Chapter X: PUF-Based Authentication

Jim Plusquellic - Univ. of New Mexico (jimp@ece.unm.edu)

1. Introduction

The internet-of-everything has created vast opportunities for the integration of microelectronic systems into nearly every aspect of our lives, but it has also expanded the attack surface of such systems, providing an ever-widening opportunity for malicious adversaries to steal private information, destroy property or worse, subvert systems in a manner that results in the loss of human life [1-15]. These problems are becoming particularly acute with the proliferation of mobile computing and the debut of new information-sharing and control systems such as the health information exchange, embedded medical devices, smart grid, home automation, smart cars, smart cards, RFID and sensor networks. Stronger, physical-layer security and trust primitives are needed for modern electronic systems to counter the advantage made available to adversaries by the increasing proliferation, diversity and complexity of software and hardware.

Physical-layer refers to components that are rooted in the hardware, and that provide support for secure execution of algorithms, and for secure generation and storage of secrets (keys). A *physical unclonable function* (PUF) is a physical-layer primitive that is designed to derive entropy (randomness) from variations in the structural and electrical characteristics of integrated circuits (ICs) [16]. Similar to DNA profiles among humans, no two ICs are (or can intentionally be manufactured to be) identical. PUFs measure and digitize small 'analog' differences among identically designed ICs to generate unique and unclonable bitstrings. The random and persistent nature of the entropy source within ICs address important physical security requirements that relate to the generation and storage of keys. Most PUF designs use standard IC manufacturing processes, which benefits low-cost applications by eliminating the need for costly non-volatile memory (NVM). PUFs can be integrated into any type of system, including system-on-a-chip (SoC), an application specific integrated circuit (ASIC) or field programmable gate array (FPGA).

This chapter focuses on the design of authentication protocols which utilize physical-layer cryptographic primitives such as the PUF, and describes the benefits (and drawbacks) they offer over traditional software-based authentication protocols. PUF-based authentication protocols are less than 15 years old and many have not yet been fully vetted. Therefore, the development of low cost, secure protocols, and proofs of their attack resilience is still very much a moving target. We provide a high-level description of algorithmic security primitives and authentication protocols, and then present a snapshot of the current state-of-the-art, fully acknowledging that the latter is rapidly evolving and still considered an open research problem by the hardware security and trust community.

2. Information Security and Cryptography

The term **information security** refers to vast array of mechanisms, protocols and algorithms which are designed to protect information from unauthorized access, modification and destruction [17]. Information security has four primary objectives including confidentiality, data integrity, authentication and non-repudiation [18]. *Confidentiality* refers to maintaining privacy or secrecy of information and is traditionally ensured using encryption techniques. *Data integrity* relates to a property of the data, that it has not been altered by an unauthorized party, and is typically implemented using secure hashing schemes. *Authentication* is a process that confirms the identity of an entity or the original source of data using corroborative evidence, and can be carried out using

modification detection codes (MDCs), message authentication codes (MACs) and digital signatures. *Non-repudiation* refers to a process that associates an entity with a commitment or action, thereby preventing the entity from claiming otherwise, and is traditionally ensured using digital signature schemes.

The primary goal of **cryptography** is to provide a theoretical basis and practical specifications for techniques that meet these information security goals. A wide variety of cryptographic primitives have been developed to provide information security. Menezes et al. [18] propose a taxonomy which partitions cryptographic primitives into three basic categories, namely *unkeyed primitives*, *symmetric-key primitives* and *public-key primitives*. Unkeyed primitives include cryptographic hash functions, one-way permutations and random sequences. The *keyed* primitives include a wide variety of symmetric and public-key ciphers, MACs (which are keyed hash functions), signatures and pseudo-random number generators (those relevant to authentication are described in the next section). Each primitive can be evaluated according to a set of criteria such as the level of security they provide as well as the performance and overhead associated with a particular implementation of the primitive.

Authentication protocols are implemented as an exchange of messages between two or more parties, usually over an unsecured network. Authentication utilizes cryptographic primitives as countermeasures to adversarial manipulation of the transmitted messages and as mechanisms to protect the interfaces of the communicating entities from information leakage and tracking. PUFs provide novel ways of designing protocols but cannot be used by themselves to implement all of the security requirements of the protocol. Sections 3 and 4 provide an overview of traditional security-related primitives commonly used in authentication protocols, as well as algorithms and evaluation metrics that are required when using PUFs for authentication. Once the groundwork of authentication has been established, we then describe several PUF implementations and PUF-based authentication protocols in Sections 5 and 6.

3. Cryptographic Primitives for Authentication Protocols

A cryptographic protocol is a distributed algorithm defined by a sequence of steps precisely specifying the actions required of two or more entities to achieve a specific security objective [18]. All protocols make use of cryptographic primitives that provide specific security properties. In this section, we briefly describe the primitives most commonly used in authentication protocols.

3.1 Random Number Generation

Random numbers are important in many cryptographic protocols, e.g., session keys, nonces for authentication, randomized procedures, etc. Random numbers must be selected uniformly from a distribution, thereby ensuring that all possible values are equally likely, as a means of maximizing the difficulty of algorithmic and brute force attacks carried out by adversaries against the protocol. Requests that are common in cryptographic protocols include ‘select an element at random from the sequence {1, 2, ..., *n*}’ or ‘generate a random string of symbols of length *m* over the alphabet Γ of *n* symbols’. *Uniformly* refers to the probability that a given symbol is selected and by definition is equal to $1/n$ for an alphabet of *n* symbols, and $1/n^m$ for a string of symbols of length *m*.

Traditionally, deriving random numbers from physical sources was difficult and costly, spurring the development of software-based alternatives such as techniques based on *pseudorandom sequences* and *seed* parameters (PRNGs) [19]. NIST recommends several such cryptographically secure PRNGs, each based on different types of cryptographic primitives such as hash functions,

MACs and block ciphers [20]. Although most are considered cryptographically secure, they each depend on a random seed with high entropy. An *entropy accumulator* can be used to derive the seed from a 'non-ideal' physical source of randomness, whereby the input bitstream produced by the non-ideal source is processed by the entropy accumulator into a *m-bit pool* of high entropy. The entropy accumulator can be a cryptographic hash function [19]. Alternatively, the physical-layer nature of PUFs make them cost-effective and well suited as the physical source of randomness. Recent work shows that appropriate post-processing of PUF responses allow them to be used directly as TRNGs, i.e., without the need of PRNGs [21].

3.2 Cryptographic Hash Functions

As mentioned above, secure hash functions are used to realize a fundamental information security property, namely that related to the *integrity of data*. Compression is a defining characteristic of many-to-one hash functions, whereby binary strings of arbitrary length are mapped to strings of fixed length *n*. The *n-bit* hash output is a compact representation of the input string. The many-to-one property implies that *collisions* are possible, a condition in which two distinct input strings map to the same hash. *Cryptographic* hash functions (referred to as hash functions subsequently) add important security-related properties to traditional hash functions and have the following characteristics [22]:

- It is *easy* to compute the hash for any input string.
- It is computationally infeasible to 1) generate the input string from its hash, 2) modify the input string without changing the hash and 3) find two different input strings which produce the same hash.

More formally, the security properties of a hash function *h* with input message *m* and output $y = h(m)$ are defined as follows:

- *preimage resistance*: Given any hash *y*, it is computationally infeasible to find an *m* such that $h(m) = y$.
- *2nd-preimage resistance*: Given an input *m*, it is computationally infeasible to find a different input *m'* such that $h(m) = h(m')$
- *collision resistance*: It is computationally infeasible to find any two distinct inputs *m* and *m'* such that $h(m) = h(m')$.

Even stronger security properties are possible, for example it should be infeasible to find two inputs that produce *similar* hashes. Ideally, the hash function should behave like a random function, where each hash is equally probably, i.e., uniformly distributed.

There are two fundamental classes of hash functions: **unkeyed hash functions** and **keyed hash functions**. Keyless hash functions can be used to create *modification detection codes* (MDCs), whose main purpose is to confirm data integrity. There are two types of MDCs: *one-way hash functions* (OWHFs) which make it difficult to find an input string *m* that hashes to specific hash value, and *collision resistant hash functions* (CRHFs), which makes it difficult to find two input strings that map to the same hash. OWHFs are preimage and 2nd-preimage resistant, and are considered **weak** one-way hash functions, while CRHFs typically have all three properties and are called **strong** one-way hash functions.

Keyed hash functions provide both message authentication and data integrity and are called *message authentication codes* (MACs) when used in symmetric-encryption protocols, and *digital signatures* when used in asymmetric encryption protocols. Both schemes hash the message and then *sign* it with a key. The receiver authenticates by applying the MAC or digital signature algorithm on the received message and verifies that the received hash matches the locally computed value. Hashing compresses the message and makes this data integrity check more efficient. Although outside the scope of this expository, the chip area and computational complexity of

cryptographic hash functions is much larger than that found in non-cryptographic hash functions [18, Ch. 9].

Similar to authentication protocols, secure hash algorithms continue to evolve, driving periodic changes and additions to the public standards [23-24]. The term *secure hash algorithm* (SHA) is used in reference to a set of public standards maintained by the National Institute of Standards and Technology (NIST). In particular, SHA-3 refers to subset of the cryptographic primitive family *Keccak*, a standard released in August of 2015 that is designed as an alternative to the SHA-2 family of secure hash functions [25].

3.3 Secure Sketches and Fuzzy Extractors

The introduction of PUFs as a primitive in authentication (and encryption) protocols made it necessary to enlist **error-correcting** and **randomness extraction** mechanisms into the suite of cryptographic primitives. The analog characteristics of the entropy source, as well as the embedded analog-to-digital instrumentation components of a PUF instantiation, combined with environmental (temperature, supply voltage), coupling and power supply noise sources make it difficult or impossible to precisely reproduce the bitstrings generated by PUFs from one run of the protocol to the next. When PUF bitstrings are used as input to traditional cryptographic primitives, such as hash functions or encryption algorithms, even a single bit-flip error in the bitstring causes a catastrophic failure in the protocol. Additionally, PUF-generated bitstrings, in many cases, are not ideal from the randomness perspective. Systematic bias effects and correlations inherent to the structure of the entropy source make it difficult for the PUF to produce a bitstring uniformly from the underlying distribution, i.e., such that all bitstrings of a given length are equally likely. Secure sketches, strong extractors, and fuzzy extractors are functions designed to deal with these deficiencies.

There are many types of error-correction algorithms that have been developed to fix errors that occur in bitstrings. The most popular algorithms used for PUFs produce **helper data** as a supplementary source of information during the initial bitstring generation (*Gen*) process, which is later used to fix bit-flip errors during reproduction (*Rep)*. The helper data is typically transmitted and stored openly, in a *public* non-secure location, and therefore, it must reveal as little as possible about the bitstring it is designed to error correct.

The *Sketch* component of a **secure sketch** takes an input *y* and returns a helper data bitstring *w* [26-27]. The *Recover* component takes a 'noisy' input *y'* and a helper bitstring *w* and returns *y"*, which is guaranteed to match the original bitstring *y* as long as the number of bit flip errors is less than *t* (*t* is a parameter that can be selected based on the level of error correction that is needed). The algorithm is characterized by a *security property*, that guarantees that if *y* is selected from a distribution with **min-entropy** *m*, then an adversary can reverse-engineer *y* from *m* with probability no greater than $2^{-m'}$ (*m'* is defined below). Entropy is a measure of the disorder or randomness in a closed system, while min-entropy refers to the worst-case behavior of a random variable and is defined by Eq. 1. It is the negative $\log_2$ of the event with maximum probability.

Dodis et al. [26-27] proposed two algorithms for a secure sketch, both based on binary error-correcting linear block codes. A linear block code is characterized with three parameters given as [*n*, *k*, *t*], which indicate that there are 2^k codewords of length *n* and each codeword is separated from all others by at least *2t-1* bits. The last parameter specifies the error correcting capability of the linear block code, in particular, that up to *t* bits can be corrected.

The **code-offset** construction is the simpler of the two linear block codes. The *Sketch(y)* procedure samples a uniform, random codeword *c* (which is independent of *y*) and produces an *n*-bit

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$ The entropy of a random variable X with probabilities p_i, ..., p_n

$$H(X) = \frac{1}{\ln(2)} \log_2\left(\frac{1}{p}\right)$$ When $p_i = 1/n$ (equal probabilities)

$$H_\infty(X) = min(-\log_2 p_i) = -\log_2(max(p_i))$$ Min-entropy **Eq. 1.**

helper data bitstring *w* using Eq. 2 [19]. The bitstring *w* represents the binary offset between *y* and *c*.

$$w = y \oplus c$$ **Eq. 2.**

Recover(y', w) computes a noisy codeword *c'* using Eq. 3 and then applies an error-correcting procedure to correct *c'* as *c" = Correct(c')*.

$$c' = y' \oplus w \quad => \quad c' = (y \oplus y') \oplus c$$ **Eq. 3.**

The error-corrected value of *y'* is computed as given by Eq. 4. If the number of bits that are

$$y'' = w \oplus c'' = y \oplus (c \oplus c'')$$ **Eq. 4.**

different between *c* and *c'* < *t*, where *t* represents the error-correcting capability of the code, then the algorithm guarantees *y = y"*. Also, *w* discloses at most *n* bits of *y*, of which *k* are independent of *y* (with *k* less than or equal to *n*). Therefore, the *remaining* min-entropy is *m - (n - k)* (specified as *m'* above), where *(n-k)* represents the min-entropy that is lost by exposing *w* to the adversary.

The second algorithm proposed in [26-27] is referred to as the **syndrome** construction. The *Sketch(y)* procedure produces an *(n-k)*-bit helper data bitstring using the operation specified by Eq. 5, where H^T is a parity-check matrix dimensioned as *(n-k)* x *n*.

$$w = y \bullet H^T$$ **Eq. 5.**

The *Recover* procedure computes a syndrome *s* using Eq. 6.

$$s = y' \bullet H^T \oplus w \quad => \quad s = (y \oplus y') \bullet H^T$$ **Eq. 6.**

Error correction is carried out by finding a unique error word *e* such that the *hamming weight* (the number of '1's) in bitstring *e* is less than or equal to *t* (the error-correcting capability of the code). Also, the error word *e* satisfies Eq. 7.

$$s = e \bullet H^T$$ **Eq. 7.**

In both the code-offset and syndrome techniques, the *Recover* procedure is more computationally complex than the *Sketch* procedure. As discussed below, the first PUF-based authentication protocols implemented the *Recover* procedure on the resource-constrained hardware token. Subsequent work proposes a **reverse fuzzy extractor**, which implements *Sketch* on the hardware token and *Recover* on the resource-rich server, making the protocol more cost-effective and attractive for this type of application environment [28].

Similar to error-correction, there is a broad range of techniques for constructing a **randomness extractor**. Section 3.1 described the requirements for random number generation, and practical approaches for extracting randomness from non-ideal physical sources, e.g., those based on the use of *seeded cryptographic PRNGs*. Reference [19], Section 6.2.2 provides a survey of techniques proposed for extracting randomness.

Fuzzy extractors combine a secure sketch with a randomness extractor as shown in Fig. 1 (adapted from [19]). A PUF-based authentication protocol, with the *hardware token*, e.g., smart card, shown on the left and the *secure server*, e.g., bank, shown on the right is also shown to illustrate one possible usage scenario. The *Sketch*, as noted above, takes an input *r*, which, e.g., might be a PUF response to a server-generated challenge *c*, as input and produces helper data *w* (labeled *1st* in the figure). The *Extractor* takes both *r* and a random number (seed) *n* and produces an *entropy distilled* version *z*, which can be stored as a *tuple* (*c*, *z*, *w*, *n*) in a secure database (DB) on the server. This component of the fuzzy extractor is called Generate or *Gen*.

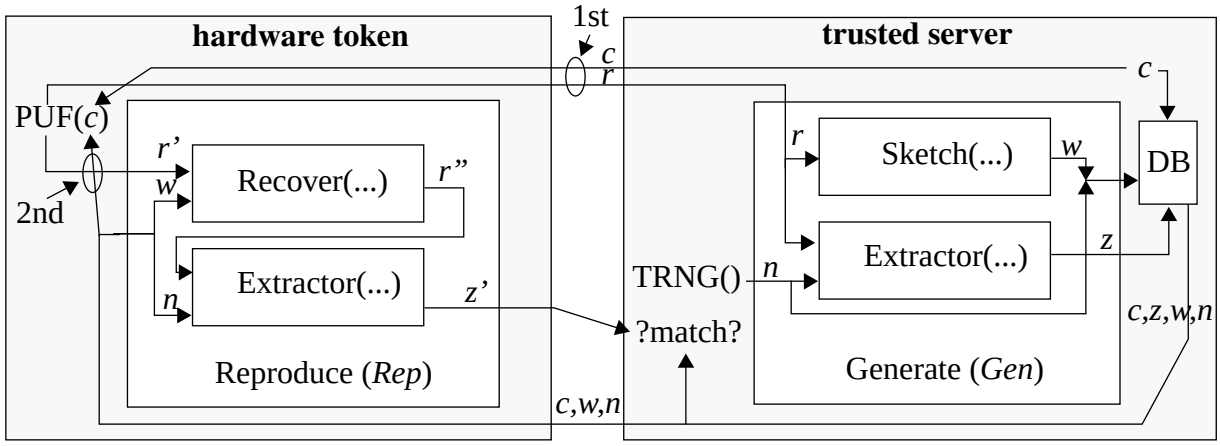


Fig. 1. Fuzzy Extractor.

Authentication in the field begins by selecting a tuple (*c*, *z*, *w*, *n*) from the DB and transmitting the challenge *c*, helper data *w* and the seed *n* to the hardware token. The PUF is challenged a second time with challenge *c* and produces a 'noisy' response *r'* (labeled *2nd* in the figure). The Reproduce or *Rep* process of the fuzzy extractor uses the Recover procedure of the secure sketch to error correct *r'* using helper data *w*. The output *r"* of Recover and the seed *n* are used by the Extractor to generate *z'*. As long as the number of bit flip errors in *r'* is less than *t* (the chosen error correction parameter), the *z'* produced by the token's Extractor will match the server-DB *z* and authentication succeeds. Note that the error corrected *z'* establishes a shared secret between the server and token, which can alternatively be used as input to traditional cryptographic primitives such as hash and block cipher functions (as opposed to being transmitted to the server as shown in the figure).

3.4 Statistical Metrics

PUF generated bitstrings are often evaluated using techniques designed to measure the statistical quality of the bitstrings, which include characteristics such as uniqueness, reproducibility and randomness. *Uniqueness* measures how different the bitstrings are from one device to another in the population. The probability mass function of the binomial distribution is the appropriate statistical characterization function for bitstrings and is given by Eq. 8, with mean and variance given by Eq. 9 and 10, resp. [29]. Eq. 8 gives the probability of getting exactly *k* successes in *n* trials.

$$f(k;n,p) = \frac{n!}{k!(n-k)!}p^k(1-p)^{n-k} \quad \textbf{Eq. 8.}$$

$$\mu_{\text{binomial}} = np \quad \textbf{Eq. 9.}$$

$$\rho_{\text{binomial}} = np(1-p) \quad \textbf{Eq. 10.}$$

Assuming the probability of a '1' in a bitstring of size n produced by a PUF is $p = 0.5$, then $\mu_{binomial}$ indicates that half of the bits will be '1' on average. The same characteristic holds across bitstrings from different devices if the probability of a '1' is 0.5 for any given bit position. It follows then that the average number of bits that are different from one bitstring to another, in this best case scenario, is 50%. The metric used to measure uniqueness is **inter-chip hamming distance** (**HD_{inter}**). HD_{inter} counts the number of bits that are different. In a typical PUF application, the count is computed over all possible pairings of bitstrings produced by different devices in the population and then divided by the total number of bits and multiplied by 100 to yield a percentage. Note that the set of bits that differ between any two arbitrary bitstrings necessarily are distinct from one pairing to another.

