*cryptography*

MDPI

*Article*

# Physical Unclonable Function (PUF)-Based e-Cash Transaction Protocol (PUF-Cash)

**Jeff Calhoun [1], Cyrus Minwalla [2], Charles Helmich [3], Fareena Saqib [4], Wenjie Che [5] and Jim Plusquellic [6,*]**

[1] University of New Mexico, 211 Terrace St NE, Albuquerque, NM, 87131
[2] Bank of Canada, 234 Wellington St. Ottawa, ON. K1A 0G9
[3] University of New Mexico, 211 Terrace St NE, Albuquerque, NM, 87131
[4] University of North Carolina, 9201 University City Blvd, Charlotte, NC, 28223
[5] New Mexico State University, 1780 E University Ave, Las Cruces, NM, 88003
[6] University of New Mexico, 211 Terrace St NE, Albuquerque, NM, 87131
* Correspondence: jimp@ece.unm.edu; Tel.: +1-240-475-1882

Received: 6 June 2019; Accepted: 16 July 2019; Published: date

**Abstract:** Electronic money (e-money or e-Cash) is the digital representation of physical banknotes augmented by added use cases of online and remote payments. This paper presents a novel, anonymous e-money transaction protocol, built based on physical unclonable functions (PUFs), titled PUF-Cash. PUF-Cash preserves user anonymity while enabling both offline and online transaction capability. The PUF's privacy-preserving property is leveraged to create blinded tokens for transaction anonymity while its hardware-based challenge–response pair authentication scheme provides a secure solution that is impervious to typical protocol attacks. The scheme is inspired from Chaum's Digicash work in the 1980s and subsequent improvements. Unlike Chaum's scheme, which relies on Rivest, Shamir and Adlemans's (RSA's) multiplicative homomorphic property to provide anonymity, the anonymity scheme proposed in this paper leverages the random and unique statistical properties of synthesized integrated circuits. PUF-Cash is implemented and demonstrated using a set of Xilinx Zynq Field Programmable Gate Arrays (FPGAs). Experimental results suggest that the hardware footprint of the solution is small, and the transaction rate is suitable for large-scale applications. An in-depth security analysis suggests that the solution possesses excellent statistical qualities in the generated authentication and encryption keys, and it is robust against a variety of attack vectors including model-building, impersonation, and side-channel variants.

**Keywords:** electronic cash; cryptographic protocol; physical unclonable function (PUF)

## 1. Introduction

The global economic engine relies on the safe and efficient transfer of value for goods received and services rendered. Electronic means of payments have spawned a hive of economic and technical activity across the world. The e-commerce landscape is profligate with electronic payment (e-payment) systems, ranging from solutions by dedicated payment processors (Visa, Mastercard) to technology partnerships (ApplePay, Google Wallet, Paypal) to new entrants in the financial services business (Square, AliPay, Tencent). Representation of value varies from intrasystem accounts to credit transfers between institutions. In all such mechanisms, the value representation supersedes privacy considerations. Moreover, the providers are incentivized to limit bilateral peer-to-peer (P2P) transfers as it interferes directly with their business model.

Schemes based on distributed ledger technology (DLT) enable P2P payments in a decentralized and pseudo-anonymous fashion. As such, they are particularly popular in academia and industry.

However, the consensus mechanism of DLT systems places practical limits on transaction rate, scalability, and user participation. In addition, fully decentralized systems lack remediation and nonrepudiation capabilities; these mechanisms are commonly offered by conventional payment platforms to protect consumers against fraud, malice, and unscrupulous merchant behavior. Finally, entry into a DLT-based system is nontrivial. For most users lacking detailed technical knowledge, entry is mediated by exchanges, where legitimate services are required to comply with Know Your Customer (KYC) and Anti-Money Laundering (AML) legislation, reducing or outright eliminating the pseudo-anonymity inherent in DLT systems, while illegitimate exchanges may act maliciously and with impunity.

Electronic money (e-money) schemes based on privacy-preserving components such as blind signatures and zero-knowledge proofs can preserve anonymity in online payments and P2P transactions. The e-money scheme proposed herein is unique in its reliance on hardware physical unclonable functions (PUFs). The chief benefit of PUFs in the e-money scheme is the preservation of user privacy from the Original Equipment Manufacturers (OEMs) and suppliers of physical devices, preserving trust in the supply chain and enforcing the user's right to determine the trust authority post-factum.

Authentication, encryption, privacy, and anonymity play central roles in e-money. PUFs enable a challenge–response pair (CRP) form of authentication, which, by itself, vastly improves on weak user password methods of authentication. Although a wide range of PUF architectures have been proposed since their introduction in 2002 [1], most are characterized as having a relatively small CRP space. Strong PUFs, on the other hand, possess a CRP space that is exponential, making them better suited for both authentication and encryption key generation roles. Among the PUF architectures proposed, several have been used in privacy-preserving authentication protocols. However, to our knowledge, no PUF-based e-money protocols or schemes capable of providing anonymity have been proposed to date.

In this paper, we present an e-money transaction protocol that leverages a previously proposed hardware embedded delay PUF called Hardware Embedded Delay PUF termed HELP [2] and previously developed authentication protocol [3] that have been suitably modified for the present application. HELP exhibits several desirable characteristics, including an exponential CRP space and model building resistance, and it can be configured to enable both privacy and anonymity. The highly unique statistical characteristics of the authentication bit-strings generated by HELP enable privacy as described in [3].

HELP's privacy preserving characteristic is extended here to allow a trusted authority to confirm that a transaction is valid without revealing the identity of the authenticating customer. The first solution to this important e-money security property was proposed by Chaum [4] utilizing RSA's multiplicative homomorphic property to generate blind signatures. The PUF-based solution presented in this paper, on the other hand, uses the intrinsic entropy available in the PUF's bit-strings to implement anonymity. PUF-Cash leverages a trusted third party (TTP) to handle fielded chip anonymization and dispute resolution, thus providing privacy and protection between customers in the protocol's message exchanges.

The proposed protocol relies on Exclusive OR (XOR), symmetric encryption (128-bit Advanced Encryption Standard (AES)), and Diffie–Hellman key exchange (DHE) to remain as lightweight as possible. This enables the final device to be implemented in a low power, embedded hardware token format that is mobile and/or wearable. Although the proposed system has limitations (i.e., the system is restricted to unitary tokens and transitivity is limited), it is shown to possess all the required security properties of e-Cash.

The specific contributions of this paper over the current state-of-the-art are given as follows:

- PUF-Cash is the first e-Cash protocol proposed where authentication bit-strings, encryption keys, and e-Cash tokens are based on random physical processes, thereby making it difficult for adversaries to falsely authenticate or break into fielded devices to steal, predict, and/or clone tokens.

- A novel, anonymous enrollment database scheme is proposed to blind transfers between Alice and Bob, thereby protecting anonymity. In addition, the protocol preserves privacy during authentication and in all subsequent message exchanges between customers and trusted parties, preventing adversaries from tracking user transactions.
- A novel PUF-based session key generation technique is proposed that provides high levels of reliability across environmental temperature and supply voltage variations.
- Transactions are bilateral, allowing customers to transfer value locally without requiring connectivity to third parties or any manner of online capability.
- All transaction processing operations are constant time or linear, enabling fast response times, high performance, and low power consumption on the fielded devices, the trusted third party (TTP), and bank servers.

The remaining components of this paper are organized as follows. Section 2 provides a literature review. Section 3 presents an overview of HELP, focusing on its authentication and bit-string generation processes. Section 4 describes the proposed e-Cash protocol, while Section 5 investigates its security properties. Section 6 presents experimental results, and conclusions are drawn in Section 7.

## 2. Literature Review

### *2.1. Electronic Cash*

Any electronic version of cash (e-Cash) must contend with two primary problems of fiat currency, namely counterfeiting and double-spending. Modern e-Cash schemes incorporate cryptographic tools such as digital signatures, secret sharing, and zero-knowledge proofs to mitigate the risk of both actions.

The very first e-Cash system was devised by Chaum [4,5] and refined into a complete transfer protocol by Chaum, Fiat, and Naor (CFN) [6]. It relied on blind signatures to preserve anonymity and a cut-and-choose shared secret for the coin exchange. The blind signature scheme is based on RSA polynomial primitives, and it reveals the identity of the user on double-spend, thus detecting but not preventing double-spending. A CFN e-coin consumes O(k2) bits of memory, where k is the security parameter. Batch RSA techniques to improve transaction efficiency were explored by Schoenmaker [7], but it was Brands' e-Cash scheme [8] that presented a first dramatic improvement over CFN. Brands used groups of prime order for the blind signature generation and replaced cut-and-choose proofs with a more efficient scheme based on the Sigma Protocol [9].

The original Chaumian e-Cash and variants, as conceived, represented indivisible e-coins. An important breakthrough was achieved by Camenisch, Hohenberger, and Lysanskaya (CHL) in developing divisible e-Cash [10,11]. In CHL e-Cash, a pseudo-random function could generate serial numbers from a single seed, provided that the user could prove to the merchant that the value $w$ was within the finite range  as supplied by the Bank. The range proof requires Fujisaki–Okamoto commitments [11] and consumes O(k + log W) space, where W-1 is the upper bound on w. CHL e-Cash uses interpolation over a straight line in the exponent to catch double-spending. It also embeds a non-interactive zero-knowledge proof of the user's secret key in each e-coin to prevent the bank from framing the user. Okamato also developed a divisible e-Cash scheme using his blind signature commit mechanism, although e-coins can only be spent in powers of two [12].

Recent advances in quantum hard cryptography have also made it possible to implement blind signatures with lattice-based asymmetric key exchanges [13]. A hardware implementation of CHL e-Cash was developed for mobile devices utilizing the ARM TrustZone trusted computing environment [14]. Use of hardware electronic wallets in cell phones or mobile devices for electronic transactions is described by Sakalauskas [15].

### *2.2. Physical Unclonable Function (PUF)-Based Authentication and Key Generation*

Physical unclonable functions (PUFs) are hardware security primitives capable of generating long sequences of random, but reproducible, bit-strings on-demand and without requiring device secrets to be stored in any type of nonvolatile memory (NVM) [1]. Regeneration of secret keys and bit-strings can be undertaken by a fielded chip under adverse environmental conditions.

Unlike cryptographically secure hash functions, which leverage the infeasibility of computing inverses for special functions, a PUF's strength is derived from the random behavior of physical processes. Integrated circuits containing a PUF are referred to as devices. Although modern semiconductor manufacturing facilities are optimized to produce exact replicas of the devices (chips), tolerances in the manufacturing process are nonzero, and small levels of process variations are always present. These process variations change the signal behavior of the chip in subtle ways (e.g., the delay of a signal propagating through a combinational logic block is somewhat different from one copy of the chip to another). The PUF architecture naturally includes a source of entropy (i.e., a set of chip signals that randomly vary within a limit), a method to measure them, and, in some cases, to create digital values that represent their magnitude. The randomness of signal variations introduced by process variation effects creates a strong, and sometimes very large, random sample space. The keys and bit-strings produced by the PUF are, therefore, very difficult, if not impossible, to predict.

