during the 'Generate PP' operation.
(5) **Generate PP**: The TI generates another set of random values $\{v_{PPa}, p_{PPa}\}$, which are used in the same fashion as described above in step 2.
(6) The TI generates a set of response bitstrings R_i and helper data bitstrings HD_i, one for each device i in the AT_{DB}, using the SF_{PPa}, p_{PPa} and $Timing_i$ as input to the SiRF PUF enrollment algorithm.
(7) The TI encrypts the challenge components, v_{PPa}, SF_{PPa}, p_{PPa}, and the set of data associated with the device responses, AID_i, R_i and HD_i, as C_1.
(8) The packet $C1$ is transmitted to Alice.
(9) Alice decrypts $C1$ and stores the population-based-components (PBC) of the challenge in the $POPA_{DB}$, which includes the v_{PPa} and SF_{PPa} elements of the packet. The customer-based-components, CBC, are stored in the $POPB_{DB}$ and include the AID_i, p_{PPa} and HD_i. Note that although the p_{PPa} component is identical for all elements in the $POPB_{DB}$, and can therefore be stored in $POPA_{DB}$, Alice will later replace this component with a new value after carrying out a payment transaction with a customer.
(10) The R_i component is first encrypted by the $HOSE$ before being stored in the $POPB_{DB}$, using Alice's long-lived key LLK_{hose} generated earlier. This protects the customer PUF response information stored in the POP_{DB} in the event an adversary gains access to Alice's device and attempts to extract information from the $POPB_{DB}$.

5.2.2 PUF-Cash Withdrawal. The message exchange diagram for the PUF-Cash Withdrawal operation is shown in Fig. 7. Alice carries out a withdrawal of eCash-tokens (eCt) on-line by contacting

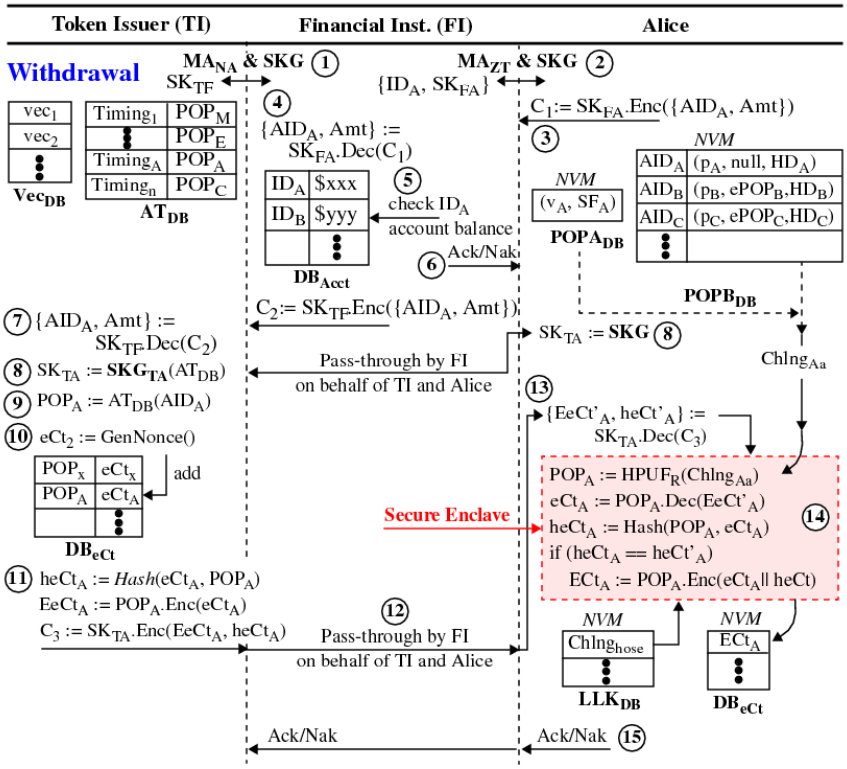

Fig. 7. PUF-Cash V3.0 message exchange for withdrawal operation between Alice, the Financial Institution and the Token Issuer.

her financial institution (FI). The FI commonly manifests as a commercial bank in our existing financial infrastructure, where Alice opens and maintains her Bank accounts, and therefore, interactions between Alice and the FI are not anonymous.

The token issuer (TI) represents the central authority and root-of-trust in the PUF-Cash system, and plays a key role in providing anonymity in eCt value transfer operations. The TI is responsible for carrying out two primary functions; it creates and later validates eCt for the FI and second, it serves as the root of security in the initiation and closure operations for the proposed propagation-of-provenance (POP) scheme.

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 use in value transfer operations with other entities, i.e., customers or commercial vendors, while providing provenance and protection against cloning and double spending.

(1)

6 PUF-CASH EXPERIMENTAL RESULTS

6.1 PUF-Cash Overhead Considerations

Similar to the MST model discussed earlier, the POP model requires Alice and Bob to store data about customers in the population, and given the potentially large customer base, the database needs to be compact.