Reproducibility measures the PUFs ability to regenerate its bitstring(s) over time and under different environmental conditions. The terms **enrollment** and **regeneration** are used in reference to the bitstring generation process. Enrollment is carried out when a new bitstring is required, while regeneration refers to the process of reproducing the same bitstring at some point later in time. The application determines whether precise regeneration is required, e.g., encryption requires exact replicas of the bitstring (when the bitstring is used as the key) while some authentication schemes have a built-in tolerance to allow some (small) fraction of *bit flip errors* to occur. Regeneration without errors is much more challenging when the process is carried out under different temperatures and/or supply voltages. The metric used to measure reproducibility is **intra-chip hamming distance** (**HD_{intra}**). Similar to HD_{inter}, HD_{intra} counts the number of bits that are different between pairings of bitstrings. For HD_{intra}, however, the pairings of bitstrings are composed from the set of bitstrings produced by a specific device, each regenerated possibly under different environmental conditions with respect to the enrollment conditions. The term *TV corners* is used in reference to the set of environmental conditions used to test the devices, e.g., 85^{o}C and +10% V_{DD} is a TV corner. Similar to HD_{inter}, HD_{intra} is usually expressed as an average percentage over all devices tested in the experiment, by counting the total number of bit flip errors that occur, dividing by the total number of bits inspected and then multiplying by 100. The ideal case is an average HD_{intra} of 0%, i.e. no devices produced any bit flip errors under any TV corner.

The NIST statistical test suite can be used to evaluate the *randomness* of PUF response bitstrings [30]. The NIST tests look for *patterns* in the bitstrings that are not likely to be found at all or above a given frequency in a 'truly random' bitstring. For example, long or short strings of 0's and 1's, or specific patterns repeated in many places in the bitstring work against randomness. The output of the NIST statistical evaluation engine is the *number of chips* that pass the *null hypothesis* for a given test, when evaluated at a *significance level* α (α is set to the default value of 0.01 which reflects a confidence of 99%). The null hypothesis is specified as the condition in which the bitstring-under-test is random. Therefore, a good result is obtained when the number of bitstrings that pass the null hypothesis is large.

The NIST test suite consists of 15 separate tests, all of which have constraints on the size of the bitstring. The following provides an intuitive overview of what the tests measure, with details regarding the bitstring size requirements and applied test statistics omitted (see [30]). The test is always conducted against what is expected in a truly random sequence of similar length.

- Frequency Test: Counts the number of '1' in a bitstring and assesses the closeness of the fraction of '1's to 0.5. All other tests assume this test is passed.
- Block Frequency Test: Same except bitstring is partitioned into *M* blocks. Ensures bitstring is 'locally' random.

- Runs Test: Analyzes the total number of *runs*, i.e., uninterrupted sequences of identical bits, and tests whether the oscillation between '0's and '1's is too fast or too slow.
- Longest Run Test: Analyzes the longest run of '1's within *M-bit* blocks, and tests if it is consistent with the length of the longest run expected in a truly random sequence.
- Rank Test: Analyzes the linear dependence among fixed length substrings in the bitstring, and tests if the *number of ranks*, i.e., number of rows that are linearly independent, of size *M*, *M-1*, etc., match the number expected in a truly random sequence.
- Fourier Transform Test: Analyzes the peak heights in the frequency spectrum of the bitstring, and tests if there are *periodic* features, i.e., repeating patterns close to each other.
- Non-overlapping and Overlapping Template Tests: Analyzes the bitstring for the number of times *pre-specified* target strings occur, to determine if too many occurrences of *non-periodic* patterns occur.
- Universal Test: Analyzes the bitstring to determine the *level of compression* that can be achieved without loss of information.
- Linear Complexity Test: Analyzes the bitstring to determine the length of the smallest set of LFSRs needed to reproduce the sequence.
- Serial and Approximate Entropy Tests: Analyzes the bitstring to test the frequency of all possible 2^m overlapping *m-bit* patterns, to determine if the number is uniform for all possible patterns.
- Cumulative Sums Test: Analyzes the bitstring to determine if the cumulative sum of incrementally increasing (decreasing) partial sequences is too large or too small.
- Random Excursions Test: Analyzes the total number of times that a particular state occurs in a cumulative sum random walk.

4. Traditional, Software-Oriented Authentication

Authentication refers to the process of 'verifying the identity of the communicating principals to one another' [31]. It is usually sub-divided into *entity authentication* and *message (or data origin) authentication* [18], with the former referring to authentication in 'real-time' between two parties about ready to engage in communication while the latter refers to data such as email that may later need to be authenticated by the receiver as to the origin and time sent. Note that authentication of the origin of data also addresses data integrity, i.e., whether the message has been tampered with by unauthorized parties, because unauthorized changes imply the data has a new source.

Authentication is typically carried out between a **prover** (claimant) *A*, e.g., a hardware token such as a smart card, and a **verifier** *B*, e.g., a secure server operated by your bank. The verifier *B* either confirms or *accepts* the prover's identity as authentic or terminates without acceptance, i.e., *rejects*. The information exchanged with verifier *B* must be designed to prevent reuse by *B*, otherwise it could impersonate *A* to a third party *C*. Protocols should guarantee that the probability of *impersonation* is negligible, even when a polynomially large number of previous authentications occur between *A* and *B*.

Authentication can be used for security objectives including access control, entity authentication, message authentication, data integrity, non-repudiation and key authentication. Authentication can be carried out using symmetric encryption techniques, e.g., via *message authentication codes* or MACs, using public/private encryption schemes via *digital signatures* and through authenticated key establishment methods. The most common usage models include access control to a resource, e.g., to computer accounts, ATMs, to software, to a building, etc.

The capabilities provided in the authentication protocol depend on the security requirements. For example, an authentication protocol may be *unilateral*, i.e., from prover to verifier, or *mutual*.

Some protocols may *preserve privacy*, to prevent malicious adversaries from tracking instances of authentications that occur between the prover and verifier over time. Others may be *symmetric* in nature, requiring the use of a shared secret between the prover and verifier provided by interactions, in real-time, with a *trusted third party* (TTP), or may be *asymmetric* with the prover and verifier maintaining their own private secrets. The computational and communication overheads associated with the protocols will depend on the type of protocol, its security requirements and the security properties that must be guaranteed.

4.1 Entity Authentication

Entity authentication techniques can be divided into 3 categories:

- *Something you know*: Passwords, PINs and secret or private keys whose knowledge is demonstrated in challenge-response protocols.
- *Something you possess*: Physical accessory, resembling a passport in function. Magnetic-striped cards, smart-cards and hand-held customized calculators (password generators) which provide time-variant passwords.
- *Something inherent*: Biometrics, e.g., human physical characteristics such as fingerprints, voice, retinal patterns and signatures.

Passwords represent the most widely used form of authentication, but are considered **weak authentication** protocols. Passwords provide unilateral and time-invariant authentication, with the *userid* serving as the claim of identity and the *password* serving as evidence supporting the claim. Attacks include eavesdropping to enable *replay*, and password guessing such as *dictionary attacks*. On most systems, the passwords are encrypted using a *one-way function* (OWF) before being stored on disk (see Section 3.2). A technique called *salting* is also commonly used to make dictionary attacks more difficult by expanding the search space for the adversary.

Two-stage authentication and password-derived keys address the insufficient entropy issue associated with human chosen passwords. An *n*-digit PIN verifies the user to the token, e.g., smart card, in the first stage. The token typically embeds additional secrets for use in stage two between the token and the system. A variant uses *passkeys* to map a user password to a cryptographic key using a OWF. The most secure of the weak authentication schemes uses *one-time passwords*, which addresses eavesdropping and replay attacks on password schemes.

Challenge-Response protocols fall in the class of **strong authentication** protocols, whereby authentication requires the prover to demonstrate knowledge of a secret without revealing the secret itself to the verifier. Here, the prover provides *a response* to a *time-variant challenge*, with the response inseparably bound to both the secret and the challenge. The challenge can be a random number, called a *nonce* (for 'used only once'), a sequence number or a timestamp. Time-variant parameters are countermeasures to replay attacks and certain types of chosen-text attacks because the uniqueness and timeliness guarantees allow one protocol instance to be distinguished from another. Note that *challenge-response* protocols requires some type of computing device and secure storage for long-term keying material.

Challenge-Response by Symmetric-key: Each pair of communicating parties share a secret key. In large communities, a trusted third party (TTP) can provide session keys in real time to circumvent the need to distribute n^2 key pairs. A common form of unilateral authentication uses random number(s) (RN) [18].

$A \leftarrow B: r_B$ (B generates random nonce r_B)

$A \rightarrow B: E_K(r_B, B^*)$

B generates random nonce r_B and transmits it to A (over an unsecured channel). A encrypts the

nonce and the identifier *B* using a shared secret key *K* and transmits the encrypted message back to *B*. *B* then decrypts and 1) checks that the r_B received matches the r_B sent and 2) verifies B* is equal to his own B. The shared secret *K* must be securely transmitted to *A* and *B* beforehand, typically using a mechanism involving a TTP, in order for this scheme to work.

Mutual authentication requires a second nonce r_A and a third message:

$A \leftarrow B: r_B$ (B generates nonce r_B)
$A \rightarrow B: E_K(r_A, r_B, B^*)$ (A generates nonce r_A)
$A \leftarrow B: E_K(r_B, r_A)$

Encryption ensures the nonces and identifiers are 'inseparably' bound as discussed above.

Challenge-Response using Keyed One-Way Functions: Encryption is considered a 'heavy weight' cryptographic primitive, and may be replaced by a one-way function (OWF) or a non-reversible function with shared key, and a challenge, for authentication in resource-constrained devices. The encryption algorithm E_K is replaced by a MAC algorithm h_K, i.e., a keyed hash function. The receiver also computes the MAC and compares it with the received MAC. These protocols require an additional cleartext field r_A to be transmitted [18].

$A \leftarrow B: r_B$ (B generates nonce r_B)
$A \rightarrow B: r_A, h_K(r_A, r_B, B)$ (A generates nonce r_A)
$A \leftarrow B: h_K(r_B, r_A, A)$

B confirms that the hash value received, designated as $h_k(r_A, r_B, B)$, is equal to the value he/she computes locally using the same hash function and shared secret *K*. *A* performs a similar validation using the transmitted hash $h_K(r_B, r_A, A)$ from *B.* As discussed in Section 3.2, the computational infeasibility of finding a second input to h_K that produces the same hash provides the security guarantee in this mutual authentication protocol.

Challenge-Response by Public-Key: Here, the prover decrypts a challenge using its secret key component of the public-private pair, which is encrypted by the verifier under its public key P_A. Alternatively, the prover can digitally sign a challenge.

$A \leftarrow B: h(r), B, P_A(r, B)$
$A \rightarrow B: r$

B chooses nonce *r*, computes the **witness** $x = h(r)$ (*h* is a OWF), where *x* demonstrates knowledge of *r* without disclosing it, and computes challenge $e = P_A(r, B)$. *A* decrypts *e* to recover *r'* and *B'*, computes $x' = h(r')$ and rejects if *x'* does not equal *x* or if *B'* does not equal *B*, otherwise *A* sends $r = r'$ to *B*. *B* succeeds with unilateral entity authentication of *A* upon verifying the received *r* agrees with his *r*. The witness prevents chosen-text attacks.

5. Physical Unclonable Functions (PUFs)

Components needed for information security can be implemented using physical-layer security primitives. A long-standing assumption of software-based security systems has been that hardware implementations of security primitives are trustworthy 'black boxes'. In particular, for *keyed* security primitives such as block ciphers, key generation and key storage are assumed to be trusted and secure, and operational state within black box implementations of security algorithms is assumed to be hidden and inaccessible. Unfortunately, models which assume a 'hardware root-of-trust' are becoming increasingly more vulnerable to attacks [32-34].

PUFs represent physical-layer security components that are designed to deal with threats to key generation and key storage. PUFs are circuit primitives that leverage within-die variations in ICs as a means of producing random bitstrings. Each IC is uniquely characterized by random manufacturing variations, and therefore, the bitstrings produced by PUFs are unique from one chip to the next. Cloning a PUF, i.e., making an exact copy, is nearly impossible because it would require control over the fabrication process that is well beyond our current capabilities. A PUF maps a set of digital "challenges" to a set of digital "responses" by exploiting these physical variations in the IC. The entropy in the responses is stored in the physical structures on the IC and can only be retrieved when the IC is powered up. The analog nature of the entropy source makes PUFs 'tamper-evident', whereby invasive attacks by adversaries will, with high probability, change its characteristics.

PUFs have been proposed which leverage variations in transistor threshold voltages [35-37], speckle patterns [38-39], delay chains and ROs [40-64], thin-film transistors [65], FPGAs [66-67], SRAMs [68-74], leakage current [75-76], metal resistance [77-81], transistor transconductance [82], the path delays of core logic macros [83][84-87], optics and phase change [88], sensors [89], switching variations [90], sub-threshold design [91], ROMs [92], buskeepers [93], microprocessors [94], using lithography effects [95-96], optical proximity correction [97], aging [98], in sub-threshold operation [99], memristors [100] and other non-volatile memories [101], in scan chains [102], phase change memory [103] and carbon-nanotubes [104]. Board-level authentication using PUFs has also recently been proposed [105] and for securing mobile system platforms [106-107].

5.1 PUF-Based Authentication

As mentioned above, authentication is the process between a prover, e.g., a hardware token and a verifier, a secure server, that confirms the identities, using corroborative evidence, of one or both parties. With the Internet-of-things (IoT), there are a growing number of applications in which the hardware token is resource-constrained, and therefore, novel authentication techniques are required that are low in cost, energy and area overhead. Conventional methods of authentication which use area-heavy cryptographic primitives and non-volatile memory (NVM) are less attractive for these types of evolving embedded applications. PUFs, on the other hand, are hardware security and trust primitives that can address issues related to low cost because they eliminate (in many proposed authentication protocols) the need for NVM. Moreover, the special class of so-called 'strong PUFs' can also reduce area and energy overheads by reducing the number and type of hardware-instantiated cryptographic primitives.

PUFs generate bitstrings that can serve the role of uniquely identifying the hardware tokens for authentication applications. The bitstrings are generated on-the-fly, thereby eliminating the need to store digital copies of them in NVM, and are (ideally) reproducible under a range of environmental variations. The ability to control the precise generation time of the secret bitstring and the sensitivity of the PUF entropy source to invasive probing attacks (which act to invalidate it) are additional attributes that make them attractive for authentication in resource-constrained hardware tokens.

PUF-based protocols have been proposed for applications including encryption, authentication, for detecting malicious alterations of design components and for activating vendor specific features on chips. Each of these applications has a unique set of requirements regarding the security properties of the PUF. For example, PUFs that produce secret keys for encryption are not subject to model building attacks (as is true for PUF-based authentication) which attempt to 'machine learn' the components of the entropy source within the chip as a means of predicting the complete response space of the PUF. This is true for encryption because the responses to challenges are typ-

ically not ‘readable’ from an interface on the chip. In general, the more access a given application provides to the PUF externally, the more resilience it needs to have to adversarial attack mechanisms. Authentication as an application for PUFs clearly falls in the category of extended access.

5.2 Strong vs. Weak PUFs

Weak PUFs are those whose challenge-response space is small while strong PUFs have very large, ideally exponential, challenge-response spaces [108-109]. The distinction between strong and weak PUF is rooted in the amount of entropy that each class can access. The larger the entropy source, the more difficult it is for an adversary, who has access to the PUF, to collect and analyze challenge-response pairs (CRPs) until the complete behavior of the PUF can be predicted. The SRAM PUF is an early example of a weak PUF with only one CRP [68] while the arbiter PUF is traditionally considered a strong PUF because of its exponentially large challenge space [41]. However, if the size of the entropy source is considered a defining characteristic, then the arbiter PUF would fail to meet the definition of a strong PUF because its response space is derived from a relatively small entropy source, in particular, as small as a couple hundred gates. Given this latter consideration, very few of the proposed PUFs meet this expanded definition. Model-building resistance using machine learning techniques has emerged as an important criterion for determining whether a PUF is strong based only on the size of its CRP space or whether it is *truly* strong, i.e., attacks that attempt to learn and predict its behavior are infeasible [110][42].

5.3 The Arbiter PUF

The most widely referenced strong PUF, the *arbiter PUF*, was the one of the first proposed, and is described in [41][42]. However, it is also widely recognized that it is considered strong based only on the size of its input challenge space, and not on the amount of entropy it possesses.

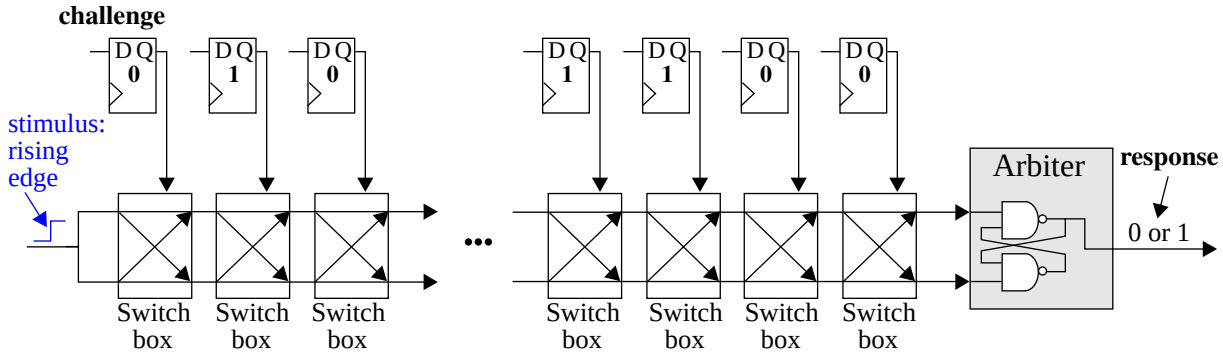


Fig. 2. Arbiter PUF [41].

The arbiter PUF measures path delays from a specialized test structure as its source of entropy as shown in Fig. 2. The test structure implements two paths, each of which can be individually configured using a set of challenge bits (stored in FFs along the top of the figure). Each of the challenge bits controls a ‘Switch box’, that can be configured in either *pass mode* and *switch mode*. Pass mode connects the upper and lower path inputs to the corresponding upper and lower path outputs, while switch mode reverses the connections. A stimulus, represented as a rising edge on the left side of the figure, cause two edges to propagate along the two paths configured by the challenge bits. The faster path controls the value stored in the *arbiter* located on the right side of the figure. If the propagating rising edge on the upper input to the arbiter arrives first, the **response** bit output becomes a ‘0’. Otherwise, the response bit is a ‘1’. The switch boxes are designed identically as a means of avoiding any type of systematic bias in the delays of the two paths[1]. Within-die process variations cause uncontrollable delay variations to occur in the switch

boxes, which in turn, makes each instance of the arbiter PUF unique in terms of its generated response bit(s). A bitstring can be obtained from the arbiter PUF by repeating the measurement process under a set of different challenges.

From this design, it is clear that the arbiter PUF has an exponential number of input challenges that can be applied, in particular, 2^n with *n* representing the number of switch boxes. However, the total amount of entropy is relatively small, and is represented by the four path segments in each of the switch boxes. For *n* equal to 128, the total number of path segments that can vary individually from one instance to another is 4*128 = 512. The exponential number of input challenges simply combine these individual sources of entropy in different ways. Model building attacks attempt to learn the delay relationships of the two configurations for each switch box [110]. Once known, the response under any challenge then becomes predictable (limited only by the noise margin of the arbiter measurement circuit).