PUF applications, such as authentication, typically expose the generated bit-strings to the outside world, which in turn makes the PUF more vulnerable to attacks. Statistical and/or machine learning systems can attempt to build a model of the PUF's challenge-response behavior based on successive queries. Such methods have been demonstrated to be successful [16]. However, as PUF architectures evolve to increased complexity, with built-in countermeasures to model-building attacks, such attacks become increasingly difficult to execute.

## 3. HELP and Authentication Protocols

Selected components of HELP and the associated authentication and session key generation protocols are discussed in this section. Additional details are available in previous work [3,17]. In the following, we use the terms HELP to refer to the entire PUF architecture, HELP algorithm to refer to the post-processing operations carried out to generate authentication bit-strings and keys, and HELP protocol in reference to the operations and messages exchanged between the device and server.

### *3.1. PUF Architecture and Soft Data*

HELP generates secret keys and bit-strings from variations that occur in the delay of signals propagating along paths in combinational logic circuits, such as those shown in the schematic of Figure 1. A launch–capture timing procedure, using two clocks, $Clk_1$ and $Clk_2$, and associated flip-flops (FFs), provides high-resolution digitized delay values for a set of paths. The term PUF Numbers, or PN, is used as a generic reference to digital numbers that capture the magnitude of a measured signal, such as propagation delay. PN define a class of PUF data referred to as soft data, which is heavily leveraged in the proposed PUF-Cash protocol.

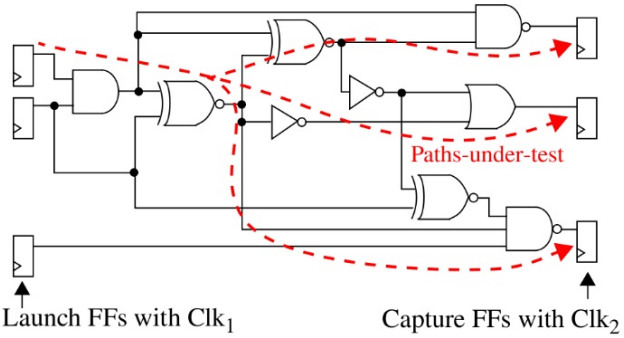


**Figure 1.** Launch-capture timing of paths through combinational logic used by HELP.

### *3.2. Enrollment and Regeneration*

PUFs are well-suited for a challenge–response pair (CRP) authentication sequence, where the challenge and response are defined as two bit-strings of '0's and '1's. PUF-based authentication protocols are characterized as having an enrollment and authentication phase. During enrollment, a trusted authority registers a new PUF by recording the CRPs for the fielded chip in a secure database. This enrollment database is constructed with a relatively small set of CRPs sampled from a much larger (exponential) CRP space. PUFs that provide an exponential space naturally (i.e., without leveraging keyed hash functions to expand the CRP space) are called strong PUFs.

During regeneration, the fielded device supplies credentials to a trusted authority for authentication. Here, the PUF is tasked with regenerating a subset of the responses stored during enrollment, which are then compared with the enrollment responses by the authenticating agent. The regenerated bit-strings commonly have bit-flip errors. Some authentication protocols allow small numbers of bit-flip errors to occur, while others employ error correction or error avoidance techniques to correct or avoid them, respectively. HELP, in particular, relies on an error avoidance scheme employed during response verification.

The enrollment process used by HELP differs from the prior description in the following manner: instead of storing CRPs in the secure database, HELP stores PN, the digitized soft data values associated with the measured path delays. A sample database is shown in Figure 2 where the rows store data for each fielded chip and the columns store digitized rising (PNR) and falling (PNF) path delays as $n$-bit fixed-point numbers, typically ranging in value between 100 and 500.

| Fielded device name | $PNR_0$ | $PNR_1$ | $PNR_2$ | ••• | $PNR_x$ | $PNF_0$ | $PNF_1$ | $PNF_2$ | ••• | $PNF_x$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $C_1$ | 380.1 | 294.8 | 366.9 | ••• | 288.0 | 364.0 | 328.0 | 328.0 | ••• | 276.2 |
| $C_2$ | 366.6 | 282.8 | 352.7 | | 278.6 | 374.3 | 334.6 | 337.1 | | 286.1 |
| $C_3$ | 366.3 | 288.4 | 355.7 | | 280.8 | 372.7 | 336.4 | 338.0 | | 282.3 |
| | ⋮ | | | | | | | | | |
| $C_n$ | 387.5 | 301.2 | 373.5 | | 292.3 | 362.9 | 325.18 | 323.7 | | 272.1 |

**Secure Server Database (DB)**

**Figure 2.** HELP PN database (DB) created during enrollment.

There are several significant benefits to storing PN instead of response bit-strings [18–23]. First, the relationship of PN to response bit-strings is one-to-many (i.e., a large number of response bit-strings can be generated from a much smaller set of PN). Second, HELP generates helper data during regeneration, which is ancillary data designed to assist the PUF architecture in precisely reproducing keys and bit-strings. The PN stored in the database allow the server to generate its own helper data by running a software version of the HELP algorithm. A novel error avoidance scheme that leverages both the fielded device and server-generated helper data provides significant improvements against bit-flip errors. Third, a novel correlation-based authentication method can allow a server to authenticate a fielded device by comparing the device and server-generated helper data only, thereby completely eliminating the need to exchange response bit-strings, which in turn, makes the process of building a predictive model that clones the device much more difficult, if not outright impossible.

The key generation and session authentications algorithms are designed to be fast, low-energy, and reliable. These goals are achieved by using only linear operations to process the stored and regenerated PN into session keys and authentication bit-strings. Other PUFs [1,18–23] can be used instead of HELP to meet these goals as long as they possess the following properties:

- The PUF is a strong PUF with an exponential search space for both the input challenges and response bit-strings.

- The PUF generates soft data that can be stored in the enrollment database (e.g., Static Random Access Memory(SRAM) PUFs are excluded).
- The PUF is capable of generating and utilizing an anonymous enrollment database.
- The soft data (PN) associated with the PUF allows a fast, linear transformation to be applied to correct the PN for changes induced by environmental temperature and/or supply voltage variations, which, in turn, enables fast bit-flip avoidance techniques to be used to provide reliable authentication bit-strings and session keys. The time consumed by complex error correction methods, although suitable for one-off authentication and session key generation, are not suitable here where short transactions times are critical.

### *3.3. Bit-String Generation*

Authentication and key generation between the fielded device and the central authority ("the bank") is accomplished by relying on three techniques: Cobra [17], for correlation-based robust authentication; DHD_Authen [3], for dual-helper-data authentication; and DHD_Key, for dual-helper-data key generation. Each of these techniques uses a bit-flip error avoidance scheme to establish trust and provide encrypted communications between the device and the bank. The error avoidance scheme is designed to improve the reproducibility of authentication bit-strings and keys on the server and devices, but it additionally acts as a natural source of randomness, improving privacy and protection against machine learning based attacks.

Error avoidance is a multilayered approach, with Figure 3a illustrating concepts at the lower layer. The graph plots a subset of 18 path delays (PN) measured by device $C_1$ during enrollment (black) and regeneration (red) along the x-axis. The PN shown here have already been processed to reduce adverse temperature–voltage variations and path length bias using procedures defined by the HELP algorithm. The objectives of these procedures are to make the black and red data points similar in value (i.e., to make the path delays independent of the temperature and supply voltage conditions under which they are measured), and to ensure that all PN are bounded by a fixed range, given in the graph as 0 to 20. Reference [3] provides details on these preprocessing components of the HELP algorithm.

The graph and annotations in Figure 3 illustrate the process implemented within the HELP algorithm to discretize the floating point PN into a response bit-string, called a single-strong-bit-string (SSBS), and a helper data bit-string, referred to as single-helper-data (SHD). Here, a PN is assigned a bit value of '0' in the SSBS if its value is smaller than 10 and a '1' otherwise. The '0' and '1' regions are each further partitioned into strong and weak regions for generating the SHD. For example, the regions between 0–2, 8–12, and 18-20, inclusive, are classified as 'weak', while the regions between 2–8 and between 12–18 are classified as 'strong'.

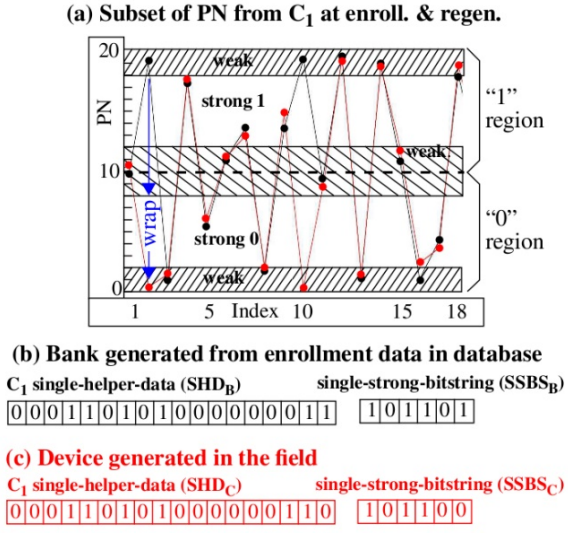

**Figure 3.** (**a**) PN for chip $C_1$ measured during enrollment (black) and regeneration (red), (**b**) Single-helper-data ($SHD_B$) and single-strong-bit-string ($SSBS_B$) generated from enrollment data stored in a secure database, and (**c**) $SHD_C$ and $SSBS_C$ generated by the device in the field during regeneration.

The term 'weak' is used because PN that fall within this region are close to 0, 10, or 20. The values 0, 10, and 20 are referred to as bit-flip lines because the bit value assigned to the SSBS changes value on either side of these lines. For example, PN above 10 are assigned '1' while those below this line are assigned '0'. Similar behavior occurs at 0 and 20 because the HELP algorithm applies a modulus to the PN, which wraps values into the opposing '0' and '1' regions.

Given these bit-flip lines, the key observation is that PN close to a bit-flip line are less reliable when regenerated in the field, due to noise, which can cause the regenerated PN to be slightly different than the enrollment PN (noise is apparent in every black–red pairing of PN in the graph). Therefore, PN close to the bit flip lines can produce different response bits in the black server-generated SSBS and in the red device-regenerated SSBS. In contrast, the strong regions provide higher resilience to bit-flip errors because the noise levels must be much larger for a bit-flip error to occur.

The SHD and SSBS bit-strings in Figure 3b,c are constructed using the data from the graph in Figure 3a. The terms $SHD_B$ and $SSBS_B$ as well as $SHD_C$ and $SSBS_C$ are used to refer to bank (server) and chip regenerated bit-strings, respectively. Note that the SSBS are shorter in length than the SHD because only PN classified as strong by the SHD bit-string are included in the SSBS bit-string.