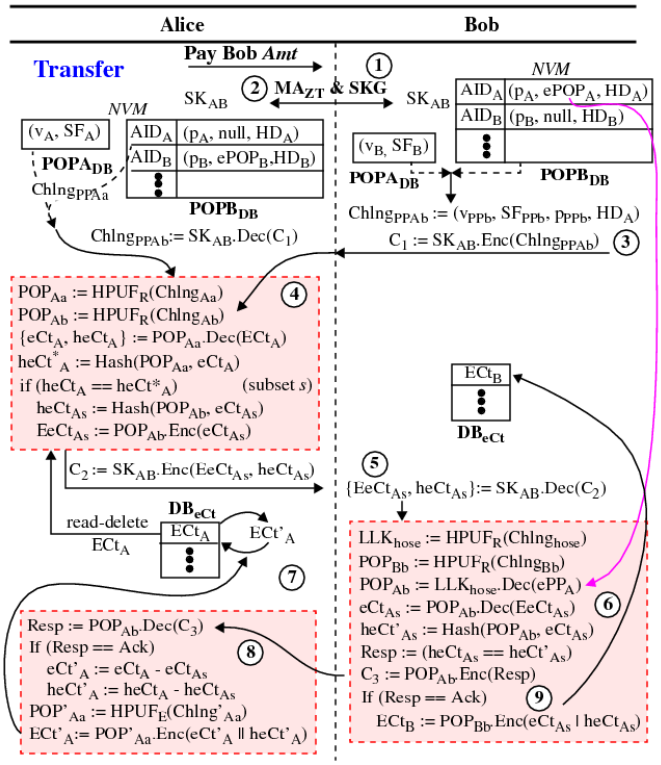

Fig. 8. PUF-Cash V3.0 message exchange for transfer operation between Alice and Bob.

7 PUF-CASH SECURITY ANALYSIS

8 CONCLUSIONS

REFERENCES

[1] 2021. *IC-Safety, LLC.* Retrieved Nov 10, 2021 from http:ic-safety.com

[2] J. Aarestad, P. Ortiz, D. Acharyya, and J. Plusquellic. 2013. HELP: A Hardware-Embedded Delay PUF. *IEEE Design and Test* 30, 2 (April 2013), 17–25.

[3] W. Che, M. Martinez-Ramon, F. Saqib, and J. Plusquellic. 2018. Delay model and machine learning exploration of a hardware-embedded delay PUF. In *Proc. IEEE International Symposium on Hardware Oriented Security and Trust (HOST).* 153–158.

[4] Derek Heeger and Jim Plusquellic. 2020. Analysis of IoT Authentication Over LoRa. In *2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS).* 458–465. https://doi.org/10.1109/DCOSS49796.2020.00078

[5] 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).*

[6] J. Ju, R. Chakraborty, C. Lamech, and J. Plusquellic. 2013. Stability analysis of a physical unclonable function based on metal resistance variations. In *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST).* 143–150. https://doi.org/10.1109/HST.2013.6581580

[7] Jim Plusquellic and Matt Areno. 2018. Correlation-Based Robust Authentication (Cobra) Using Helper Data Only. *Cryptography* 2, 3 (2018). https://doi.org/10.3390/cryptography2030021

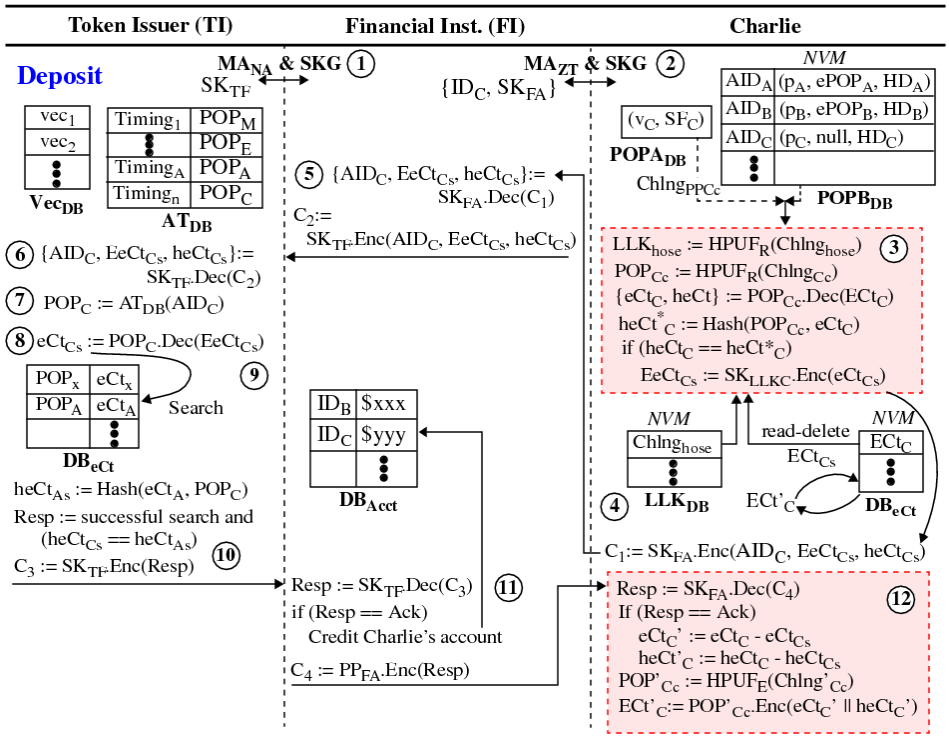

Fig. 9. PUF-Cash V3.0 message exchange for deposit operation between Charlie, the Financial Institution and the Token Issuer.