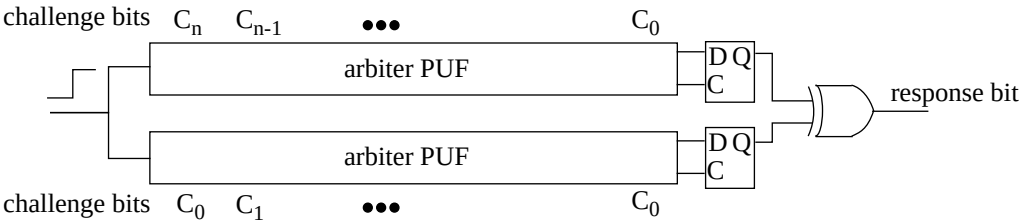


Fig. 3. XOR-mixed Arbiter PUF [44][111][112].

The model-building weakness of the arbiter PUF is addressed in follow-on work, where the outputs of *n* arbiter PUFs are XOR'ed, to create a XOR-mixed arbiter PUF [44][111][112]. Fig. 3 shows an example in which 2 arbiter PUF output bits are XOR'ed. The goal is to create an XOR network large enough to achieve the *avalanche criterion*. This criterion is commonly found in cryptographic hash and encryption functions where flipping one of the input bits (or a bit in the key for encryption) causes half of the output bits to flip. For the XOR-mixed PUF, the goal is to achieve the avalanche effect by flipping one of the challenge bits. Although this helps significantly with model building, particularly with networks of XORs greater than 4, larger XOR networks also reduce reliability by creating a *noise-based* avalanche effect, i.e., any odd number of bit flips that occur on the inputs of any given XOR network results in a response bit flip error. As reported in [111], if a single arbiter PUF has an HD_{intra} of 5% (intra-chip HD measures the PUF's ability to reproduce the same bitstring over repeated applications of the challenge, usually under different environmental conditions), the HD_{intra} increases to 19% for a 4-XOR-mixed arbiter PUF, i.e., nearly 1/5 of the response bits have bit flip errors. Therefore, error-correction using techniques described in Section 3.3 become critical to ensuring proper functional operation when used in authentication protocols.

5.4 Hardware-Embedded Delay PUF (HELP)

Similar to arbiter PUFs, the hardware-embedded delay PUF (HELP) derives its entropy from variations in path delays. However, HELP measures delays from existing functional units. Therefore, no dedicated test structures are required. Another major benefit of using existing functional units is the amount of entropy that can be potentially leveraged. Cryptographic functional units are particularly attractive because of the complexity of their interconnection networks. On the

1. Note that achieving an unbiased layout in an FPGA is a challenging and non-trivial process.

down side, the lack of control over the configuration of paths in functional units creates issues related to systematic bias and reliability, as described in the following sections.

Interestingly, the authors of the first silicon-based PUF paper describe their notion of a 'better PUF' in Ongoing and Future Work section, which turns out, based on our work, to be well founded [40]. The basic concept of measuring path delays from a core logic functional unit was implemented first by Li and Lach [83], but was not fully developed as a PUF primitive. In particular, the authors do not address the bias introduced by paths of different lengths nor do they deal with the reliability issues associated with paths that glitch.

Our development of HELP began in 2011 on a 90 nm ASIC implementation [86], but was fully developed as an intrinsic PUF (with full integration of the control logic, entropy source and measurement components) on a 130 nm Xilinx V2Pro [84-85], and more recently using a 28 nm Xilinx Zynq architecture [87]. We have developed solutions for path length bias and glitching that occur when core logic functional units are used as the source of entropy, as well as techniques that improve the attack resilience of HELP when used in low cost authentication applications. This section describes the characteristics of the most recent incarnation of HELP and presents new results.

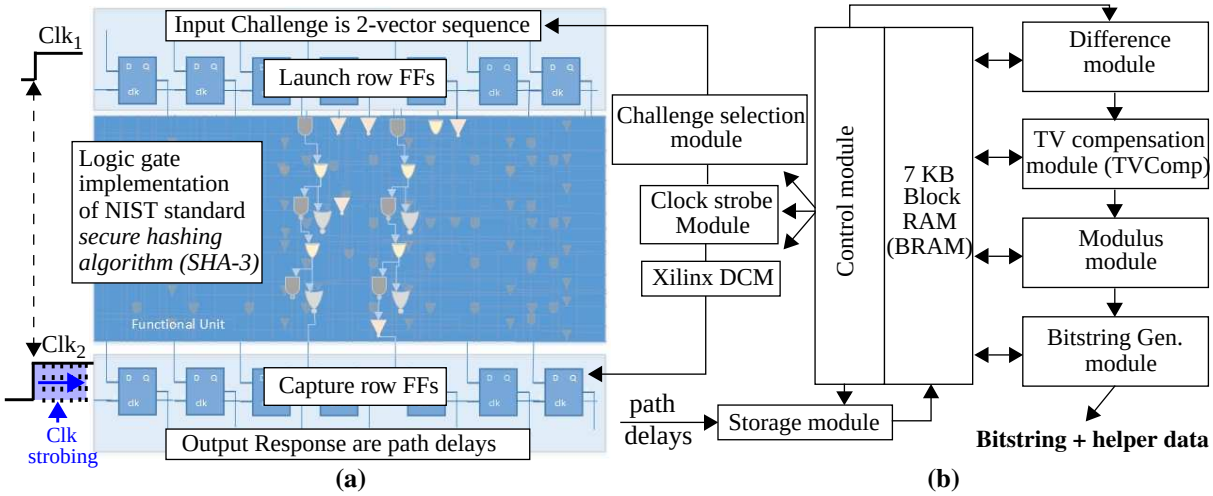


Fig. 4. HELP Block Diagram: (a) Instantiation of the HELP entropy source and (b) HELP processing engine.

The original version of HELP made use of an embedded test structure called REBEL [113] for measuring path delays and detecting glitches [84-85][86]. Recent implementations of HELP measure path delays in glitch-free functional units, which allows a simplified version of REBEL to be used [87]. The simplified version eliminates the delay chain component and instead samples the path delays at the capture FF directly.

HELP attaches to an on-chip module, such as a hardware implementation of the Secure Hashing Algorithm (SHA-3) [114], as shown on the left side of Fig. 4. The data path component of the SHA-3 algorithm, configured as *keccak-f[200]*, is used in our FPGA experiments. This combinational data path component includes 416 primary inputs (PIs) and 400 primary outputs (POs) and is implemented on a Xilinx Zynq FPGA using 1936 LUTs.

Similar to the arbiter PUF described in the previous section, within-die variations in path delays are the main source of entropy for HELP. Manufacturing variations change the relative path delays through the functional unit in different ways, and therefore, each instance of the functional unit is uniquely characterized by these delays. However, the structure of the paths in the

arbiter PUF is very different than those in a typical functional unit, i.e., the arbiter PUF paths are symmetric and regular (by design) while the paths within a typical functional unit exhibit no such regularity.

Functional unit paths exhibit *fan-out* and then *reconvergence* of fan-out at various points within the logic structure of the functional unit (called reconvergent-fanout), as shown on the right side of Fig. 5. Also, the lengths of the paths can vary widely, e.g., the *short paths* shown have 3 or fewer gates while the *long paths* are 5 or more gates in length. Both of these characteristics make it more difficult to build a PUF with good statistical characteristics. Reconvergent-fanout can cause ***glitching***, i.e., static and dynamic hazards, to occur on the primary outputs, whereby output signals transition more than once. Glitching creates ambiguity regarding the 'correct' timing value to use for the path. Operating the functional unit under different environmental conditions, e.g., temperature and supply voltage, exacerbates the problem, where paths that are glitch-free under nominal environmental conditions suddenly become glitchy under adverse conditions. Moreover, the systematic bias associated with paths of different lengths significantly degrades the statistical randomness and uniqueness characteristics of the PUF. We have developed several techniques to deal with both of these problems. Our most recent work, described here, implements the functional unit using a special *glitch-free* logic style called *wave differential dynamic logic* (WDDL) [115-116], while the systematic bias introduced by paths of different lengths is dealt with by applying a *modulus* to the digitized path delay, which effectively removes the bias.

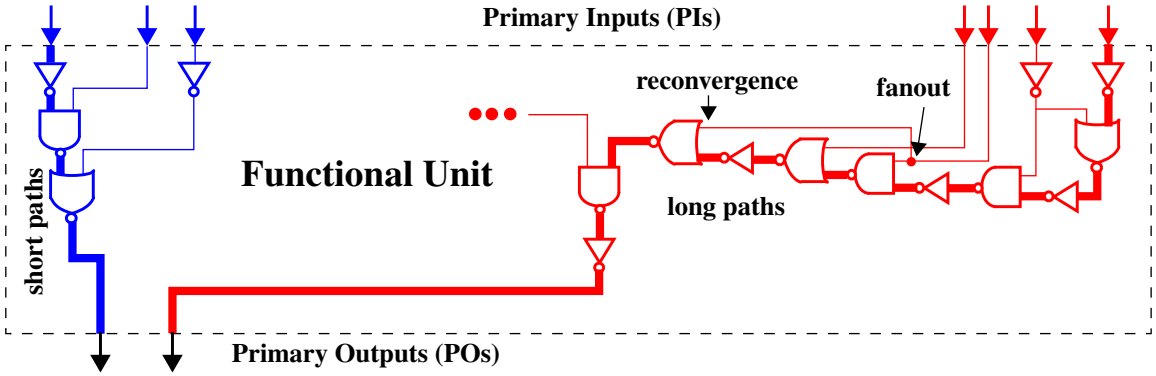


Fig. 5. Portion of a functional unit schematic, showing fan-out and reconvergence of paths, and the presence of different length (short vs. long) paths.

5.4.1 Clock Strobing

Path delay is defined as the amount of time (Δt) it takes for a set of 0-to-1 and 1-to-0 bit transitions introduced on the PIs of the functional unit (input challenge) to propagate through the logic gate network and emerge on a PO. HELP uses a clock-strobing technique to obtain high resolution measurements of path delays as shown on the left side of Fig. 4. A series of launch-capture operations are applied in which the vector sequence that defines the input challenge is applied repeatedly to the PIs using the Launch row flip-flops (FFs) and the output responses are measured on the POs using the Capture row FFs. On each application, the phase of the capture clock, Clk_2, is incremented forward with respect to Clk_1, by small Δts (on order of 20 ps), until the emerging signal transition on a PO is successfully captured in the Capture row FFs. A set of XOR gates connected to the Capture row FF inputs and outputs (not shown) provide a simple means of determining when this occurs. When an XOR gate value becomes 0, then the input and output of the FF are the same (indicating a successful capture). The first occurrence in which this occurs during the clock strobe sweep causes the current phase shift value to be recorded as the digitized delay value

for this path. This operation is applied to all POs simultaneously.

The phase shifting module for Clk_2 is shown in the middle of Fig. 4. On-chip digital clock managers (DCMs) are commonly included in FPGA architectures. For example, Xilinx FPGAs typically incorporate at least one DCM with a digitally controlled *fine phase shift* control mechanism even on their lowest cost FPGAs. For low-cost components that do not include a DCM with this capability, a fine phase shift mechanism can be implemented with a small area overhead using a multi-tapped delay chain.

The right side of Fig. 4 shows the HELP processing engine. The digitized path delays are collected by a *storage* module and stored in an on-chip block RAM (BRAM). Each digitized timing value is stored as a 14-bit value, with 10 binary digits serving to cover the fine phase shift sweep range of 0 to1023 and 4 binary digits of fixed point precision to enable up to 16 samples of each path delay to be measured and averaged. The 7 KByte BRAM allows 4096 path delays to be stored. We configure the applied challenges to test 2048 paths with rising transitions and 2048 paths with falling transitions. The 14-bit digitized path delays are referred to as PUFNums or **PN**.

5.4.2 PN Processing

Once the PN are collected, a sequence of mathematical operations are applied as shown on the right side of the Fig. 4 to produce the bitstring and helper data. The *difference* module creates unique, pseudo-random pairings between the rising and falling PN groups using two seeded linear feedback shift registers (LFSRs). The two 11-bit *LFSR seeds* are user-specified parameters. The PN differences, referred to as **PND**, are stored in the lower 2048 memory locations of the BRAM as values in the range +/- 511 with 4 binary digits of fixed point precision, overwriting the original set of rising-edge PN.

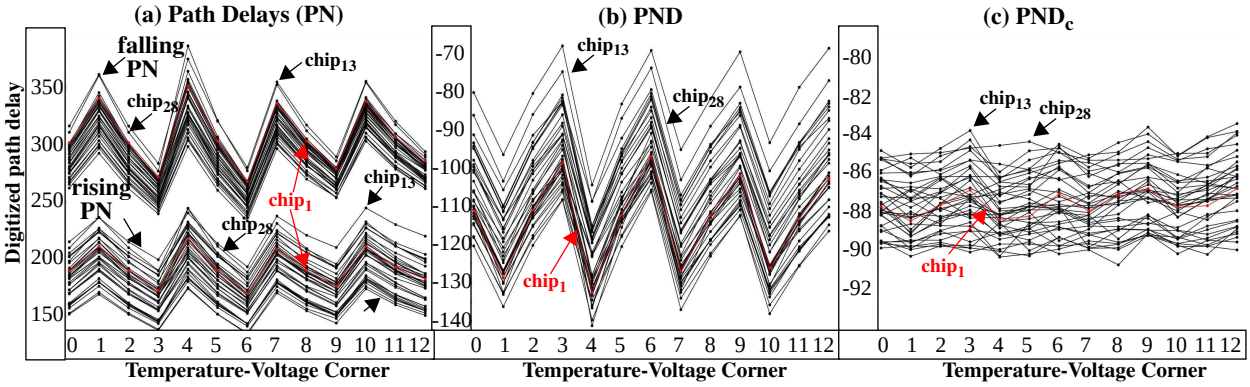


Fig. 6. (a) Example rising and falling path delays (PN), (b) PND and (c) PND$_c$.

Fig. 6(a) shows an example of this process using two groups of 38 curves, one curve for each Xilinx Zynq 7020 chip that was tested. The curves shown along the bottom depict the PN obtained from rising transition tests and those along the top are the PN from falling transition tests. The 13 line-connected points associated with each curve represent the chip's PN measured over a range of environmental conditions, called temperature-voltage (TV) corners. The PN at the x-axis position given by 0 are those measured under nominal conditions (referred to as **enrollment** values below), i.e., at 25^{o}C, 1.00V. The PN at positions 1, 2 and 3 are also measured at 25^{o}C but at supply voltages of 0.95, 1.00 and 1.05 V. Similarly, the other groups of 3 consecutive points along the x-axis are measured at these supply voltages but at temperatures -40^{o}C, 85^{o}C and 100^{o}C. The PN measured under TV corners numbered 1 to 12 are referred to as **regeneration** val-

ues. Fig. 6(b) plots the **PND** defined by subtracting point-wise, each falling PN from the corresponding rising PN for the same chip.

5.4.3 Temperature-Voltage (TV) Compensation

PUFs must be able to reproduce their bitstrings as precisely as possible, ideally without any bit flip errors, over a range of environmental conditions in which temperature and supply voltage are different from the conditions present during enrollment. No PUF construction to date is able to completely eliminate bit flip errors during regeneration, but some are more resilient to them than others. A method called temperature-voltage compensation (**TVComp** as shown on right side of Fig. 4) is proposed for the HELP PUF as a mechanism to improve its resilience to bit flip errors.

For HELP, bit flip errors occur because changes to the chip's ambient temperature and supply voltage change its path delays (called *TV noise*). TVComp applies a linear transformation to the path delay differences (PND) as a means of shifting and scaling them to a common reference. The goal is to define a transformation that eliminates the saw-tooth behavior in the curves shown in Fig. 6(b), making them as flat and straight as possible.

TVComp is applied to the entire set of 2048 PND measured for each chip at each of the 13 TV corners separately (note, Fig. 6(b) shows only one of the PND from the larger set of 2048 PND that exist for each chip and TV corner). The TVComp procedure first converts the PND to 'standardized' values. Equation (11) represents the first transformation which makes use of two constants, μ_{TVx} and $Rng_{TVx,}$ obtained from a histogram distribution of the measured PND. The second

$$\text{zval}_i = \frac{(\text{PND}_{\text{TVx}} - \mu_{\text{TVx}})}{\text{Rng}_{\text{TVx}}} \qquad \textbf{Eq. 11.}$$

$$\text{TVCPNDiff}_i = \text{zval}_i \text{Rng}_{\text{ref}} + \mu_{\text{ref}} \qquad \textbf{Eq. 12.}$$

transformation is represented by Equation (12), which translates the standardized *zvals* to a new distribution with mean μ_{ref} and range Rng_{ref}. The **reference** mean and range values are user-selectable parameters of the HELP algorithm.

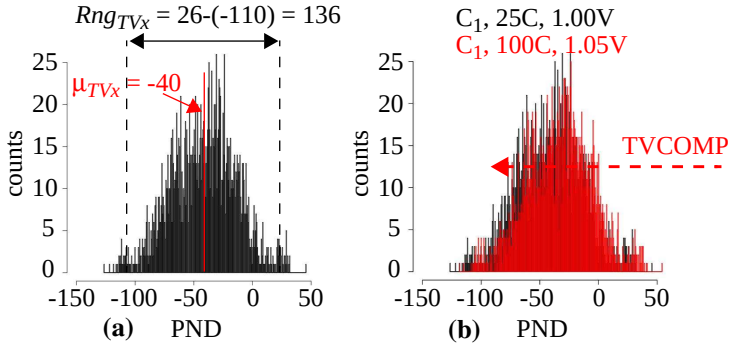


Fig. 7. (a) PND distribution for chip C_1 with μ_{TVx} and Rng_{TVx} depicted and (b) Chip C_1 PND at 2 TV corners.

As an example, Fig. 7(a) shows the PND histogram distribution for chip C_1 at 25C, 1.00V. The μ_{TVx} is shown as -40 while the Rng_{TVx} is computed between the 5% and 95% as 136. Fig 7(b) superimposes the PND histograms for C_1 at 25C, 1.00V and 100C, 1.05V. The TVComp process will shift (and scale) this distribution to the left to remove the adverse effects introduced by the change in environmental conditions.

A second illustration of the effect of TVCOMP is shown in Figs. 6(b) and 6(c). The data in Fig. 6(c) is obtained by applying TVCOMP procedure to the 2048 PND measured under each of

the 13 TV corners for each chip, i.e., 13 TV corners * 38 chips = 494 separate applications. Since the same reference mean and range are used for all transformations, TVComp eliminates both TV noise and chip-wide performance differences between the chips. Note that the curves in Fig. 6(c) no longer exhibit the saw-tooth behavior introduced by TV noise[1].

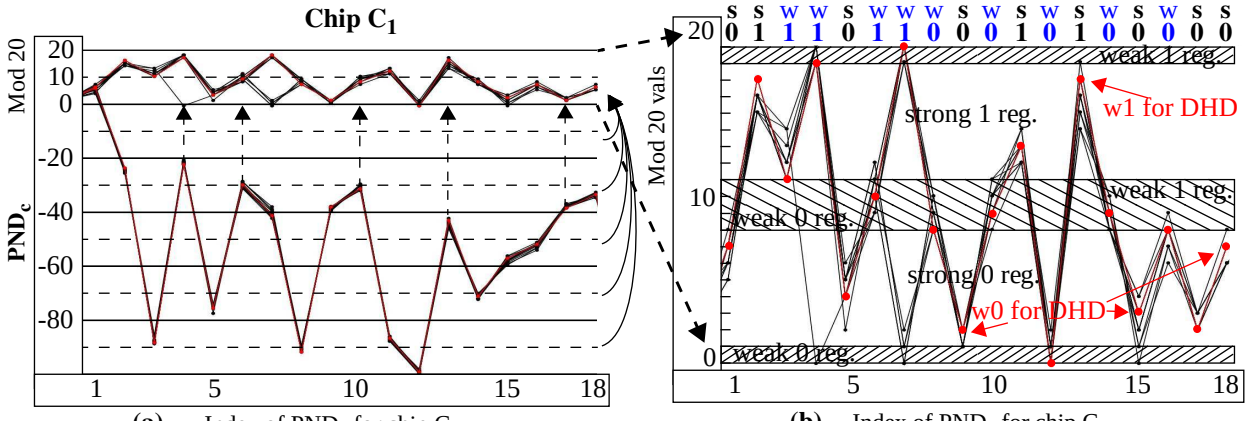
The differences that remain in the TVCOMP'ed PND (subsequently referred to as **PND_c**) shown in Fig. 6(c) are those introduced by WDV and *uncompensated* TV noise (**UC-TVNoise**). For this particular PND, the TVCOMP process is able to reduce TV noise to approx. 2 in the worst case, which translates to approx. 36 ps. In general, PND_c with larger levels of UC-TVNoise are more likely to introduce bit flip errors.