The term 'single' is used in reference to the bit-strings shown in Figure 3b,c because the algorithm uses only server-generated PN or only device-regenerated PN to construct the bit-strings. As we will show, a more resilient bit-flip avoidance scheme can be constructed when all four bit-strings are used together. The technique is referred to as the dual-helper-data (DHD) scheme, as described below.

### *3.4. Device Authentication Using Cobra*

The server (bank) authenticates the device using a privacy-preserving technique called Cobra. Cobra uses only the SHD generated by the device ($SHD_C$) for authentication (i.e., the device-generated $SSBS_C$ is not transmitted to the server). The process is illustrated in Figure 4a showing the fielded chip $SHD_C$ in red, which is transmitted to the bank, and the $SHD_{Bi}$ computed by the bank using PN stored in its enrollment database. The bank pairs the $SHD_C$ with each of the $SHD_{Bi}$ and then computes a correlation coefficient ($CC_i$) using the bitwise Exclusive Not-OR (XNOR) operator as:

$$CC_i = \sum_{j=1}^{2048} \left( \overline{SHD_{Cj} \oplus SHD_{Bij}} \right)$$

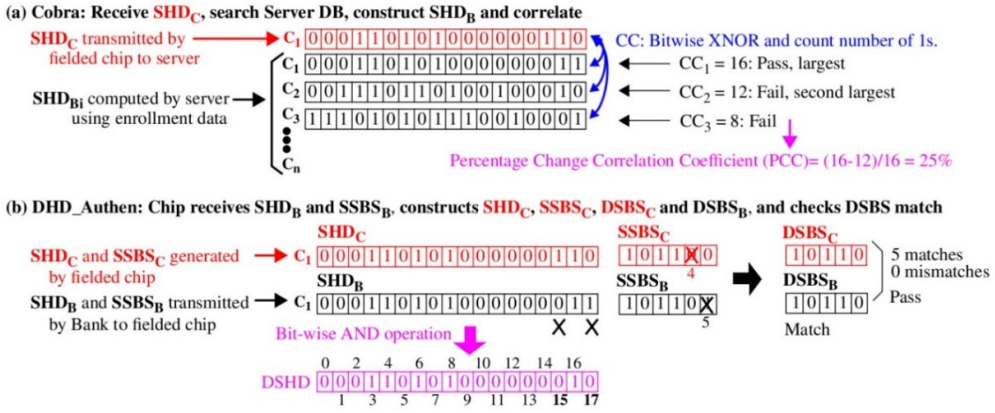


**Figure 4.** Bit-string construction algorithms (**a**) Cobra for device authentication and (**b**) DHD_Authen for server authentication.

The $CC_i$ simply counts the number of bits that match in the two bit-strings. The two largest $CC_i$s, $x$ and $y$, are selected and used to compute a percentage-change correlation coefficient (PCC) as:

$$\text{PCC} = \frac{CC_X - CC_Y}{CC_X} \times 100$$

If the PCC exceeds a fixed threshold (e.g., 10%), then the chip ID from the enrollment database with the largest CC is treated as the authenticating chip and authentication succeeds. Typical values for the PCC in our hardware experiments are 25%, with the largest CC exceeding 1800 matching bits.

The technique proposed for Cobra works because the bank-generated and device-generated SSBS 'track' each other; therefore, the classification of the SSBS bits as weak and strong in the $SHD_C$ and $SHD_B$ are highly correlated. For example, the same classification of weak and strong bits occurs 16 times for the 18 PN shown in Figure 3. In contrast, the PN produced by other chips $C_i$ are impacted by within-die variations (and noise), which reduces the correlation between the $SHD_C$ and their $SHD_{Bi}$ significantly.

### *3.5. Bank Authentication Using DHD_Authen*

The successful completion of device authentication both authenticates and identifies the device to the server. Once completed, the device then carries out a server authentication operation using the DHD_Authen process shown in Figure 4b. Unlike Cobra, server authentication requires both the $SHD_B$ and $SSBS_B$ to be transmitted to the device. Note that even though the response bit-string ($SSBS_B$) is revealed to the adversary in this case, it is not possible to model-build the device using the server enrollment data for three reasons. First, the adversary must first succeed with device authentication before server authentication is performed. Second, the server can detect attempts by adversaries to repeatedly request $SSBS_B$ for a specific chip and refuse to interact with the adversary. And third, previous work shows that model-building is not possible even when the adversary has unrestricted access to collect SSBS from the device [24]. So, it follows that a successful attack using a limited number of $SSBS_B$ provided by the bank would be highly unlikely.

From Figure 4b, the DHD_Authen fetches the $SHD_B$ and $SSBS_B$ from the bank, as well as the challenges (not shown). These bit-strings are constructed by the bank using a software version of the HELP algorithm applied to the enrollment data for $C_1$. The device generates its own version of these bit-strings, $SHD_C$ and $SSBS_C$, by applying the challenges to its hardware PUF. With all four bit-strings available, the device then executes the dual-helper-data (DHD) algorithm. DHD first bitwise ANDs the $SHD_C$ and $SHD_B$ bit-strings, which is shown by the DSHD bit-string along the bottom of Figure 4b. Then, two DSBS bit-strings are constructed from the SSBS bit-strings by eliminating SSBS bits where the corresponding SHD bits change from 1 to 0 in the DSHD bit-string. This happens for DSHD bits at positions 15 and 17. The $SHD_C$ bit at position 15 is a 1, which means the $SSBS_C$ includes a corresponding strong bit, which must be eliminated. The deletion of this bit is indicated with the 'X' over the $SSBS_C$ bit at position 4. Similarly, the DSHD bit at position 17 indicates the $SSBS_B$ at position 5 needs to be eliminated. The final versions of the $DSBS_C$ and $DSBS_B$ bit-strings are shown on the far right in Figure 4b. The DSBS bit-strings are then compared, and if they match, authentication succeeds.

The dual-helper-data algorithm significantly reduces the probability of the bit flip error between the $DSBS_C$ and $DSBS_B$ bit-strings because errors can only occur now when both the fielded device and server classify the bit as strong. From Figure 3a, this is only possible when noise that occurs during regeneration shifts the red PN by more than twice the width of the weak region (not the same width, as is true of the single-helper-data scheme). The improvement in reliability provided by the DHD algorithm allows the device to set a pass-fail threshold as shown, where all bits in the two DSBS bit-strings must match exactly.

### *3.6. Session Key Generation Using DHD_Key*

In cases where the device and bank need to exchange encrypted data, a novel algorithm, called DHD_Key, can be used to create a shared secret key. This process is carried out only after both the device and bank have successfully authenticated. The DHD_Key protocol also leverages the DHD

algorithm, as described for the DHD_Authen process, but with two fundamental differences. First, the bank sends only the $SHD_B$ to the device (i.e., the $SSBS_B$ is not transmitted). Second, the device sends its $SHD_C$ to the bank so it too can carry out the DHD algorithm.

A graphic illustration of the process is shown in Figure 5. First, the bank sends challenges to the device (not shown), and both generate and exchange their $SHD_B$ and $SHD_C$ bit-strings. Then each side calculates the bitwise AND of the SHD bit-strings, eliminating bits in their corresponding SSBS bit-strings to produce the $DSBS_B$ and $DSBS_C$ bit-strings. As discussed above, the DHD algorithm significantly increases the probability that the two independently-generated DSBS bit-strings are identical. In such cases, these bit-strings can be used as a session key by, for example, the AES encryption algorithm, to encrypt communication between the device and bank.

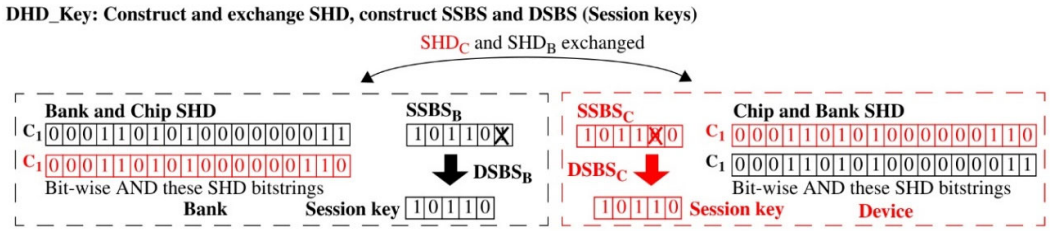


**Figure 5.** Session key construction process implemented by the DHD_Key algorithm.

## 4. PUF-Based e-Cash Transaction Protocol (PUF-Cash)

The PUF-Cash protocol consists of four entities: Alice, Bob, a trusted third party (TTP), and the bank. The protocol extends HELP protocols described above in multiple, distinct ways. First, a special enrollment process generates two timing databases (DBs), one which labels the timing data collected from the device with the owner's name (e.g., Alice and Bob) or $C_x$ as given by the leftmost column of Figure 2, and a second anonymous DB that stores timing data sets where the rows from Figure 2 are randomly permuted and stored without the identifying labels. Note that enrollment is done at a trusted facility by the device manufacturer or an independent third-party contractor after the devices are manufactured and not at the bank. The anonymous DB enables transactions with the bank to be blinded, which is in the spirit of Chaum's scheme [4,5]. Here, the original two-party protocol is extended to four parties, namely, Alice, Bob, the bank, and a trusted third party (TTP).The message exchange protocol contains multiple steps to facilitate a secure, potentially offline, payment transaction between Alice and Bob. . XOR encryption is employed, wherever safe to do so, to maintain confidentiality while minimizing computational complexity.

The messages exchanged during the enrollment process for PUF-Cash are shown in Figure 6 where the terms $NAT_{DB}$ and $\{c_{k1}\}$ are used in reference to the non-anonymous DB, and $AT_{DB}$ and $\{c_{k2}\}$ are used for the anonymous DB. The challenge set $\{c_k\}$ is partitioned into two subsets $\{c_{k1}\}$ and $\{c_{k2}\}$. The challenges in $\{c_{k1}\}$ are chosen randomly from the CRP space defined for HELP. One half of the challenge set is common, and applied to all devices, while subsets are randomly chosen for each device from the other half. Note that this partitioning of $\{c_{k1}\}$ is not explicitly depicted in Figure 6. The common set of challenges enables privacy between Alice, Bob, and the bank because the device does not identify itself in the message exchange during authentication. Instead, the Cobra algorithm performed at the bank carries out a search process using enrollment data that is associated with the common challenges to authenticate and identify the device as part of the trusted set. Once identified, the bank can select device-specific challenges for the bank authentication and session key generation processes as needed. The set $\{c_{k2}\}$ is also chosen randomly but is common for all devices. The bank applies a search here as well, which requires all enrolled devices in the $AT_{DB}$ to store PN for these common challenges. Both the $NAT_{DB}$ and $AT_{DB}$ are transferred from the trusted enrollment facility to the bank, and both are protected as sensitive data.

