Lightweight and Secure PUF Key Storage Using Limits of Machine Learning

Meng-Day (Mandel) Yu[1], David M'Raihi[1], Richard Sowell[1], and Srinivas Devadas[2]

[1] Verayo Inc., San Jose, CA, USA
{myu,david,rsowell}@verayo.com
[2] MIT, Cambridge, MA, USA
devadas@mit.edu

Abstract. A lightweight and secure key storage scheme using silicon Physical Unclonable Functions (PUFs) is described. To derive stable PUF bits from chip manufacturing variations, a lightweight error correction code (ECC) encoder / decoder is used. With a register count of 69, this codec core does not use any traditional error correction techniques and is 75% smaller than a previous provably secure implementation, and yet achieves robust environmental performance in $65nm$ FPGA and 0.13μ ASIC implementations. The security of the syndrome bits uses a new security argument that relies on what *cannot* be learned from a machine learning perspective. The number of *Leaked Bits* is determined for each Syndrome Word, reducible using *Syndrome Distribution Shaping*. The design is secure from a min-entropy standpoint against a machine-learning-equipped adversary that, given a ceiling of leaked bits, has a classification error bounded by ϵ. Numerical examples are given using latest machine learning results.

Keywords: Physical Unclonable Functions, Key Generation, Syndrome Distribution Shaping, Machine Learning, FPGA, ASIC.

1 Introduction

Gassend et al. introduced silicon-based Physical Unclonable Functions (PUFs) in [5], [6]; PUFs generate responses based on device manufacturing variations. Given a challenge as input, a PUF produces a response that is based on manufacturing variations on a particular instance of a silicon device. As such, PUF responses are noisy; although most of the response bits stay the same from run to run, some of the bits may flip. PUF noise increases with a change in voltage, temperature, and age between the *provisioning* condition, where a reference snapshot of the response is taken, and the *regeneration* condition. To derive stable PUF bits, some form of error correction code (ECC) or equivalent function is required. A set of *syndrome* bits is generated during provisioning, to help correct the regenerated PUF response back to the provisioned snapshot. There have been relatively few works that explicitly address the issue of information leaked via syndrome bits in the context of key storage using PUFs.

B. Preneel and T. Takagi (Eds.): CHES 2011, LNCS 6917, pp. 358–373, 2011.
© International Association for Cryptologic Research 2011

Security arguments for the syndrome bits have taken several forms. The idea is to construct an argument that quantifies the amount of secrecy remaining in the provisioned PUF secret given that the syndrome is known to the adversary. One frequently cited work is by Dodis et al. [4], which contains an often-used result that Code-Offset Generic Syndrome with a code word length of n has an entropy loss of $n - \log A(n, 2t+1)$, where $A(n, 2t+1)$ represents the size of the largest code for a Hamming space with minimal distance of $2t+1$. Assuming an optimal code, the value $n - \log A(n, 2t+1)$ is also the size of the parity encoded as the syndrome. This result is useful to the extent that the number of parity bits in the error correction code does not exceed the min-entropy of the PUF bits that are used to derive the n-bit codeword. To safely account for the case where a large number of syndrome bits are used (a seemingly necessary tradeoff to reduce ECC complexity), additional security arguments may be useful. Yu and Devadas in [21] developed an alternative to Code-Offset Syndrome, using a technique called Index-Based Syndrome (IBS) coding, which was proven to be information-theoretically secure under the assumption that the PUF output bits are independent and identically distributed (i.i.d.); the security arguments apply even for a heavily biased (and thus min-entropy reduced) PUF.