The implementation of the HELP algorithm shown in Fig. 4 constructs a histogram distribution in the upper 2048 memory locations of the BRAM using the 2048 PND stored in the lower portion and then parses the distribution to obtain μ_{TVx} and Rng_{TVx}. Once the distribution constants are available, the PND in the low portion of the BRAM are converted to PND_c.

The last operation applied to the PN is represented by the *Modulus* operation shown on the right side of Fig. 4. Modulus is a standard mathematical operation that computes the positive remainder after dividing by the modulus. The Modulus operation is required by HELP to eliminate the path length bias that exists in the PND_c, which acts to reduce randomness and uniqueness in the generated bitstrings. The value of the Modulus is also a user-selectable parameter, similar to the LFSR seed, mean and range parameters, and is discussed further in the following. The HELP engine shown in Fig. 4 overwrites the PND_c after applying the Modulus. The final values, called **$MPND_c$**, are used in the bitstring generation process.

5.4.4 Bit Generation Algorithm

The bitstring generation process uses a fifth user-specified parameter, called the *Margin*, as a means of improving the reliability of the bitstring regeneration process. The bottom portion of Fig. 8(a) plots 18 of the 2048 PND_c from $Chip_1$ along the x-axis. The red curve line-connects the data points obtained under enrollment conditions while the black curves line-connect data points under the 12 regeneration TV corners.



(a) Index of PND_c for chip C_1 **(b)** Index of PND_c for chip C_1

Fig. 8. Strong/Weak PND_c classification using margining.

The curves plotted along the top of Fig. 8(a) show the $MPND_c$ values after a modulus of 20 is

1. TV compensation also serves as a countermeasure to prevent adversaries from manipulating temperature and supply voltage as a physical attack mechanism.

applied. Fig. 8(b) enlarges the upper portion of Fig. 8(a) and includes a set of margins of size 2 surrounding two strong bit regions of size 6. Designators along the top given as ‘s0’, ‘s1’, ‘w0’ and ‘w1’ classify each of the enrollment data points as either a strong 0 or 1, or a weak 0 or 1, resp. Data points that fall on or within the hatched areas are classified as weak as a mechanism to avoid bit flip errors introduced by UC-TVNoise that occurs during regeneration.

The Margin method improves bitstring reproducibility by eliminating data points classified as ‘weak’ in the bitstring generation process. For example, the data points at indexes 4, 6, 7, 8, 10 and 14 would introduce bit flip errors at one or more of the TV corners during regeneration because at least one of the regeneration data points is in the opposite bit value region from the corresponding enrollment value. We refer to this bitstring generation technique as the **Single Helper Data** (SHD) scheme since the classification of the $MPND_c$ as strong or weak is determined solely by the enrollment data.

A second technique, referred to as the **Dual Helper Data** (DHD) scheme, requires that both the enrollment and regeneration $MPND_c$ be in strong bit regions before allowing the bit to be used in the bitstring during regeneration. The *helper data*, which represents the classification of the $MPND_c$ as strong or weak, is bitwise ‘AND’ed, and then both the enrollment and regeneration bitstrings are generated (the enrollment data is assumed to be collected earlier in time and stored on a secure server). The DHD scheme doubles the protection provided by the margin against bit flip errors because the $MPND_c$ produced during regeneration must now change and move across both a ‘0’ and ‘1’ margin before it can introduce a bit flip error. This is true because both the enrollment and regeneration $MPND_c$ must be classified as strong to be included in the bitstring and the strong bit regions are separated by 2*margin.

Fig. 8 highlights four cases where an enrollment-classified strong bit would be reclassified as weak in the DHD scheme because 1 or more of the regeneration PND_c falls within a weak region. This shows that in addition to doubling the protection against bit flip errors, the DHD scheme can potentially produce different bitstrings each time the chip regenerates it. Therefore, **DHD increases entropy** by leveraging UC-TVNoise (and sampling noise to a smaller degree). This feature is a benefit for authentication applications because only half of the helper data is revealed to the adversary while the other half is generated and kept on the chip or server. The missing helper data adds uncertainty for an adversary as to the final form of the bitstring. Encryption applications can leverage both of these DHD benefits as well by exchanging the chip and server helper data bitstrings while keeping the generated keys private. These benefits of DHD are expanded upon in the following sections.

5.4.5 Entropy Analysis

The Margin technique using either the SHD or DHD schemes adds uniqueness to the regenerated bitstring. This is true because weak bits are excluded from the bitstring based on the position of the PND_c and Margins and therefore, different chips utilize different bits in the constructed bitstring. Figs. 9(a) and (b) depict several scenarios that show how the Margin and the position of the PND_c affect bitstring generation. The line-connected curves in Fig. 9 are analogous to those described earlier in reference to Fig. 6(c). Fig. 9(a) plots a set of 20 different PND_c to illustrate how PND_c distribute across the range defined by the Modulus, which is set to 20. Fig. 9(b) is a blow-up of the bottom portion of Fig. 9(a).

As indicated earlier, within-die process variations change path delays uniquely in different chips, which is reflected by the y-dimensional spread within each group of PND_c. For the data set

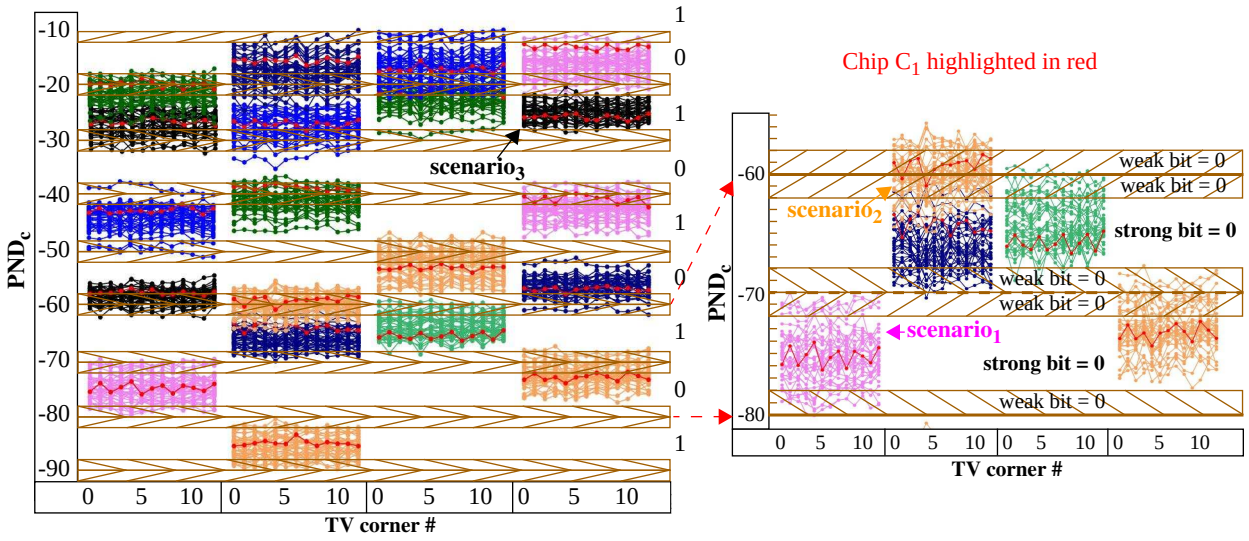


Fig. 9. (a) Example PND_c (20 groups) from 38 chips (y-axis) across 1 enrollment and 12 TV corners (x-axis), and (b) blow-up of -60 to -80 region.

labeled as $scenario_1$ in Fig. 9(b), the range occupied by the PND_c is approx. 10. The y position of the overall data set is such that, except for a few points, the bit generated by this data will be 0 for all 38 chips.

However, the enrollment data points (left-most) for some chips fall within the weak bit regions and therefore, this bit is skipped for these chips using either the SHD or DHD schemes. Moreover, UC-TVNoise causes some of the regeneration data points to move from their strong bit positions in the enrollment data to weak bits during regeneration. The DHD scheme excludes this bit for these chips as well, creating differences in the generated bitstring for the same chip at different TV corners, while simultaneously providing a 2x Margin to bit flip errors. Moreover, the relative position of the curve associated with each chip, with respect to the other chips, changes in each data set so it is unpredictable which data points are excluded during bitstring generation for any particular chip. The curve for chip C_1 is highlighted in red in each of the PND_c groups to illustrate the change in its relative position with respect to other chips in the group.

The data set labeled $scenario_2$ in Fig. 9(b) shows a second possibility, that is closest to the 'ideal' case because the position and range of the curves spans the y-axis into both the strong 0 and strong 1 bit regions. The number of possible results regarding the status of the bit includes those described for $scenario_1$ plus an additional possibility that some chips generate a strong 1 bit and others a strong 0 bit. In contrast, $scenario_3$ labeled in Fig. 9(a) is closest to the 'worst' case where nearly the entire data set is positioned with the strong 0 region. Note that this scenario is only possible when the Modulus is large enough to create strong bit regions that upper-bound the smallest range (WDV + UC-TVNoise) found among the $MPND_c$ groups. Generating bitstrings with Moduli larger than 4*Margin + this smallest range begins to reduce their statistical quality. The analysis presented in subsequent sections shows that the upper-bound for this data set is Modulus = 28.

5.4.6 Statistical Analysis of the Bitstrings

The bitstrings generated using the DHD scheme are subjected to the NIST statistical test suite as well as Inter-chip and Intra-chip hamming distance (HD) tests. The analysis is carried out using two different *reference* scaling factors for TVCOMP, referred to as *minimum* (Min) and *mean*

scaling. The μ_{ref} and Rng_{ref} scaling constants derived from the set of path distributions for the 38 chips are used as the reference values in Equation 12 to scale all chip data before applying the Modulus operation and DHD bitstring generation procedures described above. The *minimum* scaling constants are derived from the chip with smallest distribution, i.e., smallest mean and range values. The *mean* scaling constants are computed from the average mean and range values across the distributions of all chips. We focus our analysis on these two scaling factors because they represent the extremes of the recommended range. We expect similar results to be produced for all scaling factors between these limits.

We use the acronym *SBS* to denote 'strong bitstring'. The DHD scheme requires two helper data bitstrings from the same chip as a means of constructing the two corresponding SBS's. The helper data bitstings, which are derived from the 2048 $MPND_c$ using the Margin technique, are bitwise AND'ed and then used to select bits for use in the construction of the SBS's. The SBS's generated using enrollment data (TV_0) and the nominal regeneration TV corner data (TV_2) from the same chip are used in the NIST statistical tests and Interchip hamming distance (HD_{Inter}) calculations below. UC-TVNoise is smallest using this combination, and therefore, it represents the worst case condition where the affect of the helper data AND'ing has the smallest impact on the additional entropy as discussed earlier. Only one of the SBS's from each chip is used in HD_{Inter} and NIST statistical tests, and the SBS's are truncated to the length of smallest bitstring among the 38 generated. The same criteria are used in the Intra-chip HD (HD_{Intra}) calculations except a much larger set of bits are processed by accumulating the results across a set of 256 different LFSR seeds (only one LFSR seed is used for NIST and HD_{Inter} tests because similar results are obtained using other seeds).

NIST Statistical Test Results:

The NIST statistical test results are shown in Fig. 10(a) and (b) for minimum and mean scaling, resp. A test is considered 'a pass' according to the NIST criteria if at least 35 of the 38 chips pass the test individually. The histogram bar heights indicate the number of chips that pass the test. The bitstrings generated using a Margin of 3 and a set of Moduli between 14 and 30 are subjected to 10 of the NIST tests. The size of the bitstring was too small for some values of the Modulus and therefore, the bar heights for these NIST test results are set to 0 (includes regions along back and left side of the 3-D histogram).

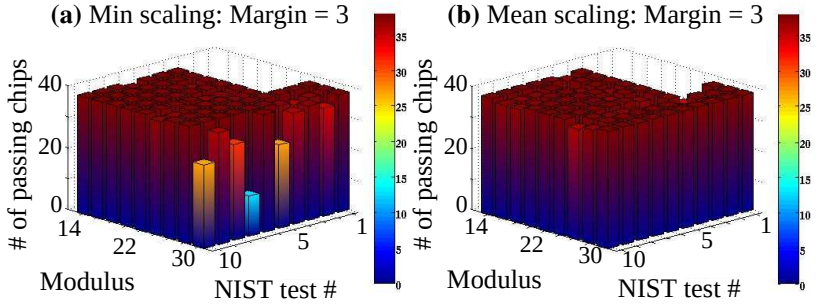


Fig. 10. NIST statistical test results using 38 chip bitstrings for each analysis and (a) Minimum scaled data and (b) Mean scaled data.

Under minimum scaling, all NIST tests are passed except for four associated with Modulus 30. These fails are related to $scenario_3$ discussed in reference to Fig. 9, where the range of within-die variation fits entirely within the strong '0' or '1' regions defined by Modulus. This is supported by the results presented under the mean scaling, where the bitstrings for Modulus 30 pass

all tests (only 1 test is failed under mean scaling, and with a value of 34 instead of 35). Mean scaling enlarges the y-dimensional spread of the data points over minimum scaling and reduces the probability that scenario$_3$ occurs. These results indicate that the bitstrings posses a high degree of randomness, which is a necessary condition for classifying the bitstrings as cryptographic quality. The results using Margins of 2 and 4 are very similar.

Interchip Hamming Distance (HD_{Inter}):

HD_{Inter} is computed using Equation 13. The symbols *NC*, *NB* and *NCC* represent 'number of chips', 'number of bits' and 'number of chip combinations', resp. This equation simply sums all the bitwise differences between each of the possible pairing of chip SBS's (*NCC*), and then converts the sum into a percentage by dividing by the total number of bits that were examined. The XOR operator generates a 1 when the pair of bits in the SBS's at the same position are different and 0 otherwise.

$$HD_{inter} = \left(\frac{1}{NCC \times NB} \sum_{i=0}^{NC} \sum_{j=i}^{NC} \left(\sum_{k=0}^{NB} (SBS_{i,k} \oplus SBS_{j,k}) \right) \right) \times 100 \qquad \textbf{Eq. 13.}$$

Fig. 11(a) shows the HD_{inter} results for a set of Moduli (x-axis) and Margins (y-axis). The ideal value for HD_{inter} is 50%, which indicates that half of the bits in any arbitrary pairing of bitstrings from the 38 chips have different values. The best values are produced for smaller Moduli, as expected. However, all values remain above 48.5%, which indicates a high degree of uniqueness among the bitstrings from different chips.

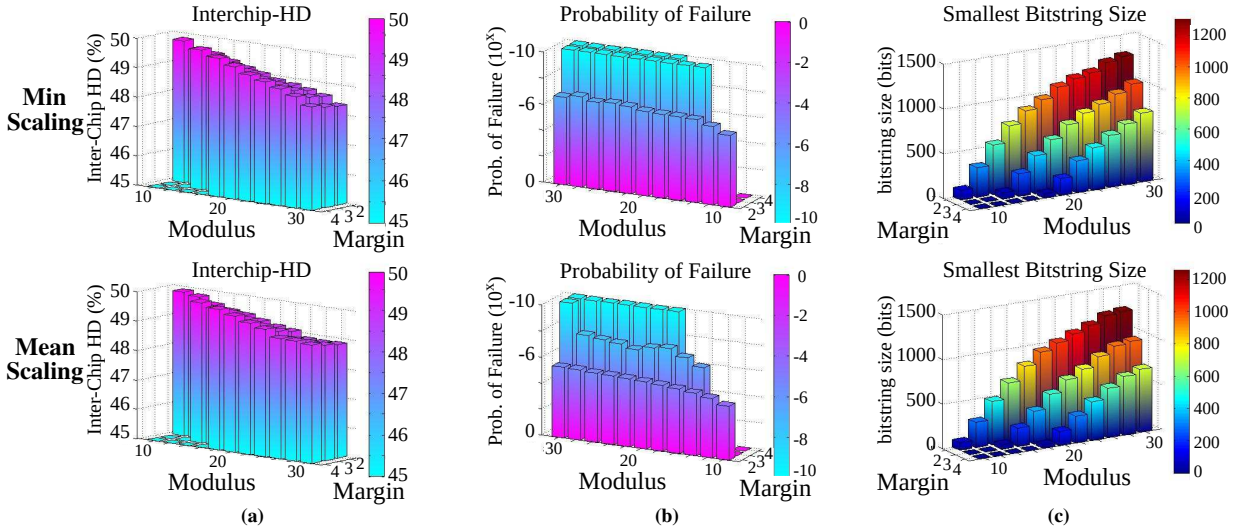


Fig. 11. (a) Interchip hamming distance (HD), (b) Probability of failure and (c) Smallest bitstring size statistics using 4096 PN.

Intrachip Hamming Distance (HD_{Intra}):

HD_{Inter} is computed using Equation 14. The symbols *NS*, *NC*, *NB* and *NT* represent 'number of seeds', 'number of chips', 'number of bits' and 'number of TV corners', resp. As indicated earlier, we repeat the HD_{intra} analysis for 256 different LFSR seeds as a means of increasing the number of bits used in the analysis. *NT* is 12 to represent each of the TV corners used to compute the pair of chip SBS's under the DHD scheme. This equation sums all the bitwise differences between each of the enrollment SBS (TV_0) bitstrings and the 12 corresponding SBS bitstrings

from the remaining TV corners for each chip and each LFSR seed, and then converts the sum into a percentage by dividing by the total number of bits that were examined. The value for *N* varied between approx. 12 million for Modulus 10 to more than 165 million for Modulus 30.

$$HD_{intra} = \left(\frac{1}{N} \sum_{z=0}^{NS} \sum_{i=0}^{NC} \sum_{j=0}^{NB} \sum_{k=1}^{NT} (SBS_{z,i,j,0} \oplus SBS_{z,i,j,k}) \right) \times 100 \qquad \textbf{Eq. 14.}$$

Fig. 11(b) reports HD_{Intra} as the probability of a bit flip failure for the same set of Moduli and Margins used in 11(a) (note the x-axis is reversed from that shown in Fig. 11(a)). The value of the exponent *x* is reported from the equation $1/10^{-x}$ so -6 indicates 1 chance in 1 million. Cases where no bit flips were detected as shown as -10. As expected, the larger Moduli produce lower probabilities of failure. The probability of failure for Margins 3 and 4 under minimum scaling are all set to 10^{-10} (no bit flip errors were detected), and are less than 10^{-6} for Margin 2 except for Modulus 10. The probability of failure under mean scaling are larger but remain below 10^{-6} for Margins 3 and 4.

Minimum Bitstring Size:

Fig. 11(c) plots the smallest bitstring size for the same set of Moduli and Margins. Smaller Moduli have smaller strong bit regions for a given Margin and therefore, fewer bits quality as strong. However, the bitstring sizes grow quickly, with at least several hundred bits available for Moduli/Margin combinations with strong bit regions of size 2 and larger. Bitstring size can be increased as needed by increasing the number of tested paths beyond 4096.

5.4.7 Security Property Analysis

In this section, we investigate several important security properties of HELP that relate to its resistance to model building and to the number of bitstrings that each token can generate using the five user-defined parameters described earlier and a sixth parameter called the *Path-Selection-Mask* (which is discussed below and in Section 6.6 as it relates to proposed authentication protocol).