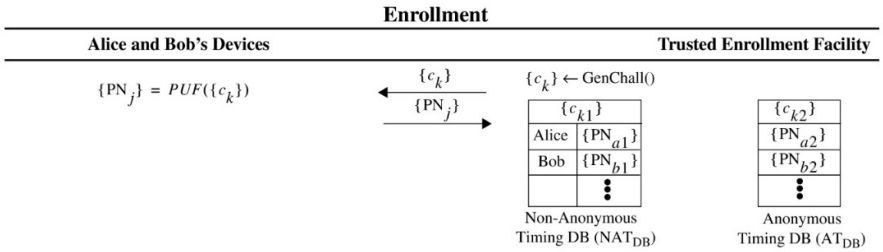


**Figure 6.** Physical unclonable function (PUF)-Cash protocol enrollment process.

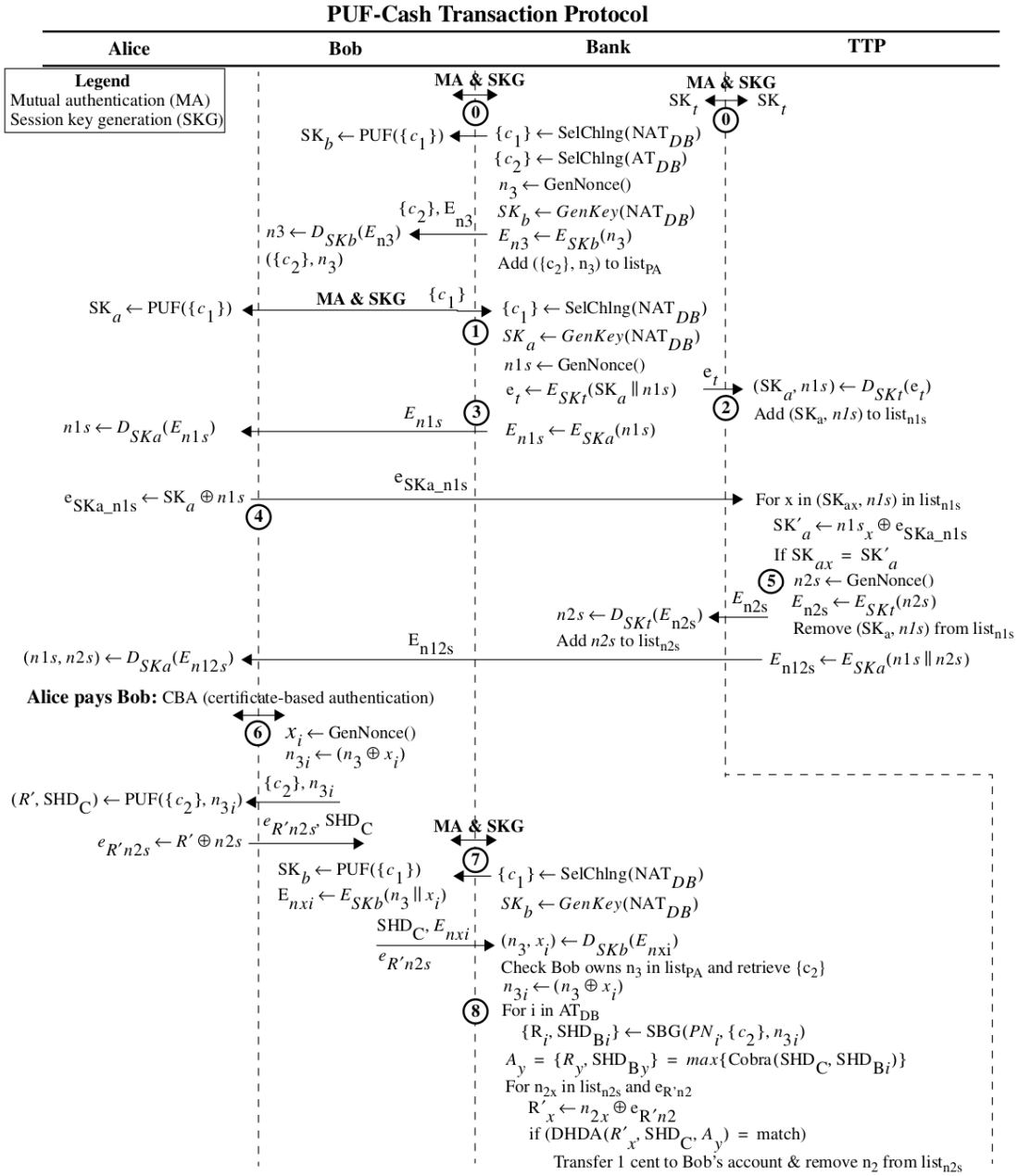


**Figure 7.** The PUF-Cash transaction protocol.

Figures 7 and 8 depict the sequence of messages and actions carried out in a single transaction by the protocol, with the formal cryptographic notation employed in Figure 7 and the major elements of the protocol in Figure 8. The following sequence of steps refer to the corresponding annotations in the figures. We use the acronyms MA to refer to mutual authentication and SKG to refer to session key generation. Note that all invocations of HELP for key generation on the device, denoted as PUF() in Figure 7, and the corresponding calls to *GenKey*() by the bank take a nonce as a parameter. However, the nonce parameter is omitted in most cases to avoid confusion with other nonces used in the protocol.

- **Step 0 (Enrollment)**: The bank and TTP mutually authenticate using the HELP algorithms Cobra and DHD_Authen, and then create a 128-bit AES session key $SK_t$ using DHD_Key. Here, we assume the TTP also contains an instance of HELP and has enrollment data stored in the bank's NAT database. Note that each of the device and server authentication operations within mutual authentication, and the session key generation, use distinct sets of challenges. This process is performed only once at startup using challenges from the $NAT_{DB}$.
  Alice and Bob also receive preauthorization information from the bank during enrollment, and later periodically in the field during withdrawal requests (only Bob's transaction is annotated in Figure 7). Preauthorization information is later used to cryptographically bind Alice's e-Cash tokens to Bob when Alice pays Bob for a product or service. Alice and Bob first carry out mutual authentication (MA) and session key generation (SKG) with the bank. The bank generates preauthorization tuples ($\{c_2\}$, $n_3$) uniquely for Alice and Bob, with the challenges $\{c_2\}$ drawn from those stored in the anonymous database ($AT_{DB}$). The bank stores the tuples ($\{c_2\}$, $n_3$) in $list_{PA}$ along with the owner's information. The bank then encrypts and transmits the ($\{c_2\}$, $n_3$) tuples to Alice and Bob, who then decrypt them using their session keys. Alice and Bob store these preauthorization tuples securely on their devices for future transactions. Note that Step 0 only occurs once or aperiodically as enrollment is updated.
- **Step 1**: An e-Cash transfer begins with Alice interacting with her bank to withdraw funds. Alice has a device containing an instance of HELP that has been enrolled with the bank, and is also now capable carrying out mutual authentication and session key generation with the bank. The challenges $\{c_1\}$ sent by the bank refer to the generation of a shared session key $SK_a$ between Alice and the bank (as noted, two different sets of challenges not shown are used for the proceeding MA operations). Alice generates a session key $SK_a$ by applying $\{c_1\}$ to her PUF, while the bank generates its copy of $SK_a$ by applying Alice's enrollment data ($PN_a$) stored in the $NAT_{DB}$ to a software version of the HELP algorithm. Alice then requests a withdrawal of $1 from her account.
- **Step 2**: In response to Alice's request, the bank obtains a set of, (e.g., 100 $n1$) nonces from a TRNG, then AES encrypts Alice's $SK_a$ and $n1s$ as $E_t = E_{SKt}(SK_a || n1s)$ and transmits $E_t$ to the TTP. Note that $SK_t$ is a prenegotiated symmetric key between the bank and TTP from Step 0. The TTP decrypts $E_t$ as ($SK_a$, $n1s$) = $D_{SKt}(E_t)$ and stores the pair ($SK_a$, $n1s$) in a list, $list_{n1s}$.
- **Step 3**: Once the bank completes the transaction with the TTP, it AES encrypts the $n1s$ as $E_{n1} = E_{SKa}(n1s)$ and then transmits them to Alice. Alice decrypts $n1s = D_{SKa}(E_{n1s})$ using her $SK_a$. The bank subtracts $1 from her account and adds the corresponding $n_1$ to a floating e-Cash pool (not shown). The $n_1$ nonces will later enable the TTP to validate that Alice has transacted with the bank.
- **Step 4**: When Alice wants to blind her token for payment, she XOR encrypts her $SK_a$ and $n1s$ as $e_{SKa\_n1s}$ and transmits them to the TTP. The TTP XOR decrypts $e_{SKa\_n1s}$ using each of the $n1s_x$ elements in $list_{n1s}$ and compares the recovered $SK'_a$ with each of the corresponding $SK_{ax}$ session keys from the pairs ($SK_{ax}$, $n1s_x$) pairs stored in $list_{n1s}$. The TTP removes the matching ($SK_{ax}$, $n1s_x$) from $list_{n1s}$. Note that finding a match also authenticates Alice to the TTP and validates her as having transacted earlier with the bank.
- **Step 5**: The TTP generates a set of corresponding $n2s$ using a TRNG, AES encrypts the $n2s$ with $SK_t$ as $E_{n2s} = E_{SKt}(n2s)$, and transmits them to the bank. The bank decrypts the $n2s$ and stores them in a list of open transactions, identified as $list_{n2s}$. The TTP also AES encrypts both the $n1s$ and $n2s$

with $SK_a$ and transmits the $E_{n12s}$ to Alice. Alice recovers the $n1s$ and $n2s$ using her copy of $SK_a$ as $(n1s, n2s) = D_{SKa}(E_{n12s})$. Alice validates the $n1s$ recovered are the same as the originals she holds. Once this step is complete, Alice's e-Cash tokens are blinded and ready for payment.

- **Step 6**: Alice wants to pay Bob $1. First, Alice establishes a secure channel with Bob using elliptic curve Diffie–Hellman ephemeral with RSA (ECDHE-RSA). This protocol is part of the TLS 1.2 standard, and an official implementation exists in OpenSSL since v1.0.1. Once a secure channel is established, Bob then generates a nonce $x_i$ and XORs it with his preauthorization $n_3$ nonce to produce $n_{3i}$. Bob supplies Alice with the ($\{c_2\}$, $n_{3i}$) preauthorization tuple. Alice applies the $\{c_2\}$ challenges to her PUF using $n_{3i}$ to select a specific set of HELP parameters. The HELP parameters further expand the CRP space beyond the $\{c_2\}$ challenges, and for the PUF-Cash protocol, they also cryptographically bind her response R' to the ($\{c_2\}$, $n_{3i}$) tuple. She then XOR encrypts R' with the $n2s$ as $e_{R'n2s}$ = (R' XOR $n_{2s}$) and transmits $e_{R'n2s}$ and the single helper data ($SHD_C$) bit-string to Bob as an e-Cash payment.
- **Step 7**: Bob authenticates and generates a session key with the bank. He AES encrypts his $n_3$ and $x_i$ with his key $SK_b$ as $E_{nxi}$. He then transmits the $SHD_C$ bit-string, $E_{nxi}$ and the $e_{R'n2s}$ to the bank. The bank decrypts $E_{nxi}$ to recover the $n_3$ and $x_i$ and validates that $n_3$ was allocated to Bob during an earlier preauthorization operation. The challenges $\{c_2\}$ associated with $n_3$ in $list_{PA}$ are retrieved. The bank creates the nonce used by Alice, $n_{3i}$, by XORing the $n_3$ and $x_i$.
- **Step 8**: The bank carries out a search process to validate each of Bob's $e_{R'n2s}$. First, the bank identifies the anonymous source (Alice) of the e-Cash tokens by executing a software version of the HELP algorithm, referred to as single-helper-data bit-string generation (SBG). This process was described earlier in Section 3.3. The challenges ($\{c_2\}$, $n_{3i}$) are used as input to the HELP algorithm along with each set of $PN_i$ stored for anonymous customer $i$ in the $AT_{DB}$ database. The output of this operation is designated as $\{R_i, SHD_{Bi}\}$ in Figure 7, where $R_i$ is the response bit-string and $SHD_{Bi}$ is the corresponding single-helper-data bit-string.

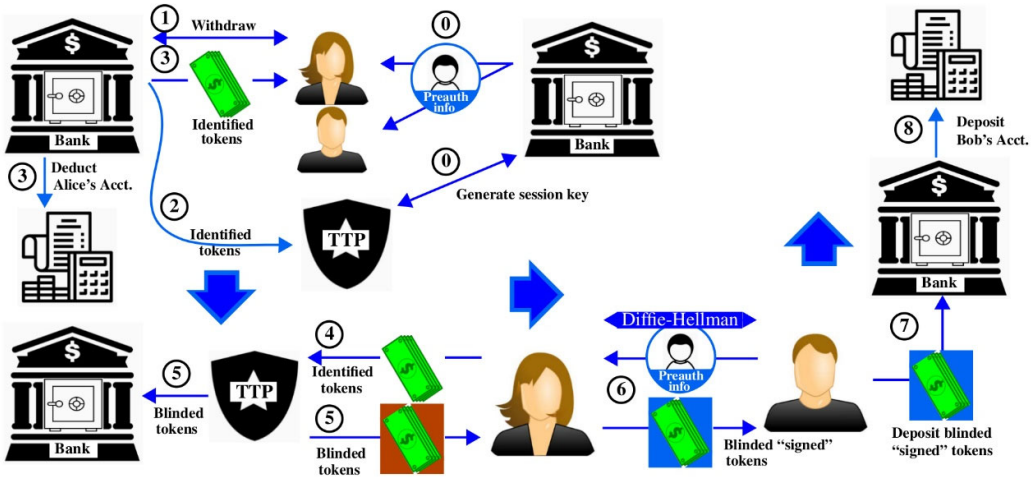


**Figure 8.** Major steps of the PUF-Cash Transaction Protocol.

Cobra is then used to correlate Alice's $SHD_C$ to each of the $SHD_{Bi}$ in an attempt to find a match (see Section 3.4). If successful, the matching pair $\{R_y, SHD_{By}\}$ is used in the following to identify the set of $n_{2x}$ stored in the bank's $list_{n2s}$ that corresponds to the set of $e_{R'n2s}$ provided by Bob.

For each $e_{R'n2}$, the $n_{2x}$ in $list_{n2s}$ are parsed to recover a $R'_x$ = ($n_{2x}$ XOR $e_{R'n2}$). A second HELP algorithm called dual-helper-data authentication (DHDA) is used to determine if the $R'_x$ matches, bit-by-bit, to the $R_y$ in the matching tuple ($R_y$, $SHD_{By}$). The DHDA uses both the $SHD_C$ and the $SHD_{By}$ helper data bit-strings and returns 'match' if successful. This process is identical to the DHD_Authen algorithm described earlier in Section 3.5.

An exact match validates the $n_{2x}$ as being associated with a withdrawal carried out earlier by Alice. In such cases, the bank transfers 1 cent to Bob's account and removes the $n_{2x}$ from $list_{n2s}$ to prevent double spending. An arbitrary $n_1$ is also removed from the e-Cash floating pool to allow the bank to maintain accountability between withdrawals and deposits. Note that although each $e_{R'n2}$ is authenticated as described, only the first $e_{R'n2}$ is targeted in the search because the remaining $e_{R'n2}$ are stored consecutively in $list_{n2s}$. Therefore, the run time is linear and proportional to the size of $list_{n2s}$.

## 5. Attack Surface Analysis

### *5.1. Overview*

An e-Cash protocol must guard against counterfeiting and double-spending. Counterfeiting specifically refers to the creation of a value or a token outside the purview of the issuing authority. Meanwhile, double-spending refers to the ability to exchange a value or spend a unique token more than once. The latter is a harder property to guard against in digital systems. Counterfeiting and double-spending are inextricably linked in value-based systems, but they may be treated distinctly in token-based systems. This section will explore various attacks possible on PUF-Cash within the context of either counterfeiting or double-spending PUF-Cash tokens.

Execution Environment

The execution environment, or hardware platform, for PUF-Cash is envisaged as a single-purpose device. This device is expected to consist of a cryptographic engine built around HELP either implemented in an Application Specific Integrated Circuit (ASIC) or on an FPGA, a nonvolatile memory for storing challenges and encrypted e-Cash tokens, and a microcontroller to transition through the protocol states, operate peripherals (display and/or biometrics), and manage communications channels. Embedding HELP inside a general computer device requires additional consideration, as the attack surface is much larger (for instance through software side-channel attacks), and is therefore relegated to future work.

### *5.2. Operational Example*

The proposed protocol consists of four entities (Alice, Bob, the bank, and TTP) and the following directed channels of communication (Alice->bank, bank->TTP, Alice->TTP, Alice-> Bob, and Bob->bank). A sample transaction sequence, annotated with numbers in Figure 8, proceeds as follows:

1. Startup and enrollment: The bank and the TTP authenticate and establish a session key $SK_t$. The e-Cash devices given to Alice and Bob are initialized with preauthorization information.
2. Alice makes a withdrawal request from the bank.
3. The bank transmits an encrypted copy of Alice's e-Cash tokens ($n1s$) and her session key $Sk_a$ to the TTP, which stores them in $list_{n1s}$.
4. The bank transmits an encrypted copy of Alice's e-Cash tokens ($n1s$) to Alice and deducts the requested amount from her account.
5. Alice then communicates with the TTP to blind her $n1s$.
6. The TTP looks up the $n1s$ received from Alice in $list_{n1s}$ and exchanges each successful $n1$ lookup with a randomly generated nonce, $n2$, constituting a blinded e-Cash token. The TTP transmits encrypted versions of the $n2s$ to both Alice and the bank. The bank stores each $n2$ in $list_{n2s.}$
7. Once the e-Cash tokens are blinded, Alice can pay Bob without requiring communications to the bank or the TTP. To initiate the transfer, Bob and Alice establish a secure channel using the Elliptic Curve Diffie–Hellman component of the TLS 1.2 standard. Bob then transmits his preauthorization information to Alice who applies it to her PUF. Alice binds her PUF response R′ to the blinded $n2s$ (signs them) and transmits them back to Bob.
8. Once Bob receives the signed $n2s$, he may deposit them into his account. Bob transmits the signed $n2s$ and his preauthorization information to the bank. The bank validates Bob's preauthorization information by looking up his $n_3$ in $list_{PA}$ and retrieves $\{c_2\}$.

9. The bank then validates each of Bob's *n*2*s* by searching for them in $list_{n2s}$. Bob's account balance is updated with the deposit amount.

PUF-Cash was originally designed to enable de-networked transactions (i.e., the timing of the communications channels is variable). Although multiple variations are possible, two primary timelines are the most practical and therefore worthy of assessment, as depicted in Figure 9.

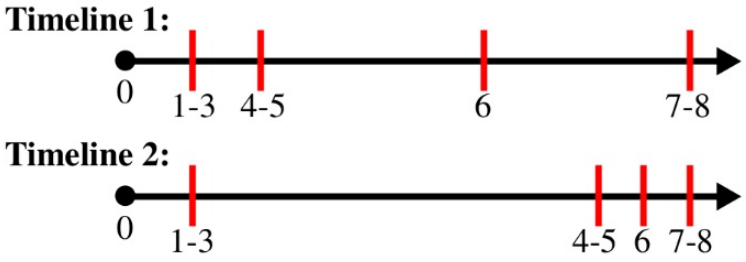


**Figure 9.** Sequence of transactions along a timeline. (1) Offline scenario with Alice holding blinded tokens for an extended period of time before transferring to Bob, and (2) online scenario where Alice blinds tokens right before transferring to Bob.

Timeline 1 depicts the offline scenario use case of PUF-Cash, where Alice and Bob may have local connectivity, but remote connectivity to the bank and TTP are absent. In this scenario, Alice will acquire tokens from the bank (1–3) and blind them with the TTP (4–5) in a very short time-frame (within seconds, limited by network latency), while connectivity exists. Then, Alice may store the blinded tokens on her device for days, much like cash, before transferring to Bob (6). This transfer will rely on local channels, and no connectivity to the bank or TTP is needed to complete the exchange. Bob may then store the transferred tokens for an additional period of time on-device before depositing them into the bank (7–8).

Meanwhile, Timeline 2 captures the merchant and online payment scenarios of PUF-Cash. Here, Bob represents a merchant terminal or an online store with persistent connectivity to the bank and TTP. Initially, Alice withdraws tokens from the bank, but stores them unblinded on-device. Unblinded tokens are directly bound to Alice and cannot be spent by other devices. Once Alice is ready to pay, the tokens are blinded (4–5), sent to Bob (6), and deposited by Bob (7–8) in rapid fashion (within seconds, limited by network latency).

It can be observed that blinded tokens are easiest to steal between Steps 4–5 and 6. Given the large delay (on the order of days) between these two steps in Timeline 1, the risk of theft is higher in Timeline 1. Therefore, the security profile of Timeline 1 is a super-set of Timeline 2.

### *5.3. Protocol Guarantees*

#### 5.3.1. XOR Exchange Security

The proposed protocol is a variant of an XOR exchange protocol, where each transaction in the sequence relies on XOR encryption. The choice for an XOR-based protocol was driven by the need to target low-computation and low-power devices, where the cryptographic payload is minimized. In an XOR protocol, the symmetric cryptosystem is assumed to be unbreakable if the keys are unique for every fresh instance of the protocol (aka a one-time pad). In PUF-Cash, the group of possible XOR keys is defined over the set of challenge–response pairs generated during enrollment that are unique to each device.