Parameter-Based Bitstring Diversity:

Due to the interaction of the user-defined parameters, we present a conservative lower-bound estimate on the number of possible parameter combinations, i.e., those that ensure the generated bitstrings are random, reliable and unique for each token. Note that the source of entropy is fixed in this analysis to a set of 4096 PN (in contrast to the analysis that includes the *Path-Selection-Mask* parameter as described in the next section). In other words, the set of five user-defined parameters, namely, μ, *Rng*, *Modulus*, *Margin* and the *LFSR seeds*, apply different transformations to the same set of PN as a means of achieving bitstring diversity. As noted earlier, the two *11-bit LFSR seed* parameters allow any of the 2048 rising edge PN to be paired with any of the 2048 falling edge PN, yielding 4,194,304 possible combinations. From the results shown in Fig. 11, the number of combinations of *Margins* and *Moduli* that yield high reliability ($< e^{-6}$) is 12 (using Moduli from 16-28 for Margin 3, and 20-28 for Margin 4, in steps of size 2). The number of different μ and *Rng* parameters is conservatively estimated to be 10 each. Therefore, a total of 4,194,304 * 12 * 10 * 10 ~= 5 billion combinations of these five user-defined parameters are possible. This lower bounds the amount of effort required by an adversary in possession of the token to read out all the possible response bitstrings. The probability of achieving this lower bound is nearly zero in practice because, in the proposed protocol, the token and server generate nonces that are used to select values of the parameters and therefore, the adversary does not have direct

control of the token's interface (details covered in Section 6.6).

***Path-Selection-Mask*-Based Bitstring Diversity:**

Unlike the parameter-based scheme, bitstring diversity introduced by the *Path-Selection-Mask* is based on changing the underlying entropy components. In other words, the 4096 PN are not fixed, but vary from one authentication to the next. In the protocol proposed in Section 6.6, path selection is performed by the server using a random number generator. Path selection involves choosing a subset *x* of *y* timing values from those produced simultaneously by the challenge. For example, assume that a challenge vector sequence produces 200 timing values and the server selects a random subset of 50. The number of ways of choosing 50 from 200 is a very large number and is given by Equation 15. This number is then multiplied by the number of vectors required

$$\text{Path-select-combos} = \binom{200}{50} = 4.5e47 \qquad \textbf{Eq. 15.}$$

to reach 4096 PN (as an example, we use 82 in our recent experimental evaluation). Therefore, the number of possible bitstrings using the *Path-Selection-Mask* is exponentially related to the number of simultaneously sensitized paths produced by a challenge and the number of PN randomly selected. More importantly, the *Path-Selection-Mask* changes the characteristics of the PND distribution, which in turn impacts how each PND is transformed through the TVComp process. In other words, even with all 5 user-defined parameters held constant, the bit value generated by a $MPND_c$ will vary because its value depends on all of he 4096 PNs selected and used in the bitstring generation process. This complex relationship is leveraged as a security property in the HELP authentication protocol as a means of both preserving privacy and adding resilience to model-building attacks.

6. PUF-Based Authentication Protocols

The tamper-evident and unclonable characteristics of PUFs can be leveraged in authentication protocols to generate nonces and repeatable random bitstrings, to provide secure storage of secrets, to reduce costs and energy requirements and to simplify key management. Although weak PUFs have been proposed for authentication as described in the examples that follow, they increase the number and type of cryptographic primitives required on the token. Strong PUFs provide a distinct advantage by eliminating some of these cryptographic primitives while providing higher resistance to protocol attacks.

The cryptographic primitives required in an authentication protocol depend on the security requirements. For example, in the simplest form, the protocol can be designed to provide unilateral, e.g., server-based, authentication as discussed in Section 4. More advanced features such as mutual authentication and privacy-preserving protocols, i.e, those that prevent token tracking, require additional cryptographic primitives and message exchanges.

Entity authentication requires the prover (hardware token) to provide both an identifier and corroborative and timely evidence of its identity, e.g., a secret, that could only have been produced by the prover itself. From Section 4, PUFs carry out user authentication under the general model of 'something you possess', e.g., a hardware token such as a smart card, which in turn, incorporate silicon-based fingerprint-like identities for authentication to a secure server, such as a bank. Bear in mind that PUFs do not address the task of identifying the user to the token. As discussed in Section 4, user-token authentication is layered on top of token-server authentication using passwords, PINs, actual human fingerprints, etc.

Although passwords, PINs, one-time passwords, etc. can be used for token-server authentication, they are considered weak authentication methods. The strong authentication methods

described in Section 4 are based on a challenge-response mechanism but implicitly require the prover *A* to demonstrate knowledge of a *secret* known to be associated with prover *A* without revealing the secret itself to the verifier *B*. The challenge-response component provides a mechanism to enable the prover to maintain the secret while allowing, in the composition of exchanged messages, the prover to demonstrate its knowledge to the verifier. In order to ensure certain security properties, the random numbers (*nonces*) that are cryptographically bound to the secret and exchanged must have sufficient entropy. Cryptographic functions such as one-way hash functions, symmetric key encryption algorithms (for MACs), and public-private encryption algorithms (for digital signatures) may also be required. PUFs can certainly be used in these types of traditional authentication schemes, e.g., for generating nonces with sufficient entropy (which we discuss below), but the large number of CRPs available in strong PUF implementations also allow for simpler schemes with stronger security properties.

6.1 Protocol 1: Strong PUF with Unprotected Interface

The simplest mechanisms called *challenge-response entity authentication*, as proposed in [38][40][120-121], exchange cleartext bitstrings directly, thereby eliminating area/energy-expensive cryptographic primitives associated with traditional schemes. A PUF whose inputs and outputs can be accessed directly, as in this scheme, is said to have *unprotected interfaces*. The protocol is shown graphically in Fig. 12 (referred to as *naive* in [121]), and consists of two phases:

- Enrollment: A process carried out in a secure environment between a token, *A* and verifier, *B*. Verifier *B* generates a sequence of randomly-chosen challenges, c_i, which are presented to token *A* and applied to the PUF, and the PUF responses, r_i are then recorded in a secure database as challenge-response pairs, crp_i, along with a unique identifier, ht_{ID} for the token.
- Authentication: Token *A* requests authentication by transmitting its ID, ht_{ID}, to the verifier *B*. Verifier *B* selects one or more challenges from the database using the ht_{ID} and transmits them across an unsecured channel to the fielded token. Token *A* applies c_i to the PUF to generate r_i', which is transmitted to *B* for verification. *B* compares r_i with r_i' and *accepts* if the two bitstrings match with a tolerance, HD_{intra}, and *rejects* otherwise. Verifier *B* removes the crp_i from the database as a countermeasure to replay attacks.

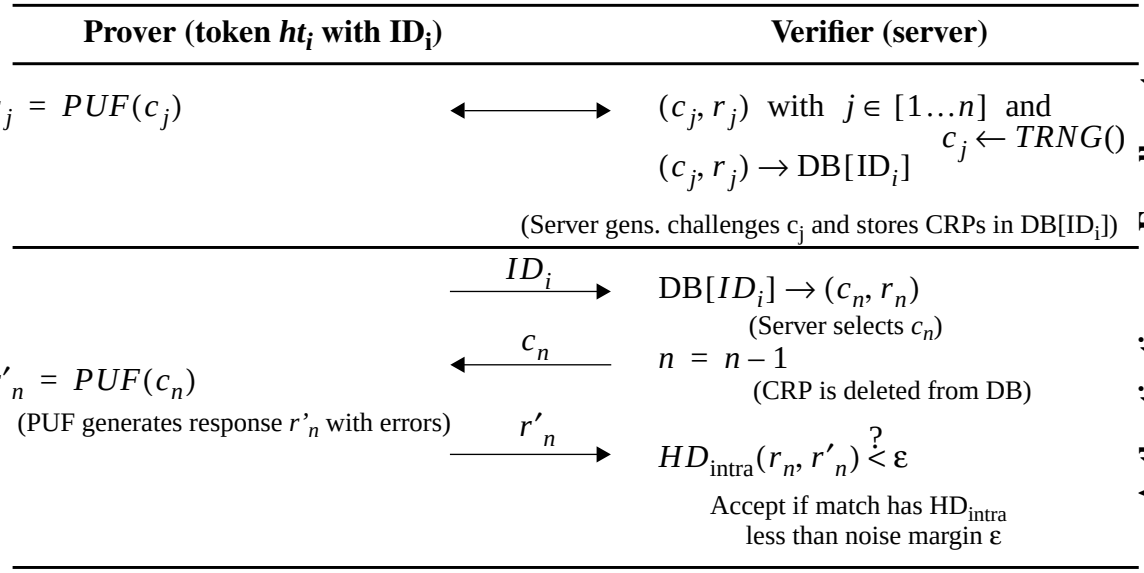


Fig. 12. Naive Strong PUF Authentication [38][121].

The protocol has the benefit of being simple to implement and is very lightweight for the token. The inability of the PUF to precisely reproduce the response r_i (in simple schemes that do not attempt error correction or error avoidance) makes it necessary to implement a error-tolerant matching scheme with $HD_{intra} > 0$. It should be noted however that large values of HD_{intra} increase the chance of impersonation, and act to reduce the strength of the authentication scheme. A second drawback is the large number of challenge-response pairs that must be recorded during enrollment, as a means of ensuring that authentication can be carried out over a long period of time. This increases the storage requirements for the verifier, since the worst-case usage scenario must be accommodated, and/or creates inconveniences for users who exceed the stored CRP capacity. Other drawbacks include the lack of resistance to *denial of service* attacks, whereby adversaries purposely deplete the server database, the inability to carry out privacy-preserving or mutual authentication and the susceptibility of the scheme to model-building attacks [122]. The latter is the primary driver for the requirement that a *truly* strong PUF be used for authentication protocols with unprotected interfaces, of which this simple protocol is an example.

A growing list of proposed protocols address these short-coming by incorporating cryptographic primitives on the prover and verifier side [19][21][39-40][123]. The inclusion of cryptographic primitives enable significant improvements to the security properties of the protocols, and additionally allow for privacy-preserving and mutual authentication. However, their use, in many cases, requires error-free response bitstrings from the PUF, which in turn requires *helper data* to be stored with the CRPs on the server. Many recent protocols target low-cost, resource-constrained applications, e.g., RFID, and attempt to minimize the implementation footprint and energy profile on the token side. Error correction algorithms, such as *secure sketches* [26-27], are asymmetric in terms of their computational cost, with helper data generation requiring fewer resources than the process of using the helper data to correct bit flip errors in the regenerated response. Recently proposed authentication protocols attempt to minimize the area and energy requirements for token-side operations by leveraging this asymmetrical relationship. We discuss several of these protocols below. An excellent review of these and other protocols [28][38][40][124-137] is provided in [121][138].

6.2 Protocol 2: Controlled PUF

The most straightforward countermeasure to model building attacks is to protect the challenge-response interface to the PUF using cryptographic hash function(s) [16**][**121**]**. One possible implementation of the protocol proposed for a *Controlled PUF* is shown in Fig. 13. The hash of the challenge prevents *chosen-challenge* attacks. This is true because the hash is a one-way-function (OWF), which makes it computationally infeasible for the adversary to control the composition of the challenge applied to the PUF. Similarly, by hashing the output of the PUF, correlations that may exist among different challenges are obfuscated, increasing the difficulty of model-building even further. The main drawback of using a OWF on the PUF responses as shown is a requirement that the responses from the PUF be error-free. This is true because even a single bit flip error in the PUF's response changes a large number of bits in the output of the OWF (avalanche effect). The functions *Gen* and *Rep* are responsible for error-correcting the response, using algorithms that were described earlier in Section 3.3.

The protocol works as follows. During enrollment in a secure environment, a one-time interface is used to allow the server to obtain PUF responses, r_j, produced from randomly generated, hashed challenges c_j. The *Gen* routine produces helper data hd_j for each r_j, which is sent to the token to produce a hashed version of the PUF response, r'_j. The 3-tuples $<c_j, r'_j, hd_j>$ produced

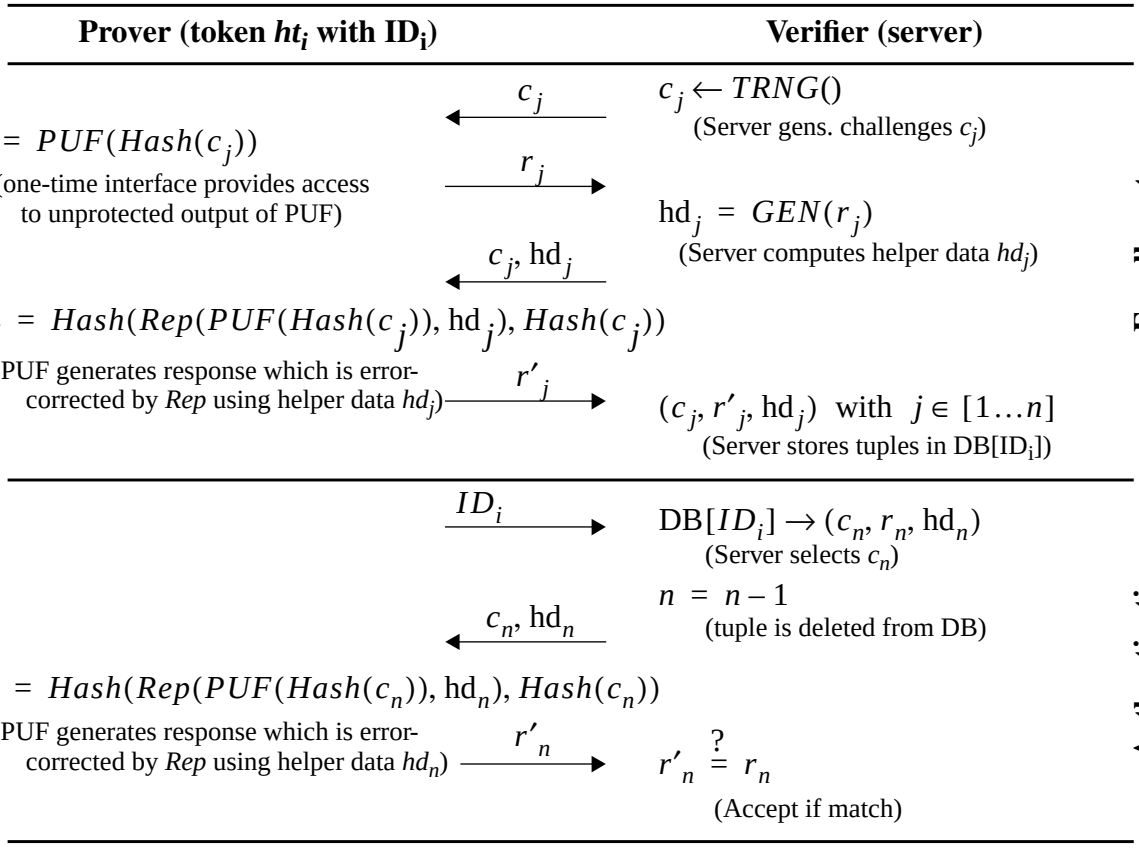


Fig. 13. Controlled PUF [16][121].

by multiple iterations of this algorithm are stored in the database for token ht_{ID}. After enrollment, a fuse is blown to disable the one-time interface. Authentication is very similar except for the *Gen* operation. Note that the response r'_n must match the stored response r_n in order for the authentication to succeed, i.e., error-correction eliminates the need for the 'fuzzy matching' component in Protocol 1. Otherwise, the benefits and drawbacks are similar as those described for Protocol 1 with additional drawbacks related to the need for a cryptographic hash function and the increased computational and energy cost associated with *Rep*.

6.3 Protocol 3: Reverse Fuzzy Extractor

Maes et al. proposes a protocol based on *reversed secure sketching* that is designed to address authentication in resource-constrained environments [19][123]. Their protocol uses the syndrome technique proposed in [26] (see Section 3.3) for error correction but reverses the roles of the prover and verifier, i.e., the prover (resource-constrained token) performs the lighter-weight *Gen* procedure while the verifier (server) performs the compute-intensive *Rep* procedure. The same process is carried out during enrollment and regeneration. Given that the sketching procedure produces a unique bitstring with bits that are different every time it is executed on the token, in order to authenticate, the verifier is required to *correct the original bitstring* stored during enrollment to match each of the regenerated bitstrings. In order to accomplish this, the helper data produced by each run of *Gen* on the token is transmitted to the verifier.

The mutual authentication protocol proposed in [19] is graphically illustrated in Fig. 14. Similar to previous protocols, enrollment involves the verifier generating challenges and storing the PUF responses r_i for ht_i in a secure database (not shown). In the proposed protocol, *only a single*

CRP is stored for each token, which is indexed by ID_i in the server's database, and then this interface is permanently disabled on the token. The authentication process begins with the token on the left generating the bitstring response again as r'_i and then multiplying it by the parity-check matrix $\mathbf{H^T}$ of the syndrome-based linear block code to produce the helper data hd_i. A random number generator is used to produce nonce n_1 that is exchanged with the verifier as a mechanism to prevent replay attacks (see Section 4 for expository on traditional challenge-response authentication). The tuple ID_i, hd_i and n_1 is transmitted over an unsecured channel to the verifier,

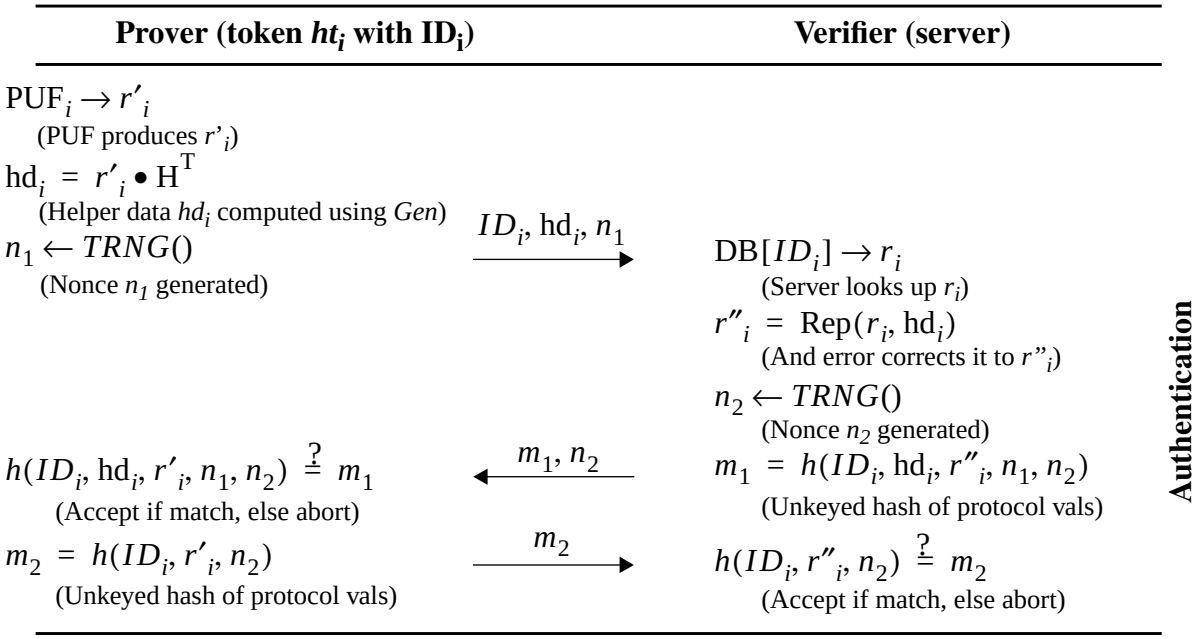


Fig. 14. "Reversed secure sketching" mutual authentication protocol proposed in [26].

The verifier looks up the response bitstring r_i generated by this token during enrollment in the secure database and invokes the *Rep* routine of the secure sketch error correction algorithm with r_i and the transmitted helper data hd_i. If the PUF response r'_i and corresponding helper data hd_i are within the error-correcting capabilities of the secure sketch algorithm, the output r''_i of *Rep* will match the r'_i generated by the token. A second nonce, n_2, is generated to enable secure mutual authentication (see Section 4) and a secure *hash* is applied to the ID_i, helper data hd_i, the regenerated response bitstring r''_i and both nonces n_1 and n_2 to produce m_1. The hash m_1 conveys to the token that the server has knowledge of the response r'_i, which allows the token to authenticate the server. This verification is carried out by the token by hashing the same values, except using its own version of r'_i and comparing the output to the transmitted m_1. If a match occurs, then r'_i must be equal to r''_i, and the token *accepts*, otherwise authentication of the server fails. The token then demonstrates knowledge of r'_i by hashing it with its ID_i and nonce n_2 and transmitting the result m_2 to the server. The server then authenticates the token using a similar process by comparing its result with m_2.

The helper data in this 'reverse' implementation of the fuzzy extractor changes from one run of the protocol to the next, based on the number and position of the bits that flip during each regeneration. The main drawbacks of the proposed scheme are that it is not privacy-preserving

and assumes that the helper data does not leak any information about the response r_i. Moreover, since most PUFs can reliably reproduce more than 80% of the secret bitstring, any correlations that occur in the helper data bitstrings introduced by these 'constant' secret bitstring components may reveal information that the adversary can use to increase the effectiveness of reverse-engineering attacks.

6.4 Protocol 4: Slender PUF Protocol

Majzoobi et al. proposed an authentication protocol [137] based on substring matching [112], again designed to address authentication in resource-constrained environments. Their protocol eliminates all types of cryptographic functions on the token, including hashing and error correction functions. The proposed protocol is demonstrated using a 4-XOR arbiter PUF, a variant of the arbiter PUF shown in Fig. 3, in which the output of 4 copies of the arbiter PUF are XOR'ed as a mechanism to increase its model-building resistance. The enrollment process involves building compact models of the arbiter PUFs using a one-time interface that allows access to the individual outputs and provides control over the input challenges. A compact model is a mathematical representation similar to what an adversary would construct when model-building the PUF.

The benefit of storing the compact models is the ability to estimate the response of the 4-XOR arbiter PUF for any arbitrary challenge. This capability is required in the proposed protocol because the challenge is composed of two parts, one part generated by the prover and one part generated by the verifier (using TRNGs). This 'on-the-fly' random challenge generation requires the verifier to generate a 'simulated' PUF response from the compact model that closely matches that produced by the actual PUF on the token. The prover's contribution to the concatenated challenge makes it impossible for an adversary to carry out a chosen-challenge attack. A third feature of the protocol relates to the manner in which authentication is performed. A seeded LFSR is used to generate a sequence of challenges that are applied to the 4-XOR PUF to produce a response bitstring. The prover then selects a fixed length substring randomly from PUF-generated response bitstring and transmits it to the verifier. The verifier authenticates the token if it can find the substring (within a predefined noise tolerance) in the corresponding estimate of the response bitstring generated from the compact model. Revealing only part of the response bitstring adds again to the difficulty of model-building.

The protocol is graphically portrayed in Fig. 15. The compact model is built during enrollment in a secure environment using a sequence of CRPs applied to the individual arbiter PUFs. The access mechanism is then disabled by blowing fuses. Authentication begins with the generation of challenges c_V and c_P by the verifier and prover, resp., which are concatenated and applied to the PUF to produce response *r*. A random index *i* is then generated that serves as the starting index into bitstring *r*. A substring of *r* is extracted as *r'*, and is returned to the verifier along with challenge c_P. The verifier uses the compact model to generate an estimate of the PUF response *r"* using the same concatenated challenge ($c_V \mid c_P$). Authentication succeeds if the verifier can locate the substring *r'* as a substring in *r"* within an error tolerance of ε.

Although the protocol is very light weight for the token, and avoids NVM, the level of model-to-hardware-correlation attained in the compact model must be very high and must be able to accommodate changes introduced by TVNoise, resulting in considerable time and effort at enrollment. PUFs that are easily modeled simplifies the development of the compact model, but also represents somewhat of a contradiction to their required resilience to model-building attacks. Also, the proposed protocol does not preserve privacy.

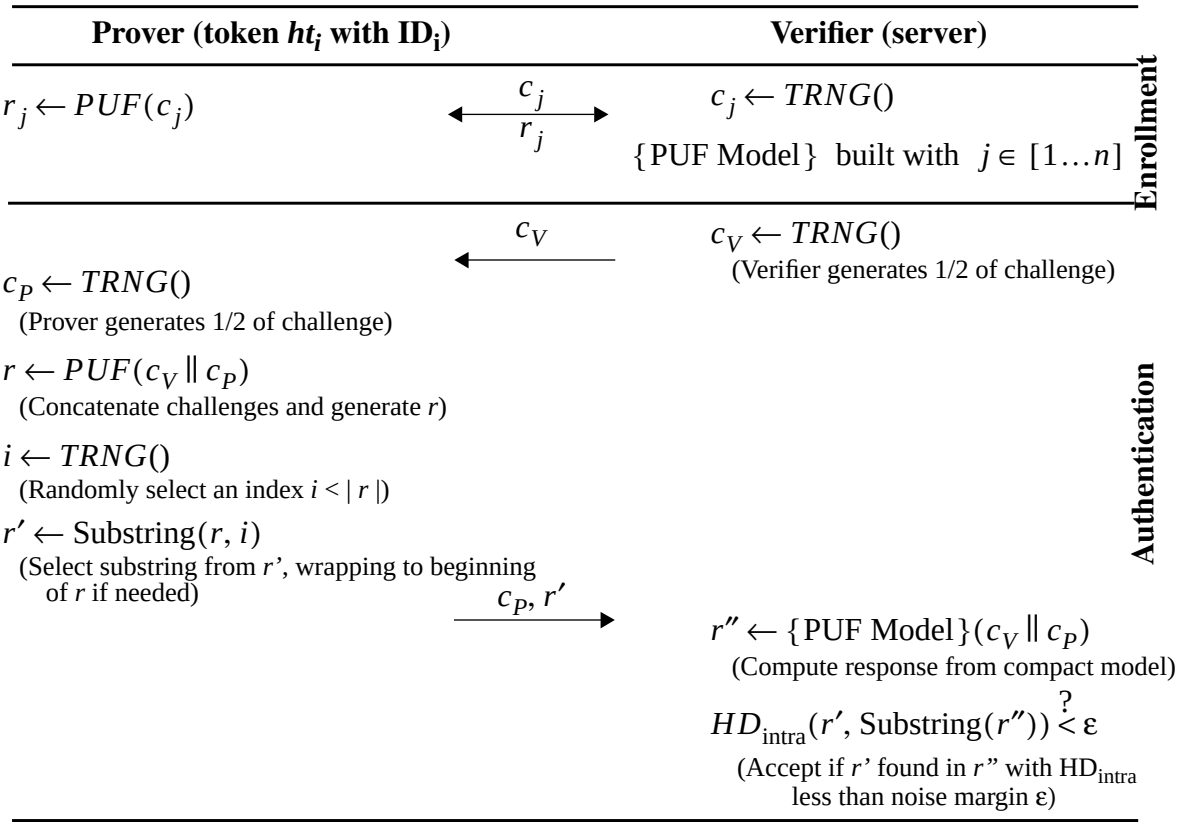


Fig. 15. Slender PUF authentication protocol proposed in [137].

6.5 Protocol 5: A Privacy-Preserving, Mutual Authentication Protocol

Aysu et al. recently proposed and implemented a PUF-based authentication protocol that provides both *privacy* and *mutual* authentication in resource-constrained environments [21]. They adapt the privacy protocol proposed by [139] to work as a reverse fuzzy extractor, as described in Section 6.3. The protocol ensures that an adversary is unable to identify or trace the tokens across multiple mutual authentications, despite the adversary having the ability to monitor and control communications and read out the contents of the token's non-volatile memory (NVM). The protocol assumes circuit-level countermeasures are implemented in the tokens to guard against other types of physical attacks, including fault injection and differential power analysis.

The protocol is designed to minimize the functional operations that are to be carried out by the token, but given the privacy goal, the protocol requires the token to implement 4 cryptographic primitives including the *Gen* operation of the fuzzy extractor algorithm, a symmetric encryption algorithm *Enc*, a random number generator *TRNG* and a pseudo-random function *PRF*. Moreover, the token makes use of an NVM to store information between authentications, in particular, a secret key sk_1 and a PUF challenge c_1. However, the protocol is designed such that leakage of this stored information cannot be used by an adversary to impersonate the token. In particular, the stored challenge is used to allow the token to reproduce a specific PUF response while the secret key is used to encrypt helper data produced by the fuzzy extractor's *Gen* operation on the token. The encryption of the helper data prevents the adversary from reverse engineering the helper data in an attempt to learn the PUF response to the NVM-stored challenge c_1.

Another key feature of the protocol, in support of the privacy objective, is the implementation of a key update mechanism. After each successful authentication, the key stored on the token and

in the server's database is updated by applying a new challenge to the PUF and obtaining its response, thereby creating a *chained* sequence of keys across successive authentications. A copy of the state information to be replaced is maintained as a countermeasure to de-synchronization, and subsequent denial-of-service, attacks.

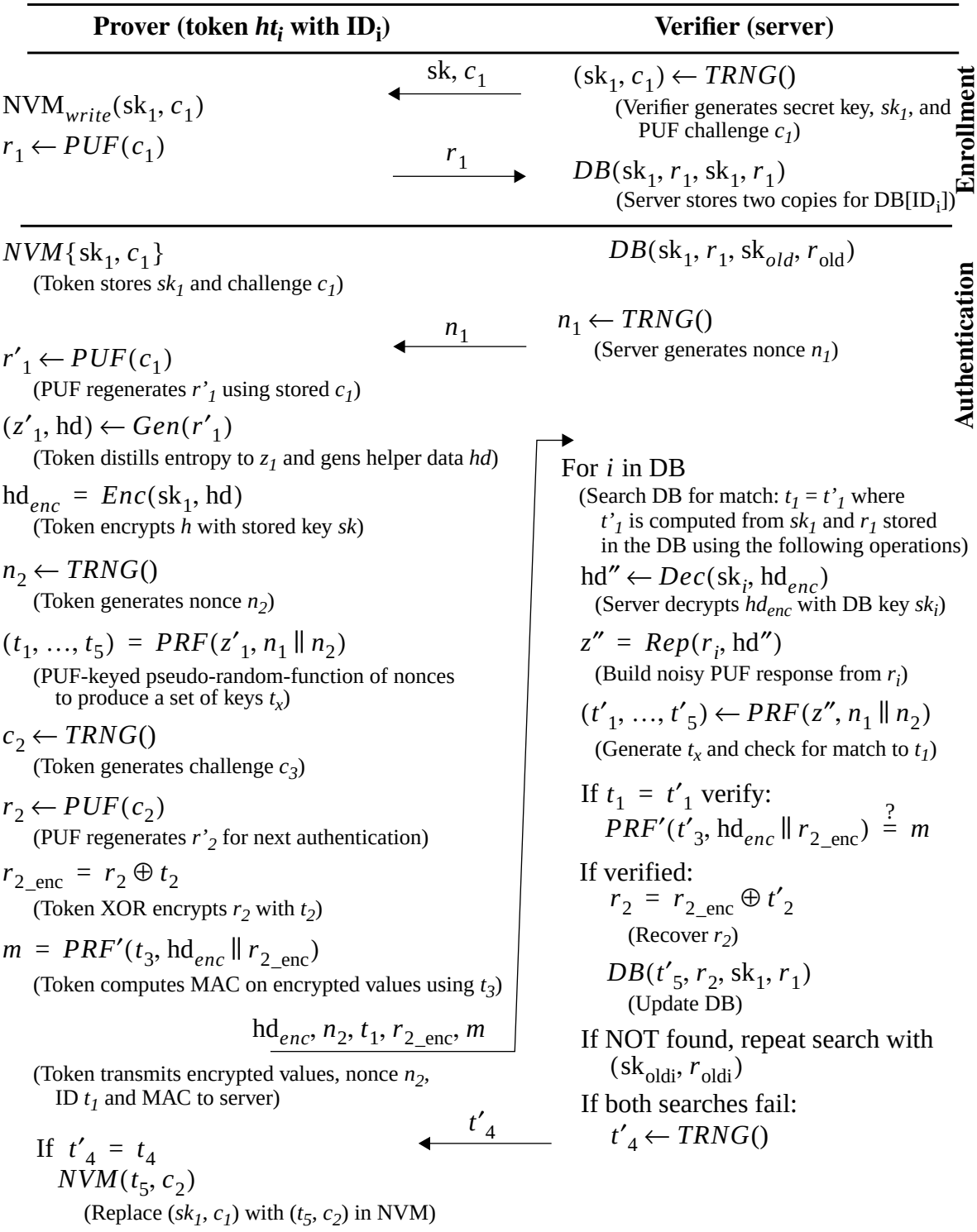


Fig. 16. Part 1: Mutual, Privacy-preserving authentication protocol proposed in [21][139].

A graphical illustration of the protocol operation is shown in Fig. 16. The Enrollment operation is carried out in a secure environment. The server generates a secret key sk_1 and a challenge c_1 that is stored in NVM on the token. The token generates a response r_1 from the PUF and provides it to the server through a one-time interface. The server stores two copies of the sk_1 and r_1 in its secure database. The combination of sk_1 and r_1 is used to derive an ID for the token, as discussed below.

The server begins the authentication process by generating a nonce n_1, which is transmitted to the token. The token's challenge c_1 is read from the NVM and used to generate a noisy PUF response r'_1. The *Gen* component of the fuzzy extractor produces z'_1 (an entropy distilled version of r'_1) and helper data *hd*. Helper data *hd* is encrypted using the key sk_1 from the NVM to produce hd_{enc}. The token then generates a nonce n_2. The PUF-generated key z'_1 and the concatenated nonces $(n_1||n_2)$ are used as input to a pseudo-random function *PRF* to produce a set of unique values t_1 through t_5 that are used as an ID, keys and challenges in the remaining steps of the protocol. A second response r_2 is obtained from the PUF using a new randomly generated challenge c_2, which will serve as the *chained* key for the next authentication (assuming this one succeeds). It is XOR-encrypted as r_{2_enc} for secure transmission to the server. *PRF'* is then used to compute a MAC *m* using t_3 as the key, over the concatenated, encrypted helper data and new key $(hd_{enc}||r_{2\_enc})$ to allow the server to check the integrity of hd_{enc} and r_{2_enc}. The encrypted values hd_{enc} and r_{2_enc} plus n_2, t_1 and *m* are transmitted to the server. The nonce n_2, as usual, introduces 'freshness' in the exchange, preventing replay attacks. The ID t_1 will be the target of a search in the server database during the server side execution of the protocol.

The server begins an exhaustive search of the database, carrying out the following operations for each entry in the DB: 1) decrypt helper data hd_{enc} using the current DB-stored sk_i to produce *hd"*, 2) construct *z"* using the fuzzy extractor's *Rep* procedure and helper data *hd"*, 3) compute t'_1 through t'_5 from $PRF(z", n_1||n_2)$ and 4) compare token generated value t_1 with t'_1. If a match is found, then the server verifies that the token's MAC *m* matches the $PRF'(t'_3, h_{enc}||r_{2\_enc})$ computed by the server. If they match, then the token's PUF-generated key r_2 is recovered using (r_{2_enc} XOR t'_2), and the database is updated by replacing $(sk_1, r_1, sk_{old}, r_{old})$ with (t'_5, r_2, sk_1, r_1). If the exhaustive search fails, then the entire process is repeated using (sk_{oldi}, r_{oldi}). If both searches fail, the server generates a random t'_4 (which guarantees failure when the token authenticates). Otherwise, the t'_4 produced from a match during the first or second search is transmitted to the token. The token compares its t_4 with the received t'_4. If they match, the token updates its NVM replacing (sk_1, c_1) with (t_5, c_2). Otherwise, the old values are retained.

Note that the old values are needed for de-synchronization attacks where the adversary prevents the last step, i.e., the proper transmission of t'_4 from the server to the token. In such cases, the server has authenticated the token and has committed the update to the DB with (t'_5, r_2, sk_1, r_1) but the token fails to authenticate the server, so the token retains its old NVM values (sk_1, c_1). On a subsequent authentication, the first search process fails to find the t'_5, r_2 components but the second search will succeed in finding sk_1, r_1. This allows the token and server to re-synchronize.

The encryption of the helper data *hd*, as mentioned, prevents the adversary from repeatedly attempting authentication to obtain multiple copies of the helper data, and then using them to

reverse engineer the PUF's secret. Note that encryption does not prevent the adversary from manipulating the helper data, and carrying out denial-of-service attacks, so the MAC operation is required to attain this security goal.

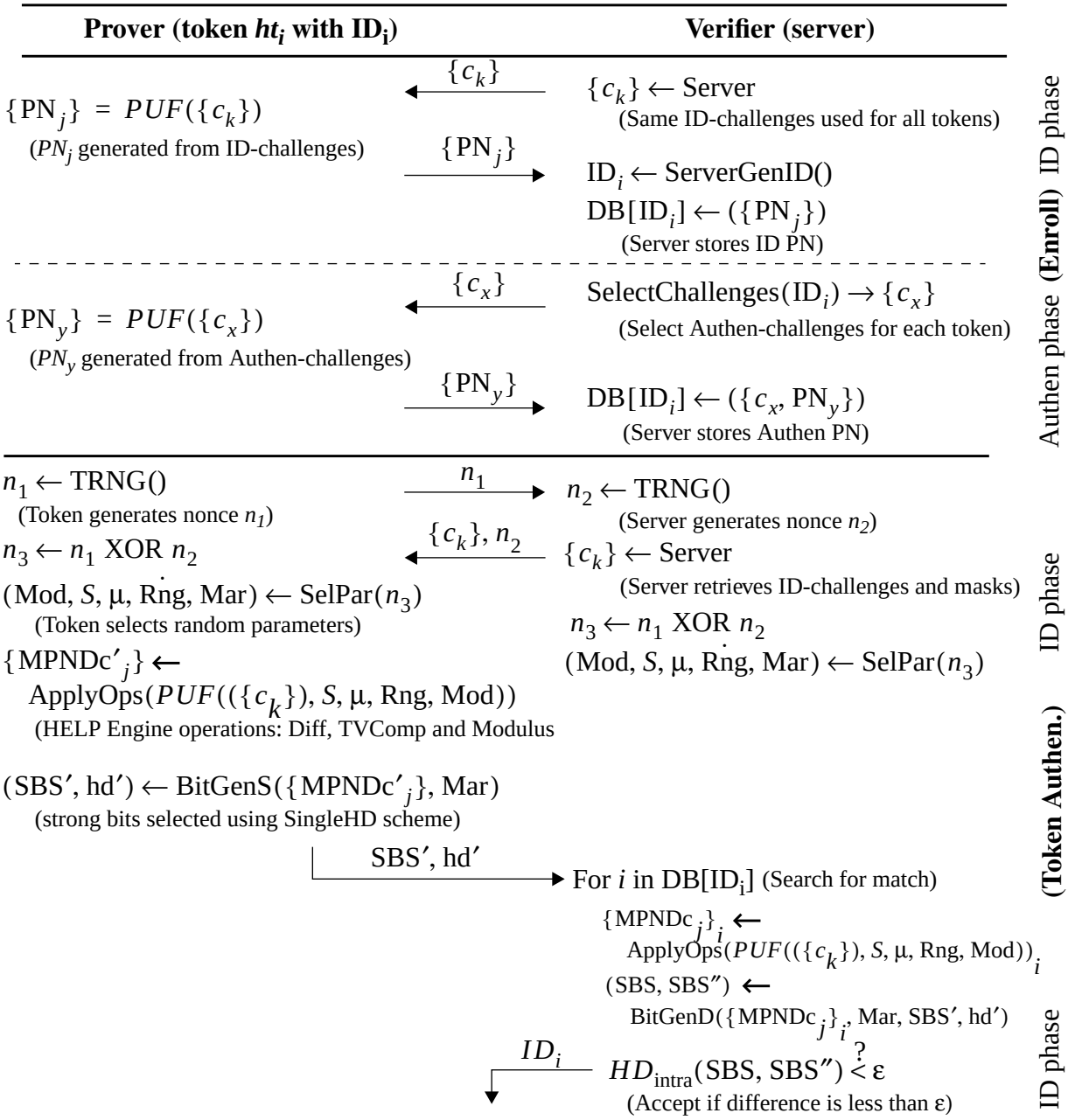
The weakest part of the algorithm is the very limited amount of PUF response information maintained by the server, i.e, effectively only one PUF response. Although the authors claim that circuit countermeasures can be used to prevent the PUF response from being extracted from the token using, e.g., differential power analysis, the entire security of the protocol is based on this premise. If, for example, the token's z'_1 is extracted, a clone that impersonates the token can be easily constructed (one that does not even need to embed a PUF), and once it authenticates successfully the first time, the authentic token is barred forever from succeeding (denial-of-service). The very limited amount of PUF response information stored on the server, although attractive from a storage overhead point-of-view, makes it vulnerable to this type of de-synchronization attack. Other issues relate to the requirement for NVM and the not-so-light-weight encryption function, which work against the low-cost, resource-constrained objective.