An XOR exchange, if improperly conducted, can leak secrets about the protocol. Here we perform an analysis to investigate the information leaked to a perpetrator who knows the protocol and has managed to infiltrate the secure channel. In this scenario, the perpetrator observes the following information assuming a single token transfer:

$$Bank \rightarrow Alice\colon\ E_{SKa}(n_1)$$

$$Alice -> TTP: \quad SKa \oplus n_1$$

$$TTP -> Alice: \quad E_{SKa}(n_1 \parallel n_2)$$

$$Bob -> Alice: \quad \{c2\}, (x_i \oplus n_3)$$

$$Alice -> Bob: \quad R' \oplus n_2$$

Messages prefixed with an $E()$ are AES-128 encrypted and not subject to XOR analysis. The remaining messages can be assembled as follows:

$$SK_a \oplus n_1 \oplus (\{c_2\} \,||\, (x_i \oplus n_3)) \oplus R' \oplus n_2$$

It can be observed that the sequence is irreducible without breaking AES-128. Therefore, the only quantity a perpetrator observes are the $\{c_2\}$ challenges issued by Bob to Alice. It is noted that none of the secrets (nonces or tokens) are uncovered by the perpetrator, and since observing the challenges confers no benefit, the protocol is proven not to leak any secrets.

#### 5.3.2. Uniqueness

HELP is a variant of a strong PUF, and as such relies on the generation of challenge–response pairs as a fundamental building block for random number generation. Therefore, the randomness of CRPs is crucial to the security of the XOR-exchange transaction sequence. In HELP, a component of the challenges, referred to as the path-select-masks, can be configured by the server to select different sets of $k$ PN from the set of $M$ possible PNs [25], where PN refers to 16-bit average path delays for each path chosen by the path-select-masks.

The total number of unique bits that each device can generate is related to the number of PN stored in the database for each device. The experimental results presented in this work utilize a database containing 3320 PNR and 3320 PNF values per device (this number can be increased into the millions if needed). The HELP processing algorithm selects 2048 PNR and 2048 PNF randomly from each of these sets to create 2048 PN differences (PND). The number of possible differences is $3320^2$ ~= 11 million or $2^{23.4}$, where each PND is associated with a unique bit.

Subsequent postprocessing steps for temperature–voltage compensation and modulus operations increase the search space by factor of approximately $2^{17}$ unique bits. Therefore, using a relatively small set of 6640 PN in the database, the total number of unique bits per device is ~$2^{40}$. Therefore, each device can generate more than 1/2 billion ($2^{29}$) unique bit-strings and keys, where each authentication and session key generation operation requires 2048 ($2^{11}$) bits. In addition to the authentication and session key generation portions, HELP is also utilized as a true random number generator (TRNG) for all nonces ($n1$, $n2$, and $n3$) used in the system.

#### 5.3.3. Anonymity

The message exchange between Alice and the TTP is loosely equivalent to Chaum's blind signature [4,5]. In Chaum, the bank signs Alice's message and blinding factor with its private key. Then, Alice removes the blinding factor from the signed message via RSA's homomorphic message. In contrast, the TTP implements the blinding factor in PUF-Cash when it exchanges $n1s$ for $n2s$ with the bank. Since the $n2s$ are anonymously granted against the anonymous challenge set, the bank has no way of knowing who the original $n2s$ were issued to upon deposit.

#### 5.3.4. Counterfeiting

Making change is a necessary but unpleasant aspect of cash transactions. In digital currency, making change can be efficient if the tokens are mutable or divisible. If exploited, the mechanism that enables divisibility also enables rapid counterfeiting, as each individual device now becomes capable of generating new tokens from old tokens. In PUF-Cash, the choice was made to issue immutable/indivisible tokens secured by the responses (R') from CRP interaction between the bank and Alice. Once XOR encrypted, these tokens are immutable, and since they include the nonce stored on the bank and TTP ledgers, unforgeable. Counterfeiting is therefore impossible.

To facilitate making change, all tokens are unitary, valued at 1 cent, and therefore each transaction can be completed by imparting a precise, finite number of whole tokens. Unitary tokens are less efficient than divisible tokens in terms of the bandwidth required per transaction. In the present protocol, each blinded token is comprised of 128 bits precisely. An example transaction of $100 requires 10,000 tokens, resulting in a reasonable 160 KB of information transfer.

#### 5.3.5. Double-Spending

In the original version of the protocol, Alice's blinded tokens were transitive, in that once the blinded tokens were in Bob's possession, they could be re-spent (i.e., transferred to Charlie without requiring contact to the bank or the TTP). The following problems emerged in this model: (1) Alice does not have any guarantees that Bob belongs to a trusted network, (2) Alice had no evidence of a transaction with Bob, and (3) it was possible for a malicious Bob to send the same blinded token to Charlie and Dave, potentially implicating Alice as the thief. Double-spending was still not possible, however, as the bank would reject the same token arriving from two different sources.

To mitigate these and other risks, preauthorization information was introduced to enable Alice to sign her e-Cash tokens ($n2s$) using challenge information from Bob's preauthorization tuple ($\{c_2\}$, $n_{3i}$). The bank also records Bob's preauthorization tuple establishing a verifiable transaction sequence between Alice and Bob and then between Bob and the bank. During a transfer, Alice binds Bob's unique nonce ($x_i$ XOR $n_3$) using Alice's PUF-generated $R'$ to each blinded token. This confers an authentication benefit to Bob and the bank because the $n2s$ are XOR encrypted during transfers and, therefore, are never exposed to adversaries or even to Bob.

Alice can lose ($R'$ XOR $n2$), similar to physical banknotes. It is possible to mitigate risk of loss if anonymity is compromised as per the following scheme: The bank can time-stamp each $n2$ received from the TTP. The $n2s$ that exceed a pre-established time duration without redemption are potential instances of funds lost by Alice. If Alice informs the bank and provides transaction logs from her own device, then the bank can coordinate with the TTP to recover funds.

#### 5.3.6. Network Attack Vectors

Network attacks exploit the communications channel and the information exchange between actors participating in the protocol. Recalling the four possible actors (Alice, Bob, bank, and TTP), the following attack vectors are possible and aimed for discussion.

1. Man-in-the-middle (MITM)
   MITM is a communications channel attack, where Mallory may masquerade as Bob to Alice and Alice to Bob. Such an attack is always possible during Step 3 in both timelines, where Alice exchanges her value with Bob. The primary guard against this attack is mutual authentication via signed public–private key pairs. This attack is thwarted by defense in depth built into the PUF-cash protocol. To successfully attack the protocol, Mallory must:

   a. Substitute Alice and Bob's certificates with equivalent versions signed by the bank (thwart mutual authentication).
   b. Issue and supply currency tokens that look equivalent to Alice's tokens.
   c. Issue and supply preauthorization information that looks equivalent to Bob's.
   d. Issue a challenge–response pair from the set of common challenges stored at the bank or TTP.

   Since all four breaches must be coincident for the attack to succeed, PUF-Cash demonstrates defense in depth and is highly resistant to an MITM attack.

2. Replay Attack
   Each step of the transaction is protected by a fresh nonce or a new secret key generation step. In the transaction between Alice and the bank, a unique secret key and fresh nonces ($n1s$) are chosen. In the transaction between Alice and the TTP, the secret key is unique to the session, generated from the challenge–response pairs uniquely chosen for that transfer, and encrypted via AES-128. Similarly, the nonces supplied by the TTP as part of the blinded tokens ($n2s$) are

also unique. In addition, the preauthorized information generated by the bank for Bob also embed a unique nonce ($x_i$ XOR $n_3$) per transaction, which is then cryptographically bound to the blinded tokens by Alice before transferring to Bob. As described in the previous section, the uniqueness of the key is guaranteed. The bank and the TTP utilize PUF-based true random number generators (TRNGs) for nonce generation, eliminating the risk of a replay attack due to poor random number generation. Finally, even when using the same parameters and challenges, any given $R'$ will change from one regeneration to the next because of environmental noise. Therefore, the probability that an identical $R'$ would be generated by Alice over the lifetime of Alice's PUF is very low, completely mitigating the risk of replay attacks.

3. Denial of Service Attack
   In the PUF-Cash protocol, CRP authentication relies on two databases, both of which are stored at the bank. Since the databases are centralized and will offer a public-facing endpoint, it becomes possible for an adversary to perform a (distributed) denial of service attack (DDoS) against these entries. Transactions between Alice and Bob are localized and, therefore, decentralized. Since no third-party is involved in the exchange, and if a low-power direct communications channel is employed for interactions between Alice and Bob, then DDoS attacks are extremely improbable.

4. Sybil (Impersonation) Attack:
   Although an instance of HELP is impervious to impersonation attacks, as a natural property of strong PUFs in general, an impersonation attack is possible if the PN stored in the bank databases ($NAT_{DB}$ and $AT_{DB}$) are leaked. These PN, coupled with a software implementation of the HELP algorithms, and stolen credentials from a field device, could enable a non-PUF, counterfeit device to impersonate Bob in the Alice <-> Bob exchange, potentially resulting in loss of funds for Alice. For this attack to succeed, the attacker must (a) breach the bank's security to access both databases, (2) find a way to match entries in $NAT_{DB}$ to entries in the anonymous $AT_{DB}$ for a single device, and (3) strip the secure channel credentials from the matching physical field device. Note that the attack can be thwarted by re-registering the impersonated device against a different set of challenges, as the hardware PUF is capable of generating much larger numbers of PN than those recorded at the bank databases.

### 5.3.7. Platform Attack Vectors

1. Model-Building Attacks

HELP is a variant of a strong PUF and, as such, is therefore theoretically susceptible to model-building attacks [16], which can be extraordinarily effective if employed via machine learning techniques [26]. Despite improvements, XOR Arbiter PUFs [27] remain particularly vulnerable to machine learning attacks [28]. HELP is resilient against machine learning attacks because the source of entropy is a complex combinational logic block implemented without identically designed layout structures. All successful modeling attacks to date leverage identically designed features within the elements that define the PUF architecture, which do not exist in HELP. Moreover, the delays of signals propagating along path segments creates uncertainty regarding which path dominates the timing and subsequent determination of the PN. In other words, the path that is actually tested under a given two-vector sequence (challenge vector) may vary from one copy of the chip to another. Last, determining the two-vector sequences that test all of the paths in the PUF is nontrivial, requiring automatic test pattern generation and, depending on the combinational logic block, may take months of computing effort. These factors combine to make it difficult to create a machine learning model and then evaluate that model at runtime [24].

2. Software Side-Channel Attacks

The PUF-Cash device is currently intended for a single-purpose device. A trusted hardware, firmware, and operating system are assumed for the protocol's execution environment if deployed as a production system; therefore, software side-channel attacks are highly improbable. Careful

consideration will need to be taken when PUF-Cash is embedded within a handheld general compute device, such as a smartphone. In a preemptive multitasking computing platform, additional side-channels within the operating system, firmware, and CPU stacks become available for exploitation and must be mitigated. Under those circumstances, cryptographic computations for the protocol may be conducted within a trusted execution environment such as TrustZone. Further research is required in this area.