6.6 Protocol 6: The HELP Authentication Protocol

Similar to Protocol 5, the HELP authentication protocol is privacy-preserving and mutual, targets resource-constrained tokens and makes the same assumptions regarding adversarial threats to the token and server [117]. However, HELP does not make use of NVM, does not implement privacy using a *chained* key-update mechanism and requires no cryptographic operation to be implemented on the token. The protocol is unique among those discussed in that it stores PUF *soft information* on the server instead of bitstrings or PUF models. Soft information refers to digitized path delay values, which from Section 5.4.2, can each be represented as an 14-bit value, depending on the digital clock manager parameters. When combined with the set of user-defined parameters described in Section 5.4, including *Modulus*, *Margin*, μ and *Rng*, two *11-bit LFSR Seeds* and a *Path-Selection-Mask*, this feature, i.e., storing path delay information, provides some distinct advantages over storing response bitstrings, as highlighted below.

The enrollment operation is graphically illustrated along the top of Fig. 17. The authentication protocol uses a common set of challenges $\{c_k\}$ for all tokens as a mechanism to preserve privacy while establishing the token's identity on the server during the *ID Phase* of in-field authentication. The challenges $\{c_k\}$ are transmitted to the token in a secure environment during enrollment and applied as inputs to the PUF. A set of PN are produced and returned to the server as $\{PN_j\}$. The server generates an internal identifier ID_i for each token using *ServerGenID* and stores the set $\{PN_j\}$ under ID_i in the secure database.

A similar process is carried out during the *Authen Phase* of enrollment except that the challenges are selected from a large set using *SelectChallenges*(ID_i) for each token among those that have been generated using random vectors or automatic test pattern generation (ATPG). The server ensures that the selected set overlaps with those chosen for other tokens, but with no more than 50% overlapping with any one token. This policy prevents the challenges used in the *Authen Phase* during in-field authentication from being used to track the token (explained further below). The set of PN $\{PN_y\}$ generated in the *Authen Phase* are also stored, along with the challenge vectors, in the secure database under ID_i. The number of structural paths for the data path component of SHA-3 is larger than 860,000, with more than 80% testable, so the set of challenge vectors available is large. Note that the task of generating 2-vector tests for all paths is likely to be computationally infeasible for even moderately sized functional units. However, it is feasible and practical to use random vectors and ATPG to target random subsets of paths for the enrollment



(If succeed, then proceed with *Authen Phase*)

Fig. 17. The HELP Authentication Protocol.

requirements.

The cardinality of $\{PN_y\}$ is approx. twice that of $\{PN_j\}$ at 8192 but both are relatively small because the parameters, particularly the *Path-Selection-Mask*, allow an exponential number of different combinations to be constructed over successive authentications. The example from Section 5.4.7 uses the *Path-Selection-Mask* to select 50 PN per challenge. In this case, the number of challenges that need to be applied in the *ID* and *Authen Phases* during enrollment is approx. 80 and 160, resp.

The protocol for token authentication is shown in the bottom portion of Fig. 17. The token initiates the process by generating and sending a nonce n_1 to the server. The server generates a nonce

n_2, retrieves the fixed set of challenges $\{c_k\}$ and transfers n_2 and the challenges to the token. Both the token and server compute (n_1 XOR n_2) to produce n_3. This strategy prevents a *chosen-message* attack by adversaries, where the 'message' refers the HELP parameters. The XOR'ed nonce n_3 is used as input to a *SelPar* function to derive the *Mod, S,* μ, *Rng, Mar* parameters. The *SelPar* function selects bit fields in n_3 for use in a lookup-table operation to *pseudo-randomly* constrain the *Mod* and *Mar* parameters to a specific set of values (as given in Fig. 11). Other bit fields are used to define μ and *Rng*, constrained, in this case, to a range of fixed-point values. The same *SelPar* operation is carried out on the server. This component of the protocol is similar to the strategy proposed for the Slender PUF Protocol described in Section 6.4 [137] but is used there for challenge selection.

The set $\{c_k\}$ of challenges are applied to the PUF to generate the set $\{PN'_j\}$. The difference, TVComp and modulus operations shown on the right side of Fig. 4 are applied to $\{PN'_j\}$ to generate the set $\{MPNDc'_j\}$. Bitstring generation using the single helper data scheme, *BitGenS,* is then performed as described in Section 5.4.4 using the *Mar* parameter. *BitGenS* produces a strong bitstring *SBS'* and helper data string *hd*', which are both transmitted to the server.

A search process is carried out on the server, where the $\{PN_j\}_i$ data for each token *i* in the database is processed in a similar fashion. However, bitstring generation is carried out using the dual helper data scheme (*BitGenD*). *BitGenD* returns an *SBS* computed using the server data and a modified bitstring *SBS"*, which is a reduced-in-size version of the token's *SBS'* (see section 5.4.4 for details). The search process terminates when the number of bits that differ in *SBS* and *SBS"* is less than a tolerance ε (which may be zero) or the database is exhausted. In the former case, the token identifier ID_i is passed to the *Authen Phase*. Otherwise, authentication terminates with failure at the end of the *ID Phase*.

Note that token privacy is preserved in the *ID Phase* because, with high probability, the transmitted information *SBS'* and *hd'* will be different from one run of the protocol to the next, given the diversity of the parameter space provided by *Mod, S,* μ, *Rng*, *Mar* and *Path-Select-Mask*. Also note that this is a compute-intensive operation for large databases because the difference, TVComp, modulus and BitGenD operations must be applied to each server data set. However, the search operation can be carried out in parallel on multiple CPUs given the independence of the operations. Trial run experiments without any type of explicit parallelism yields runtimes of 200 us per database entry using a database of 10,000 elements when evaluated on an Intel i7-4702HQ CPU @ 2.2 GHz running Linux.

The *Authen Phase* is not shown but is identical to the *ID Phase* with the following exceptions. The subset of 80 token-specific challenges $\{c_1\}$ are randomly selected from the larger set of 160 in $\{c_x\}$ that were applied during enrollment. As indicated earlier, the 160 challenges selected for a token overlap with those selected for other tokens, making it impossible for adversaries to track specific tokens across multiple authentications. A second difference is that the *Authen* phase represents the *mutual authentication* step, in which the server is authenticated to the token. Therefore, the server generates the *SBS'* and *hd'* using the Single Helper Data scheme, which are then transmitted to the token, and the token implements the Dual Helper Data scheme and fuzzy match operations (opposite to that shown in Fig. 17)[1]. This is possible in a resource-constrained environ-

1. If needed, an optional third phase can be implemented to carry out a second token authentication but using the $\{c_x\}$ challenges instead of the *ID Phase* $\{c_k\}$ challenges.

ment because of the symmetry in energy requirements of the proposed error avoidance schemes, i.e., the work performed by the Single Helper Data and Dual Helper Data schemes are nearly the same.

7. PUF-based Authentication for SoC

System-on-chip (SoC) devices continue to proliferate as core components in IoT applications. Although not considered a resource-constrained device, the heterogeneous multi-core, multi-technology characteristics of SoCs, many of which integrate third party IP, make them easy targets for sabotage, reverse engineering, substitution and cloning. The threat is exacerbated when the SoC integrates cryptographic IP blocks. PUF-based authentication mechanisms can be used to detect manipulation and substitution in the supply-chain and later as installed components in fielded systems.

Applications of PUF-based authentication in SoC is expanding. Recent work focuses on preventing scan chain attacks, carrying out entity authentication and providing authentication of bitstreams for FPGAs. For example, the authors of [140] propose a secure test wrapper which allows testing of multiple IP blocks using PUF-based authentication as a mechanism to improve the security of SoCs that embed IP cores. A low-cost PUF-based authentication architecture designed to secure code execution in IoT SoCs is proposed in [141] The proposed architecture extracts a PUF-based key from the processor's cache to address threats against code and data authenticity and integrity. A scan chain PUF is proposed in [142] for authenticating SoCs as part of an Infrastructure IP designed to provide multiple security functions. An overview of traditional and modern-day bitstream authentication (and encryption) in FPGAs is provided in [143].

8. Conclusion

Authentication protocols, although proposed initially for digital systems over 40 years ago, continue to evolve as new cryptographic functions, such as the Physical Unclonable Function, become available as primitives for enabling physical layer security properties including secure key generation and storage. Adversarial attack surfaces are widening with the proliferation of low-cost and embedded devices for home automation, RFID, smart cards/cars/grids, embedded medical devices, and other types of Internet-of-Things applications. Adversarial attack mechanisms, including physical-layer information extraction techniques, model building and sophisticated network communication tracking algorithms, exacerbate the task of implementing secure unilateral, mutual and privacy preserving authentication protocols. The introduction of PUFs as primitives can be leveraged to serve as significant countermeasures to adversarial attack mechanisms, particularly for authentication in resource-constrained environments.

This chapter covered both traditional and emerging PUF-based authentication protocols. The primary function of a PUF is to securely generate and store secrets, that can be converted, at any instance in time, into bitstrings for direct use in authentication functions and/or as keys for hashing and encryption functions within authentication protocols. The source of a PUF's entropy is based primarily on within-die variations that occur among circuit components of an integrated circuit. Within-die variations are uncontrollable and unique to each copy of the IC, which allows the PUF to produce unclonable and instance-specific bitstrings.

The integration of PUFs into commercial products is not yet wide-spread. However, published work on PUF constructions and their use in security and trust protocols is growing day-by-day. A wide variety of PUF primitives exist, each with distinctive characteristics related to the number of generated bits (weak vs. strong), robustness to on-chip noise sources, and the statistical quality of the generated bitstrings, e.g., randomness and uniqueness. Existing work shows how PUFs can

address shortcoming and provide new capabilities to traditional software-based approaches to authentication but, as discussed in [121][138], care must be taken to properly characterize the security properties of specific PUF constructions in order to ensure functional and/or practical implementations.

9. References

[1] K. M. Goertzel, “Integrated Circuit Security Threats and Hardware Assurance Countermeasures”, Real-Time Information Assurance”, *CrossTalk*, Nov./Dec. 2013.

[2] S. Pope, B. S. Cohen, V. Sharma, R. R. Wagner, L. W. Linholm and S. Gillespie, “Verifying Trust For Defense Use Commercial Semiconductors”.

[3] “Grand Challenges for Engineering”, http://www.engineeringchallenges.org/cms/8996/9042.aspx

[4] “Defense Science Board Task Force On High Performance Microchip Supply“, Office of the Under Secretary of Defense, Feb. 2005, http://www.acq.osd.mil/dsb/reports/2005-02-HPMS_Report_Final.pdf

[5] Dean Collins, “TRUST, A Proposed Plan for Trusted Integrated Circuits”, http://www.stormingmedia.us/95/9546/A954654.html

[6] Senator Joe Lieberman, “National Security Aspects of the Global Migration of the U.S. Semiconductor Industry,” June 2003, http://lieberman.senate.gov/documents/whitepapers/semiconductor.pdf

[7] “TRUST in Integrated Circuits (TIC)”, http://www.darpa.mil/mto/solicitations/baa07-24/index.html

[8] “National Cyber Leap Year Summit 2009: Co-Chairs’ Report”, September 16, 2009, http://www.qinetiq-na.com/Collateral/Documents/English-US/InTheNews_docs/National_Cyber_Leap_Year_Summit_2009_Co-Chairs_Report.pdf

[9] “Integrity and Reliability of Integrated Circuits”, DARPA-BAA-10-33, 2010.

[10] “Trusted Integrated Chips (TIC)”, IARPA-BAA-11-09, 2011.

[11] Bureau of Industry and Security, U.S. Department of Commence. Defense Industrial Base Assessment: Counterfeit Electronics. http://www.bis.doc.gov/index.php/forms-documents/doc_download/37-defense-industrial-base-assessment-of-counterfeit-electronics-2010

[12] B. Grow, C.-C. Tschang, C. Edwards and B. Burnsed, “Dangerous Fakes”, Businessweek, 2008, http://www.businessweek.com/stories/2008-10-01/dangerous-fakes

[13] L. W. Kessler and T. Sharpe, “Faked Parts Detection”, 2010, http://www.circuitsassembly.com/cms/component/content/article/159/9937-smt

[14] J. Stradley and D. Karraker, “The Electronic Part Supply Chain and Risks of Counterfeit Parts in Defense applications”, *IEEE Trans. on Components and Packaging Technology*, Vol. 29, No. 3, 2006, pp. 703-705.

[15] H. Ke, J. M. Carulli and Y. Makris, “Counterfeit Electronics: A Rising Threat in the Semiconductor Manufacturing Industry”, *International Test Conference (ITC)*, 2013, pp. 1-4.

[16] B. Gassend, D. E. Clarke, M. van Dijk, S. Devadas, “Controlled Physical Random Functions”, *Conference on Computer Security Applications*, 2002, pp. 149-160.

[17] https://en.wikipedia.org/wiki/Information_security

[18] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, “Handbook of Applied Cryptography”, CRC Press, ISBN: 0-8493-8523-7, Oct. 1996, http://cacr.uwaterloo.ca/hac/

[19] R. Maes, “Physical Unclonable Functions, Constructions, Properties and Applications”, *Springer*, 2013, ISBN 978-3-642-41394-0

[20] E. Barker and J. Kelsey, “Recommendation of Random Number Generation using Deterministic Random Bit Generators”, NIST SP800-90A, https://en.wikipedia.org/wiki/NIST_SP_800-90A.

[21] A. Aysu, E. Gulcan, D. Moriyama, P. Schaumont, and M. Yung, "End-to-end Design of a PUF-based Privacy Preserving Authentication Protocol", *CHES*, 2015.

[22] https://en.wikipedia.org/wiki/Cryptographic_hash_function

[23] https://en.wikipedia.org/wiki/SHA-3

[24] https://en.wikipedia.org/wiki/Secure_Hash_Algorithm
[25] http://www.nist.gov/manuscript-publication-search.cfm?pub_id=919061
[26] Y. Dodis, L. Reyzin, A. Smith, “Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data”, *Advances in cryptology* (EUROCRYPT), 2004, pp. 523-540.
[27] Y. Dodis, R. Ostrovsky, L. Reyzin, A. Smith, “Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data”, *SIAM Journal on Computing*, 38(1), 2008, 97-139.
[28] A. Van Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, “Reverse Fuzzy Extractors: Enabling Lightweight Mutual Authentication for PUF-enabled RFIDs”, *Vol. 7397 of Lecture Notes in Computer Science*, 2012, pp. 374-389.
[29] https://en.wikipedia.org/wiki/Binomial_distribution
[30] NIST: Computer Security Division, Statistical Tests, http://csrc.nist.gov/groups/ST/toolkit/rng/stats_tests.html
[31] R. Needham and M. Schroeder, "Using Encryption for Authentication in Large Networks of Computers", *Communications of the ACM*, Vol. 21, No. 12, 1978, pp. 993-999.
[32] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, C. Wachter, “Ron Was Wrong, Whit Is Right”, 2012, *Cryptology ePrint Archive*, Report 2012/064.
[33] R. Torrance and D. James, “The State-of-the-Art in IC Reverse Engineering”, In Lecture Notes in Computer Science (LNCS), *Workshop on Cryptographic Hardware and Embedded Systems*, Vol. 5747, 2009, pp. 363-381.
[34] P. C. Kocher, J. Jaffe, B. Jun, “Differential Power Analysis”, In Lecture Notes in Computer Science (LNCS), *Advances in Cryptology*, Vol. 1666, 1999, pp. 388-397.
[35] K. Lofstrom, W. R. Daasch, D. Taylor, “IC Identification Circuits using Device Mismatch”, *International Solid State Circuits Conference*, 2000, pp. 372-373.
[36] D. Puntin, S. Stanzione, G. Iannaccone, “CMOS Unclonable System for Secure Authentication based on Device Variability”, *Conference on Solid-State Circuits Conference*, 2008, pp. 130-133.
[37] S. Stanzione and G. Iannaccone, “Silicon Physical Unclonable Function Resistant to a 1025-trial Brute Force Attack in 90 nm CMOS”, *Symposium VLSI Circuits*, 2009, pp. 116-117.
[38] R. Pappu, “Physical One-Way Functions”, *PhD Thesis*, MIT, ch. 9, 2001.
[39] R. S. Pappu, B. Recht, J. Taylor, N. Gershenfeld, “Physical One-Way Functions,” *Science*, 297(6), 2002, pp. 2026-2030.
[40] B. Gassend and D. E. Clarke and M. van Dijk, S. Devadas, “Silicon Physical Random Functions”, *Conference on Computer and Communications Security*, 2002, 148-160.
[41] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. van Dijk, S. Devadas, “A Technique to Build a Secret Key in Integrated Circuits for Identification and Authentication Applications”, *Symposium of VLSI Circuits*, 2004, pp. 176-179.
[42] D. Lim, “Extracting Secret Keys from Integrated Circuits”, M.S. Thesis, MIT, 2004.
[43] D. Lim, J. W. Lee, B. Gassend, G.E. Suh, M. van Dijk, S. Devadas, “Extracting Secret Keys from Integrated Circuits“, *Trans. on Very Large Scale Integration Systems*, 13(10), Oct. 2005, pp. 1200-1205.
[44] G. E. Suh and S. Devadas, “Physical Unclonable Functions for Device Authentication and Secret Key Generation”, *Design Automation Conference*, 2007, pp. 9-14.
[45] M. Majzoobi, F. Koushanfar, M. Potkonjak, “Lightweight Secure PUFs”, *Conference on Computer-Aided Design*, 2008.
[46] M. Majzoobi, F. Koushanfar, M. Potkonjak, "Testing Techniques for Hardware Security", *International Test Conference*, 2008, pp. 185-189.
[47] E. Ozturk, G. Hammouri, B. Sunar, “Physical Unclonable Function with Tristate Buffers”, *Symposium on Circuits and Systems*, 2008, pp. 3194-3197.
[48] E. Ozturk, G. Hammouri, B. Sunar, “Towards Robust Low Cost Authentication for Pervasive Devices”, *Conference on Pervasive Computing and Communications*, 2008 pp. 170-178.
[49] B. Gassend, M. Van Dijk, D. Clarke, E. Torlak, S. Devadas, P. Tuyls, “Controlled Physical Random Functions and Applications”, *ACM Transactions on Information and System Securi-*

ty, Volume 10, Number 4, 2008.
[50] S. Devadas, E. Suh, S. Paral, R. Sowell, T. Ziola, V. Khandelwal, "Design and Implementation of PUF-Based 'Unclonable' RFID ICs for Anti-Counterfeiting and Security Applications", *Conference on RFID*, 2008, pp. 58-64.
[51] G. Qu and C. Yin, "Temperature-Aware Cooperative Ring Oscillator PUF", *Workshop on Hardware-Oriented Security and Trust*, 2009, pp. 36-42.
[52] A. Maiti and P.Schaumont, "Improving the Quality of a Physical Unclonable Function using Configurable Ring Oscillators", *Conference on Field Programmable Logic and Applications*, 2009. pp. 703-707.
[53] A. Maiti, J. Casarona, L. McHale, R. Schaumont, "A Large Scale Characterization of RO-PUF", *Symposium on Hardware-Oriented Security and Trust*, 2010, pp. 94-99.
[54] Y. Hori, T. Yoshida, T. Katashita, A. Satoh, "Quantitative and Statistical Performance Evaluation of Arbiter Physical Unclonable Functions on FPGAs", *Conference on Reconfigurable Computing and FPGAs*, 2010, pp. 298-303.
[55] C.-E.D. Yin and Gang Qu, "LISA: Maximizing RO PUF's Secret Extraction", *Symposium on Hardware-Oriented Security and Trust*, 2010, pp. 100-105.
[56] C. Costea, F. Bernard, V. Fischer, R. Fouquet, "Analysis and Enhancement of Ring Oscillators Based Physical Unclonable Functions in FPGAs", *Conference on Reconfigurable Computing and FPGAs*, 2010, pp. 262-267.
[57] M. Majzoobi, F. Koushanfar, S. Devadas, "FPGA PUF using Programmable Delay Lines", *Workshop on Information Forensics and Security*, 2010, pp. 1-6.
[58] X. Xin, J. Kaps, K. Gaj, "A Configurable Ring-Oscillator-Based PUF for Xilinx FPGAs", *Conference on Digital System Design*, 2011, pp. 651-657.
[59] C. Qingqing, G. Csaba, P. Lugli, U. Schlichtmann, U. Ruhrmair, "The Bistable Ring PUF: A New Architecture for Strong Physical Unclonable Functions", *Symposium on Hardware-Oriented Security and Trust*, 2011, pp. 134-141.
[60] C. Qingqing, G. Csaba, P. Lugli, U. Schlichtmann, U. Ruhrmair, "Characterization of the Bistable Ring PUF", *Design, Automation & Test in Europe Conference*, 2012, pp. 459-1462.
[61] S. S. Mansouri and E. Dubrova, "Ring Oscillator Physical Unclonable Function with Multi Level Supply Voltages", *International Conference on Computer Design*, 2012, pp. 520-521.
[62] T. Addabbo, A. Fort, M. Mugnaini, S. Rocchi, V. Vignoli, "Statistical Characterization of a FPGA PUF Module Based on Ring Oscillators", *Instrumentation and Measurement Technology Conference*, 2012, pp. 1770-1773.
[63] A. Maiti, K. Inyoung, P. Schaumont, "A Robust Physical Unclonable Function With Enhanced Challenge-Response Set", *Trans. on Information Forensics and Security*, Volume: 7, Issue: 1, Part: 2, 2012, pp. 333-345.
[64] Y. Meng-Day, R. Sowell, A. Singh, D. M'Raihi, S. Devadas, "Performance Metrics and Empirical Results of a PUF Cryptographic Key Generation ASIC", *Symposium on Hardware-Oriented Security and Trust*, 2012, pp. 108-115.
[65] S. Maeda, H. Kuriyama, T. Ipposhi, S. Maegawa, Y. Inoue, M. Inuishi, N. Kotani, T. Nishimura, "An Artificial Fingerprint Device (AFD): a Study of Identification Number Applications Utilizing Characteristics Variation of Polycrystalline Silicon TFTs", *Trans. on Electron Devices*, number 50, issue 6, June, 2003, pp.1451- 1458.
[66] E. Simpson and P. Schaumont, "Offline Hardware/Software Authentication for Reconfigurable Platforms", *Cryptographic Hardware and Embedded Systems*, Volume 4249, Oct., 2006, pp. 10-13.
[67] B. Habib, K. Gaj and J.-P. Kaps, "FPGA PUF Based on Programmable LUT Delays", *Euromicro Conference on Digital System Design (DSD)*, 2013, pp. 697-704.
[68] J. Guajardo, S. S. Kumar, G.-J. Schrijen, P. Tuyls, "Physical Unclonable Functions and Public Key Crypto for FPGA IP Protection", *Conference on Field Programmable Logic and Applications*, 2007, 189-195.
[69] Y. Su and J. Holleman and B. Otis, "A 1.6pJ/bit 96% Stable Chip ID Generating Circuit using Process Variations", *International Solid State Circuits Conference*, 2007, pp. 406-407.
[70] J. Guajardo, S. S. Kumar and G. Schrijen and P. Tuyls, "Brand and IP Protection with Phys-

ical Unclonable Functions", *Symposium on Circuits and Systems*, 2008, pp. 3186-3189.
[71] S. S. Kumar, J. Guajardo, R. Maes, Geert-Jan Schrijen, P. Tuyls, "Extended Abstract: The Butterfly PUF Protecting IP on Every FPGA", *Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 70-73.
[72] M. Kassem, M. Mansour, A. Chehab, A. Kayssi, "A Sub-Threshold SRAM Based PUF", *Conference on Energy Aware Computing*, 2010, pp. 1-4.
[73] C. Bohm, M. Hofer, W. Pribyl, "A Microcontroller SRAM-PUF", *Conference on Network and System Security*, 2011, pp. 25-30.
[74] M. Bhargava, C. Cakir and K. Mai, "Reliability Enhancement of Bi-Stable PUFs in 65nm Bulk CMOS", *Workshop on Hardware-Oriented Security and Trust*, 2012, 79-83.
[75] Y. Alkabani and F. Koushanfar and N. Kiyavash and M. Potkonjak, "Trusted Integrated Circuits: A Nondestructive Hidden Characteristics Extraction Approach", *Information Hiding*, 2008.
[76] D. Ganta, V. Vivekraja, K. Priya, L. Nazhandali, "A Highly Stable Leakage-Based Silicon Physical Unclonable Functions", *Conference on VLSI Design*, 2011, pp. 135-140.
[77] R. Helinski, D. Acharyya, J. Plusquellic, "Physical Unclonable Function Defined Using Power Distribution System Equivalent Resistance Variations", *Design Automation Conference*, 2009, pp. 676-681.
[78] R. Helinski, D. Acharyya, J. Plusquellic, "Quality Metric Evaluation of a Physical Unclonable Function Derived from an IC's Power Distribution System", *Design Automation Conference*, 2010, pp. 240-243.
[79] J. Ju, R. Chakraborty, R. Rad, J. Plusquellic, "Bit String Analysis of Physical Unclonable Functions based on Resistance Variations in Metals and Transistors", *Sym. on Hardware-Oriented Security and Trust*, 2012, pp. 13-20.
[80] J. Ju, R. Chakraborty, C. Lamech and J. Plusquellic, "Stability Analysis of a Physical Unclonable Function based on Metal Resistance Variations", *Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 143-150.
[81] D. Ismari and J. Plusquellic, "IP-Level Implementation of a Resistance-Based Physical Unclonable Function", accepted to *Symposium on Hardware-Oriented Security and Trust (HOST)*, 2014.
[82] R. Chakraborty, C. Lamech, D. Acharyya and J. Plusquellic, "A Transmission Gate Physical Unclonable Function and On-Chip Voltage-to-Digital Conversion Technique", *Design Automation Conference (DAC)*, 2013, pp. 1-10.
[83] J. Li and J. Lach, "At-Speed Delay Characterization for IC Authentication and Trojan Horse Detection, *International Workshop on Hardware-Oriented Security and Trust (HOST)*, 2008, pp. 8-14.
[84] J. Aarestad, P. Ortiz, D. Acharyya and J. Plusquellic, "HELP: A Hardware-Embedded Delay-Based PUF", IEEE Design and Test of Computers, Vol. 30, Issue: 2, Mar., 2013. pp. 17-25.
[85] J. Aarestad, D. Acharyya, J. Plusquellic, "An Error-Tolerant Bit Generation Technique For Use With A Hardware-Embedded Path Delay PUF", *Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 151-158.
[86] F. Saqib, M. Areno, J. Aarestad and J. Plusquellic, "An ASIC Implementation of a Hardware-Embedded Physical Unclonable Function", *IET Computers & Digital Techniques*, Vol. 8, Issue 6, Nov. 2014, pp. 288-299.
[87] W. Che, F. Saqib, and J. Plusquellic, "PUF-Based Authentication, Invited Paper, *International Conference on Computer Aided Design*, Nov. 2015.
[88] K. Kursawe, A. -R. Sadeghi, D. Schellekens, B. Skoric, P. Tuyls, "Reconfigurable Physical Unclonable Functions - Enabling Technology for Tamper-Resistant Storage", *Workshop on Hardware-Oriented Security and Trust*, 2009, pp.22-29.
[89] K. Rosenfeld, E. Gavas, R. Karri, "Sensor Physical Unclonable Functions", *Symposium on Hardware-Oriented Security and Trust*, 2010, pp. 112-117.
[90] W. Xiaoxiao and M. Tehranipoor, "Novel Physical Unclonable Function with Process and Environmental Variations", *Conference on Design, Automation & Test in Europe*, 2010, pp. 1065-1070.

[91] L. Lin, D. Holcomb, D.K. Krishnappa, P. Shabadi, W. Burleson, "Low-Power Sub-Threshold Design of Secure Physical Unclonable Functions", *Symposium on Low-Power Electronics and Design*, 2010, pp. 43-48.
[92] U. Ruhrmair, C. Jaeger, M. Bator, M. Stutzmann, P. Lugli, G. Csaba, "Applications of High-Capacity Crossbar Memories in Cryptography", *Trans. on Nanotechnology*, Volume: 10, Issue: 3, 2011, pp. 489-498.
[93] P. Simons, E. van der Sluis, V. van der Leest, "Buskeeper PUFs, a Promising Alternative to D Flip-Flop PUFs", *Symposium on Hardware-Oriented Security and Trust (HOST)*, 2012, pp. 7-12.
[94] A. Maiti and P. Schaumont, "A Novel Microprocessor-Intrinsic Physical Unclonable Function", *Field Programmable Logic and Applications*, 2012, pp. 380-387.
[95] A. Sreedhar and S. Kundu, "Physically Unclonable Functions for Embeded Security Based on Lithographic Variation", *Conference on Design, Automation & Test in Europe*, 2012, pp. 96-105.
[96] R. Kumar, S. N. Dhanuskodi and S. Kundu, "On Manufacturing Aware Physical Design to Improve the Uniqueness of Silicon-Based Physically Unclonable Functions", *International Conference on Embedded Systems*, 2014, pp. 381-386.
[97] D. Forte and A. Srivastava, "On Improving the Uniqueness of Silicon-Based Physically Unclonable Functions via Optical Proximity Correction", *Design Automation Conference*, 2012, pp. 7-12.
[98] S. Meguerdichian and M. Potkonjak, "Device Aging-Based Physically Unclonable Functions", *Conference on Design Automation Conference*, 2011, pp. 288-289.
[99] M. Kalyanaraman and M. Orshansky, "Novel Strong PUF based on Nonlinearity of MOSFET Subthreshold Operation", *Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 13-18.
[100] G. S. Rose, N. McDonald, Y. Lok-Kwong, B. Wysocki and K. Xu, "Foundations of Memristor Based PUF Architectures", *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2013, pp. 52-57.
[101] W. Che, S. Bhunia and J. Plusquellic, "A Non-Volatile Memory-based Physically Unclonable Function without Helper Data", *International Conference on Computer-Aided Design (ICCAD)*, 2014.
[102] Z. Yu, A. R. Krishna and S. Bhunia, "ScanPUF: Robust Ultralow-Overhead PUF using Scan Chain", *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2013, pp. 626-631.
[103] L. Zhang, Z. H. Kong and C-H. Chang, "PCKGen: A Phase Change Memory Based Cryptographic Key Generator", *International Symposium on Circuits and Systems (ISCAS)*, 2013, pp. 1444-1447.
[104] S. T. C. Konigsmark, L. K. Hwang, C. Deming, M. D. F. Wong, "CNPUF: A Carbon Nanotube-based Physically Unclonable Function for Secure Low-Energy Hardware Design", *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014, pp. 73-78.
[105] F. Zhang, A. Henessy, and S. Bhunia, "Robust Counterfeit PCB Detection Exploiting Intrinsic Trace Impedance Variations", *VLSI Test Symposium*, April 2015.
[106] M. Areno and J. Plusquellic, "Securing Trusted Execution Environments with PUF Generated Secret Keys", *TrustCom*, 2012.
[107] M. Areno and J. Plusquellic, "Secure Mobile Association and Data Protection with Enhanced Cryptographic Engines", *PRISMS*, 2013.
[108] J. Guajardo, S. S. Kumar, G. T. Schrijen, and P. Tuyls, "FPGA Intrinsic PUFs and their Use for IP Protection", *Cryptographic Hardware and Embedded Systems*, Volume 4727, 2007, pp. 63-80.
[109] U. Rührmair, H. Busch, S. Katzenbeisser, "Strong PUFs: Models, Constructions, and Security Proofs", *Towards Hardware-Intrinsic Security*, Editors Ahmad-Reza Sadeghi and David Naccache, Springer, 2010, pp. 79-95.
[110] B. Gassend, D. Lim, D. Clarke, M. van Dijk, S. Devadas, "Identification and Authentication of Integrated Circuits", *Concurr. Comput.* 16(11), 2004, pp. 1077-1098.

[111] M. Majzoobi, F. Koushanfar, and M. Potkonjak, “Testing Techniques for Hardware Security”, *International Test Conference*, 2008, pp. 1-10.
[112] Z. Paral and S. Devadas, “Reliable and Efficient PUF-based Key Generation using Pattern Matching”, *Symposium on Hardware-Oriented Security and Trust*, 2011, pp. 128-133.
[113] C. Lamech, J. Aarestad, J. Plusquellic, R. Rad, K. Agarwal, "REBEL and TDC: Two Embedded Test Structures for On-Chip Measurements of Within-Die Path Delay Variations", *International Conference on Computer-Aided Design*, pp. 170-177, 2011.
[114] https://en.wikipedia.org/wiki/SHA-3
[115] K. Tiri and I. Verbauwhede, “A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation”, *DATE*, 2004, pp. 246-251.
[116] K. Tiri and I. Verbauwhede, "A Digital Design Flow for Secure Integrated Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1197-1208, July 2006.
[117] W. Che, F. Saqib and J. Plusquellic, “A Privacy-Preserving, Mutual PUF-Based Authentication Protocol”, submitted to special issue “Physical Security in Cryptography Environment”, Cryptography Journal (http://www.mdpi.com/journal/cryptography), Aug. 2016.
[118] S. M. Nowick and C. W. O’Donnell, “On the Existence of Hazard-Free Multi-Level Logic”, ASYNC, 2003.
[119] http://zedboard.org/product/zedboard
[120] D. C. Ranasinghe, C. W. Engels, P. H. Cole, “Security and Privacy: Modest Proposals for Low-Cost RFID Systems”, *Auto-ID Labs Research Workshop*, 2004.
[121] J. Delvaux, D. Gu, D. Schellekens and I. Verbauwhede, “Secure Lightweight Entity Authentication with Strong PUFs: Mission Impossible?”, *CHES*, 2014, pp. 451-475.
[122] U. Rührmair, F. Sehnke, J. Solter, G. Dror, S. Devadas, J. Schmidhuber, “Modeling attacks on Physical Unclonable Functions”, *Conference on Computer and Communications Security*, 2010, pp. 237-249.
[123] A. Van Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A.-R. Sadeghi, I. Verbauwhede, C. Wachsmann, “Reverse Fuzzy Extractors: Enabling Lightweight Mutual Authentication for PUF-Enabled RFIDs”, *International Conference on Financial Cryptography and Data Security*, 2012.
[124] L. Bolotny and G. Robins, “Physically Unclonable Function-based Security and Privacy in RFID Systems”, *PerCom*, 2007, pp. 211-220.
[125] E. Ozturk, G. Hammouri, and B. Sunar, “Towards Robust Low Cost Authentication for Pervasive Devices”, *PerCom*, 2008, pp. 170-178.
[126] G. Hammouri, E. Ozturk, and B. Sunar, “A Tamper-Proof and Lightweight Authentication Scheme, *Pervasive and Mobile Computing*, 2008, 807-818.
[127] L. Kulseng, Z. Yu, Y. Wei, and Y. Guan, “Lightweight Mutual Authentication and Ownership Transfer for RFID Systems”, *INFOCOM*, 2010, pp. 251-255.
[128] A.-R. Sadeghi, I. Visconti, and C. Wachsmann, “Enhancing RFID Security and Privacy by Physically Unclonable Functions”, *Information Security and Cryptography*, 2010, pp. 281-305.
[129] S. Katzenbeisser, Unal Kocabas, V. Van Der Leest, A. Sadeghi, G. J. Schrijen, H. Schroder, and C. Wachsmann, “Recyclable PUFs: Logically Reconfigurable PUFs”, *CHES*, 2011, pp. 374-389.
[130] U. Kocabas, A. Peter, S. Katzenbeisser, and A. Sadeghi, “Converse PUF-Based Authentication” *TRUST*, 2012, pp. 142-158.
[131] Y. S. Lee, T. Y. Kim, and H. J. Lee, “Mutual Authentication Protocol for Enhanced RFID Security and Anticounterfeiting”, *WAINA*, 2012, pp. 558-563.
[132] Y. Jin, W. Xin, H. Sun, and Z. Chen, “PUF-Based RFID Authentication Protocol against Secret Key Leakage”, *Lecture Notes in Computer Science*, Vol. 7235, 2012, pp. 318-329.
[133] Y. Xu and Z. He, “Design of a Security Protocol for Low-Cost RFID”, *WiCOM*, 2012, pp. 1-3.
[134] Y. S. Lee, H. J. Lee, and E. Alasaarela, “Mutual Authentication in Wireless Body Sensor Networks Based on Physical Unclonable Function”, *IWCMC*, 2013, pp. 1314-1318.

[135] M.-D. M. Yu, D. M'Rahi, I. Verbauwhede, and S. Devadas, "A Noise Bifurcation Architecture for Linear Additive Physical Functions, *HOST*, 2014, pp. 124-129.
[136] S. T. C. Konigsmark, L. K. Hwang, D. Chen, and M. D. F. Wong, "System-of-PUFs: Multilevel Security for Embedded Systems", *CODES*, pp. 27:1-27:10, 2014.
[137] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas, "Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching", *Symposium on Security and Privacy Workshop*, 2012, pp. 33-44.
[138] J. Delvaux, D. Gu, R. Peeters and I. Verbauwhede, "A Survey on Lightweight Entity Authentication with Strong PUFs", *Cryptology ePrint Archive*: Report 2014/977.
[139] D. Moriyama, S. Matsuo, M. Yung, "PUF-based RFID Authentication Secure and Private under Complete Memory Leakage", *IACR Cryptology ePrint Archive 2013*, 712 (2013), http://eprint.iacr.org/2013/712
[140] A. Das, U. Kocabas, A.-R. Sadeghi and I. Verbauwhede, "PUF-based Secure Test Wrapper Design for Cryptographic SoC Testing", *Design, Automation and Test in Europe*, 2012, pp. 866-869.
[141] C. Hoffman, M. Cortes, D. F. Aranha and G. Araujo, "Computer Security by Hardware-Intrinsic Authentication", *Hardware/Software Codesign and System Synthesis*, 2015, pp. 143-152.
[142] X. Wang,Y. Zheng, A. Basak, S. Bhunia, "IIPS: Infrastructure IP for Secure SoC Design", *Transactions on Computers*, Vol. 64, Issue 8, 2015, pp. 2226 - 2238.
[143] S. M. Trimberger, J. J. Moore, "FPGA Security: Motivations, Features, and Applications", *Proceedings of the IEEE*, 2014, pp. 1248-1265.