3. Power-Channel Attacks

Although HELP itself is not susceptible to power analysis attacks because it does not repeatedly use data as required by such attacks, encryption engines such as AES, which use the PUF-generated key, are susceptible. Side-channel attack countermeasures can be employed as needed to increase the resistance against such attacks [29].

## 6. Experimental Results

A prototype implementation of PUF-Cash was built and tested to evaluate performance, measure overhead, and observe the statistical quality of the authentication bit-strings and session keys. The system implementation is shown in Figure 10. Note that the implementation did not rely on specialized hardware or software configurations to maximize end-to-end performance, rather, choices for the hardware and software platforms were made from commercial off-the-shelf components to observe the representative performance in typical environments. The black solid arrows indicate network-based communication, while the blue dotted arrows signify database access by the bank's server. The components of the prototype are given as follows:

- **Bank**: A Dell PowerEdge T440 Server with thirty-two cores at 1.8 GHz each and 128 GB of main memory.
- **Databases:** Two lightweight sqlite3 databases, configured with enrollment data (data sets) for 95 FPGAs, each containing 6640 PN per FPGA. These databases are expanded in separate experiments to include increments of 1000 randomly generated data sets up to 5095. The databases also store the two-vector sequences and path-select-masks (challenges) in a form that allows random subsets of 4096 PN to be drawn out for each of HELP-related operations, including the mutual authentication (MA), session key generation (SKG), and R′ (Rp) response generation operations. Each fielded device adds approximately 400 KB of enrollment data to each of the databases. The sizes of the databases with 95 data sets are approximately 32 MB, while those with 5095 data sets are just over 2 GB.
- **TTP, Merchant, Alice, Bob, and Charles:** Digilent Zybo boards with Xilinx Zynq 7010 SoC FPGAs represent the fielded chips. Each contains an instance of HELP, and each has enrollment data stored in the databases.

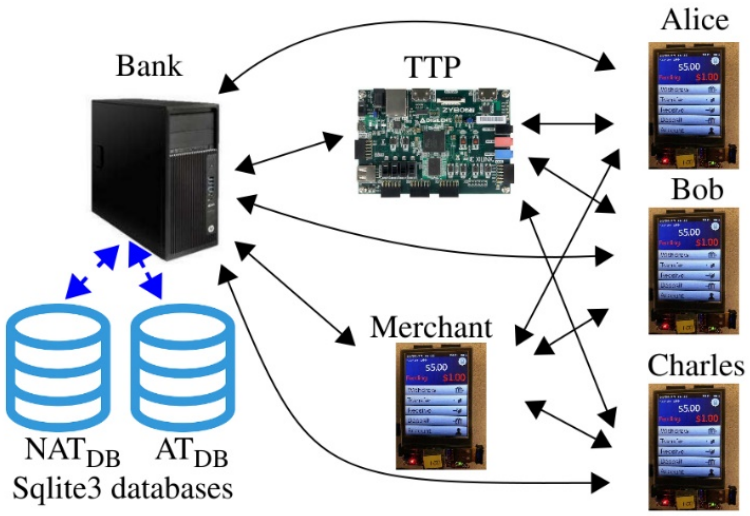

**Figure 10.** Experiment Setup. TTP, trusted third party.

All six computing components (i.e., server + 5 Zybo boards) are internet connected using a 100 MB local area network. Alice, Bob, and Charles are configured to withdraw tokens from the bank and then engage to pay their e-Cash to the merchant. The merchant accepts the e-Cash tokens in exchange for some product or service. Alice, Bob, and Charles repeat this operation indefinitely, following the steps of the protocol shown in Figure 7.

The protocol was run multiple times using six instances of the databases, configured with 95 through 5095 fielded devices in steps of 1000. These six experiments were repeated three times using e-Cash tokens of size 1, 5000, and per transaction. Given a 1 cent denomination, these experiments represent transfers of 1 cent, $50, and $100, respectively. Note that each nonce representing the tokens ($n1$, $n2$ and $n3$) are 128 bits in size.

The C program implementing the interface at the bank was configured to record transaction times for the withdrawal, preauthorization, and deposit transactions. Figure 11 reports the average transaction times of all three of these operations measured by running the algorithm on a single processor (of the 32). The times are plotted as a function of the database size (x-axis) and number of e-Cash tokens (y-axis) and include network communication delays. TTP transaction times associated with the ($n_1$, $n_2$) exchange with Alice and the bank are less than 500 ms in all experiments. Moreover, the transaction time associated with the TTP-bank authentication and session key generation occurs only once at the beginning of the protocol run and takes approximately 3 s to complete.

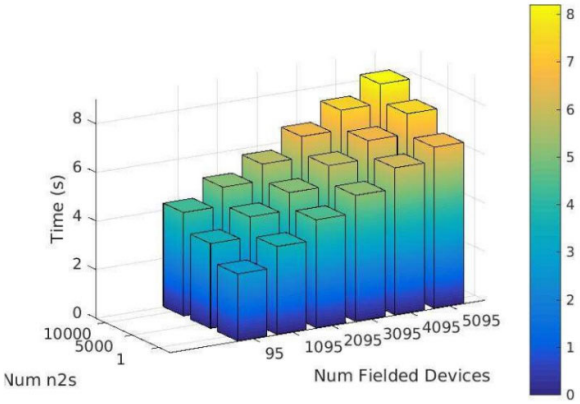


**Figure 11.** Average transaction times as measured by the bank.

This graph clearly shows that the protocol transaction times scale linearly with the number of fielded devices and tokens, with an upper bound of less than 10 s per transaction. Given the independence of the operations, we expect that a multithreaded implementation would produce transaction times as shown here but for much larger databases (e.g., those configured with enrollment data for hundreds of thousands of fielded devices).

Each authentication, session key generation, and Rp response generation operation carried out by the HELP hardware implementation takes approximately one second. These components of the transaction times are included in the times reported in Figure 11. No failures in authentication or session key regeneration were observed over the runs of the protocol carried out in these experiments.

The PUF-Cash deposit transaction time is proportional to ($n$ + $m$), with $n$ representing the number of customers enrolled in the $AT_{DB}$ and $m$ representing the number of deposited tokens. The time to construct the $SHD_{Bi}$ for each customer is approximately 2 ms, and the match time per token is approximately 60 µs. This enables each processor to compute $SHD_{Bi}$ at a rate of more than 500 per second and to carry out token matching operations at a rate of more than 16,000 per second. These rates were validated using a special run of the protocol in which the number of $n2s$ in $list_{n2s}$ was allowed to grow to sizes with more than 2 million tokens.

A separate run of the protocol was carried out for 63 h using 5000 tokens per transaction and the database configured with 95 customers. The authentication bit-strings (ABs) from DHD_Authen

(Figure 4b) and session keys (SKs) from DHD_Key (Figure 5) generated by the 5 Zybo boards were stored and analyzed using a set of statistical tests. Each of Alice, Bob, and Charles carried out 6422 fielded chip MA and SKG operations with the bank at a little over 100/h. The merchant mutually authenticated 38,534 times, once each for a preauthorization and a deposit transaction, over the 63 h test period at approximately 610/h, which is approximately once every 6 s. Note that these rates are measured using only one processor of the 32 available on the server; therefore, significant speedups can be achieved by introducing parallelism.

An open source version of 128-bit AES software was used to encrypt data as specified in the protocol. All authentication bit-strings (ABs) and session keys (SKs) generated on the Zybo boards were produced on-the-fly by a Zynq programmable logic implementation of HELP. The resource utilization of HELP is 6500 Look-up Tables (LUTs), 1900 FF, 1 Mixed Mode Clock Manager (MMCM), and 1 27-bit multiplier. An ASIC implementation of HELP synthesized with a commercial standard cell library, implemented using a glitch-free 32-bit column of AES as the source of entropy and with a 128-stage time-to-digital converter for timing, consists of 10,779 gates and 2476 FFs.

*Statistical Analysis*

Figure 12 illustrates two statistical metrics that relate to the randomness of the ABs and SKs generated by each Zybo board representing Alice, Bob, Charles, and the merchant. Figure 12a plots the Shannon entropy and the minimum entropy (titled 'min-entropy'), while Figure 12b plots Hamming distance (HD) and the 3-σvariation associated with the HD distributions. These randomness statistics are computed using the ABs and SKs generated within each 1 h time interval (x-axis) over the 63 h testing period. The statistics using ABs are shown in blue, while those associated with the SKs are shown in black. The results for the four fielded devices are plotted in separate curves and are super-imposed. Note that the merchant generates only ABs so there are only three SK curves. The smallest bit-string size observed across all authentications is 185 bits and the average size is 240 bits. All session key sizes are 128 bits.

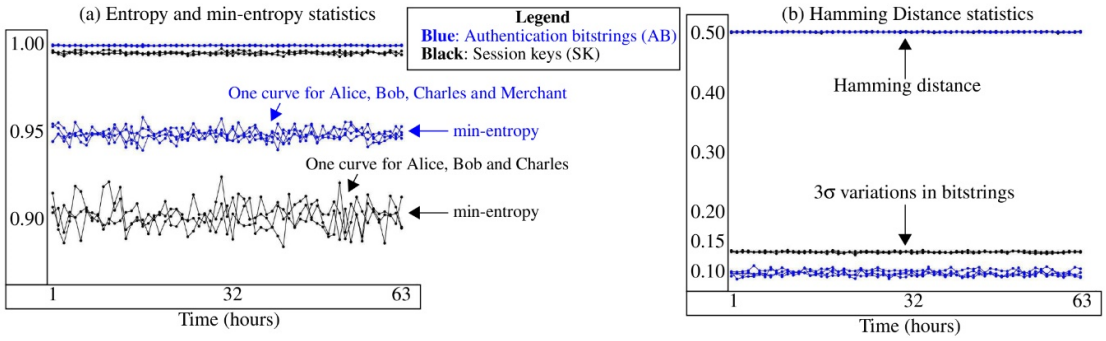


**Figure 12.** (**a**) Entropy and min-entropy statistics and (**b**) Hamming distance (HD) statistics computed using groups of authentication bit-strings (ABs) and session keys (SKs) collected during each 1 h period over the 63 h test period. Approximately 100 bit-strings are used for each Alice, Bob, and Charles statistic and 610 bit-strings for the merchant. The authentication bit-strings vary in size but are at least 185 bits (average is 240 bits), while the session key bit-strings are always 128 bits. The HDs are computed using bit-strings generated by the same chip at different times. The number of HDs computed for each pairing of bit-strings in each 1 h interval is more than 5000 for Alice, Bob, and Charles and more than 185,000 for the merchant.

Entropy and minimum entropy are computed using Equations (1) and (2), and the average values for each group of bit-strings are plotted in Figure 12a. Entropy and min-entropy both measure the information content in the bit-strings where values close to 1.0 indicate the information content is maximal and is the best result possible for bit-strings that need to exhibit true random behavior. The symbol *NB* represents the number of bits, which is 128 for AES session keys but varies for ABs. The AB entropy of approximately 0.998 is slightly better than the SK entropy of approximately 0.995,

but both are near the ideal of 1.00. This trend is also reflected in the min-entropy results with AB min-entropy larger on average (~0.95) than SK min-entropy (~0.91).

$$H(X) = \sum_{i=1}^{2048}\left(-\sum_{j=0}^{1} p_{ij} \cdot \log_2 p_{ij}\right); \quad (1)$$

$$H_\infty(X) = \sum_{i=1}^{2048}\left(-\log_2\left(max\left(p_{ij}\right)\right)\right). \quad (2)$$

Hamming distance is computed by pairing the bit-strings from each group under all combinations and then counting the number of differences in each pairing. Equation (3) gives the expression for HD with *NB* representing the smallest bit-string length among the groups of AB. For the SE HD, *NB* is fixed at 128. From Figure 12b, AB and SK HDs for each of the groups are at or very close to the ideal value of 0.50. The expected 3σ for SE HD using a binomial distribution is $3\sqrt{2.5NB}$, which as a fraction is $\frac{3\sqrt{128\times0.25}}{128} = 0.13$, so the data closely match the expected result. The expected 3σ for AB HD using 240 bits (the average number) is 0.097, which is again a good match to the results.

$$HD = \left(\frac{1}{NCC} \cdot \sum_{i=1}^{NC}\sum_{j=i+1}^{NC} \frac{\left(\sum_{k=1}^{NB}\left(BS_{i,k} \oplus BS_{j,k}\right)\right)}{NB}\right). \quad (3)$$

A long, concatenated set of AB and SK bit-strings were created using the 6422 individual bit-strings from each fielded device. The four AB bit-strings are 3,373,776 bits long, while the three SK bit-strings are 822,016 bits long. The bit-strings were subjected to the 15 tests defined within the National Institute of Standards and Technology (NIST) statistical test suite [30] to evaluate randomness, and all tests were passed.

Interchip HD measures uniqueness of the bit-strings from one fielded device to another, in contrast to the HDs reported above, which measure randomness on each device. The Interchip HDs computed using the AB bit-strings for four fielded devices is 0.5000 and using the SK bit-strings for three fielded devices is 0.4995. A reliability analysis included in prior work [3] depicts bit flip error probabilities at less than 1 in a million ($1 \times 10^{-6}$) for HELP when evaluated across industrial-level specifications for supply voltage and temperature.

## 7. Conclusions

A PUF-based e-cash protocol, called PUF-Cash, is proposed and evaluated on FPGAs using metrics related to performance, area overhead, and security. The latter is measured by analyzing the statistical characteristics of the generated authentication bit-strings and session keys using the NIST statistical test suite and techniques that measure entropy, min-entropy, and hamming distance. The PUF-Cash implementation is shown (1) to meet acceptable time constraints associated with a commercial e-cash system, (2) to have low FPGA resource utilization, and (3) to possess robust statistical properties. PUF-Cash represents the first PUF-based protocol for e-cash systems and presents novel techniques for providing anonymity, for increasing robustness to model-building attacks, and for reducing computing complexity on resource-constrained customer devices. Experimental results indicate that PUF cash is resilient, scalable, and can be fielded on commercially available hardware. Future work will focus on token transitivity to enable Alice->Bob->Charlie transfers and PUF-based field authentication between Alice and Bob, negating the need for external certificate-based authentication layers.

**Author Contributions:** Conceptualization, J.P. and C.M.; methodology, J.P., C.M., and J.C.; software, J.P. and F.S.; validation, J.P. and W.C.; formal analysis, C.M. and J.P., C.H. investigation, J.P., C.H., and C.M.; resources, J.P. and F.S.; data curation, J.P. and F.S.; writing—original draft preparation, J.P. and C.M.; writing—J.C., J.P., C.M., and X.X.; visualization, J.C. and W.C.; supervision, J.P.; project administration, J.P.; funding acquisition, none.

**Funding:** This research received no external funding.

**Conflicts of Interest**: The authors declare no conflict of interest.

## References

1. Gassend, B.; Clarke, D.E.; Van Dijk, M.; Devadas, S. Silicon Physical Random Functions. In Proceedings of the Conference on Computer and Communications Security, New York, NY, USA, 18–22 November 2002; pp. 148–160.
2. Aarestad, J.; Ortiz, P.; Acharyya, D.; Plusquellic, J. HELP: A Hardware-Embedded Delay-Based PUF. *Des. Test Comput.* **2013**, *30*, 17–25.
3. Che, W.; Martin, M.; Pocklassery, G.; Kajuluri, V.K.; Saqib, F.; Plusquellic, J. A Privacy-Preserving, Mutual PUF-Based Authentication Protocol. *Cryptography* **2017**, *1*, 3.
4. Chaum, D. Security Without Identification: Transaction Systems to Make Big Brother Obsolete. *Commun. ACM* **1987**, *28*, 1030–1044.
5. Chaum, D. Blind Signatures for Untraceable Payments. In *Advances in Cryptology*; Springer: Boston, MA, USA, 1982.
6. Chaum, D.; Fiat, A.; Naor, M. Untraceable Electronic Cash. In *Advances in Cryptology—CRYPTO' 88, Proceedings of the Conference on the Theory and Application of Cryptography, Santa Barbara, CA, USA, 21–25 August 1988*; Springer: Berlin/Heidelberg, Germany, 1842; pp. 319–327.
7. Schoenmaker, B. Security Aspects of the e-cash Payment System. In *State of the Art in Applied Cryptography*; Springer: Berlin/Heidelberg, Germany,1998; Volume 1528, pp. 338–352.
8. Brands, S. Untraceable Off-Line Cash in Wallet with Observers. In Proceedings of the International Cryptology Conference on Advances in Cryptology, Santa Barbara, CA, USA, 22–26 August 1993; pp. 302–318.
9. Pointcheval, D.; Stern, J. Provably Secure Blind Signature Schemes. *Adv. Cryptol.* **1996**, *1163*, 252–265.
10. Camenisch, J.; Lysanskaya, A.; Meyerovich, M. *Endorsed E-Cash IEEE Symposium on Security and Privacy*; IEEE: Berkeley, CA, USA, 2007.
11. Fujisaki, E.; Okamoto, T. Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations. *Adv. Cryptogr*. **1997**, *1294*, 16–30.
12. Okamoto, T. An efficient divisible electronic cash scheme. In *Advances in Cryptology—CRYPT0' 95, Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 27–31 August 1995;* Springer: Berlin/Heidelberg, Germany, 1842; pp. 438–451.
13. Ruckert, M. Lattice-based Blind Signatures. In Proceedings of the AsiaCrypt, Singapore, 5–9 December 2010.
14. Yang, B.; Yang, K.; Zhang, Z.; Qin, Y.; Feng, D. AEP-M: Practical Anonymous E-Payment for Mobile Devices using ARM TrustZone and Divisible E-Cash, Information Security. In *Information Security, Proceedings of the International Conference on Information Security, Honolulu, HI, USA, 3–6 September 2016*; Springer: Berlin/Heidelberg, Germany, 1842; pp. 130–146.
15. Sakalauskas, E.; Muleravicius, J.; Timofejeva, I. Computational Resources for Mobile E-Wallet System with Observers. In Proceedings of the Electronics, Palanga, Lithuania, 19–21 June 2017.
16. Rührmair, U.; Sehnke, F.; Sölter, J.; Dror, G.; Devadas, S.; Schmidhuber, J. Modeling Attacks on Physical Unclonable Functions. In Proceedings of the 17th ACM Conference on Computer and Communications Security, Chicago, IL, USA, 4–8 October 2010; pp. 237–249.
17. Plusquellic, J.; Areno, M. Correlation-Based Robust Authentication (Cobra) Using Helper Data Only. Available online: http://www.mdpi.com/2410-387X/2/3/21 (accessed on 21 February 2018).
18. Bolotny, L.; Robins, G. Physically Unclonable Function-based Security and Privacy in RFID Systems. In Proceedings of the PerCom, White Plains, NY, USA, 19–23 March 2007; pp. 211–220.
19. Hammouri, G.; Ozturk, E.; Sunar, B. A Tamper-Proof and Lightweight Authentication Scheme. *Pervasive Mob. Comput.* **2008**, *4*, 807–818.
20. Sadeghi, A.-R.; Visconti, I.; Wachsmann, C. Enhancing RFID Security and Privacy by Physically Unclonable Functions. *Inf. Secur. Cryptogr*. **2010**, *23*, 281–305.
21. Kocabas, U.; Peter, A.; Katzenbeisser, S.; Sadeghi, A. Converse PUF-Based Authentication. In *Trust and Trustworthy Computing, Proceedings of the International Conference on Trust and Trustworthy Computing, Vienna, Austria, 13–15 June 2012*; Springer: Berlin/Heidelberg, Germany; pp. 142–158.
22. Majzoobi, M.; Rostami, M.; Koushanfar, F.; Wallach, D.S.; Devadas, S. Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching. In Proceedings of the Symposium on Security and Privacy Workshop, San Francisco, CA, USA, 24–25 May 2012; pp. 33–44.
23. Aysu, A.; Gulcan, E.; Moryama, D.; Schaumont, P. Compact and Low-power ASIP Design for Lightweight PUF-based Authentication Protocols. *IET Inf. Secur*. **2016**, *10*, 232–241.

24. Che, W.; Martinez-Ramon, M.; Saqib, F.; Plusquellic, J. Delay Model and Machine Learning Exploration of a Hardware-Embedded Delay PUF. In Proceedings of the International Symposium on Hardware-Oriented Security and Trust, Washington, DC, USA, 30 April–4 May 2018.
25. Che, W.; Kajuluri, V.K.; Saqib, F.; Plusquellic, J. Leveraging Distributions in Physical Unclonable Functions *Cryptography* **2017**, *1*, 17.
26. Rührmair, U.; Xu, X.; Solter, J.; Mahmoud, A.; Majzoobi, M.; Koushanfar, F.; Burleson, W. Efficient Power and Timing Side Channels for Physical Unclonable Functions. In *Cryptographic Hardware and Embedded Systems–CHES 2014, Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Busan, Korea, 23–26 September 2014*; Springer: Berlin/Heidelberg, Germany; pp. 476–492.
27. Rostami, M.; Majzoobi, M.; Koushanfar, F.; Wallach, D.; Devadas, S. Robust and Reverse-Engineering Resilient PUF Authentication and Key-Exchange by Substring Matching. *IEE Trans. Emerg. Top. Comput*. **2014**, *2*, 37–49.
28. Delvaux, J. Machine-Learning Attacks on PolyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF-FSMs. *Trans. Inf. Forensics Secur.* **2019**, *14*, 2043–2058.
29. Mangard, S.; Oswald, E.; Popp, T. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*; Springer: Berlin/Heidelberg, Germany, 2007.
30. Bassham, L.; Rukhin, A.; Soto, J.; Nechvatal, J.; Smid, M.; Barker, E.; Leigh, S.; Levenson, M.; Vangel, M.; Banks, D.; et al. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Available online: https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final (accessed on 15 April 2010).