Although the work of [21] achieved a quadratic reduction in *ECC complexity*, the work does not explicitly describe the *PUF complexity* required for producing the i.i.d. PUF output bits beyond a brief mention of using disjoint oscillator pairs. The PUF complexity (number of PUF elements, and in this case disjoint oscillator pairs per key bit) is $2520/128 = 19.7$, taking [21] at face value.[1] The current work, by contrast, describes a lightweight key storage mechanism that is lightweight both in terms of its ECC complexity *and* PUF complexity, by using the indexing scheme in [21] as a starting point and eliminating the BCH coding. It achieves a 75% reduction in ECC complexity compared to [21] and achieves a PUF complexity of 5 (using ten 64-sum PUFs, see Figure 1 for a description of k-sum PUFs) to as little as 1 (using two 64-sum PUFs); this is a 4x to 20x improvement. The PUF complexity reduction derives from a machine-learning-based security argument that each additional syndrome bit does not require a linear increase in the number of PUF elements (e.g., disjoint oscillator pair [18] [21] or a memory cell [1] [12] [7] [17] [8]) but instead relies on assumptions on what *cannot* be learned about a challengeable physical system.

Machine learning theory as pioneered in [20] is interested primarily in what *can* be learned. For example, the number of training samples needed to learn reasonably well a hypothesis class grows linearly with the Vapnik-Chervonenkis dimension of that class. In the context of learning a k-sum PUF, the theory suggests that the number of training samples required for learning grows linearly in the number of parameters in the learning model, which corresponds to the number of summation stages in the PUF. Empirical results in [13] [14] [15] show

[1] Using the example parameters described in [21], a 128-bit key would require 5 BCH(63,30,t=6) blocks, or 315 bits. Since each bit is coded using a 3-bit index, picking the best value out of 8 PUF output bits, a total of 2520 disjoint oscillator pairs are needed.

that the required number of training samples does in fact grow linearly with the number of delay sums. The current work turns this (apparent) weakness of a PUF into a strength by taking advantage of what cannot be learned; this results in reduction in the PUF complexity required. PUF complexity is the number of PUF elements per key bit; for a k-sum PUF (Figure 1), a PUF element is a pair of ring oscillators whose frequency difference effectively forms a stage in the PUF.

1.1 Contributions

The main contributions of the current work include the following:

- 75% reduction in *ECC complexity* because no traditional error correction codes are used. We note that [21] mentions this possibility; in this paper we provide extensive supporting experiments;
- 4x to 20x reduction in *PUF complexity*;
- ASIC implementation results. We believe this paper to be the first to give results on reliable key generation in an ASIC with integrated ECC;
- Accelerated aging result on stable PUF keys, well beyond published PUF aging results;
- A new metric, *Leaked Bits*, which entropically computes leakage per Syndrome Word;
- A new technique called *Syndrome Distribution Shaping*, to minimize *Leaked Bits*;
- A new security argument relating machine learning classification error ϵ to the average min-entropy remaining in the PUF-derived secret.

1.2 Related Works

To the best knowledge of the authors, the current work is the first to marry results from two fields: machine learning [13] [14] [15] and PUF-based key generation [1] [5] [11] [12] [18] [21]. [5] pioneered the use of error correction on PUF outputs using 2D Hamming codes. [4] provides a security framework for using Code-Offset Syndrome. [18] took a more robust approach to account for environmental noise using a single stage BCH(255) code. [1] used a two-stage coding approach, with the use of heavy first-stage repetition coding to reduce second-stage ECC complexity. [12] introduced the use of soft-decision decoding. The Code-Offset Syndrome in [4], however, yields at best very little (or in some cases negative) remaining min-entropy for some of the more efficient recent approaches where a large number of syndrome bits are produced (e.g., by a repetition coding stage) to reduce ECC complexity.[2] [21] introduced an alternative to Code-Offset Syndrome; under the assumption that PUF output bits

[2] Consider a PUF with a min-entropy = 0.8. If a (5,1,5) repetition code is used, 4 bits are leaked via syndrome (assume code is optimal). This is also the min-entropy of the 5 PUF output bits ($5 \times 0.8 = 4$) used to form the code word. No secrecy remains from a min-entropy standpoint.

are i.i.d., Index-Based Syndrome Coding (IBS) results in syndrome bits that are information-theoretically secure. The current work uses IBS as a starting point and eliminates the BCH coding of [21], and uses a security framework that eliminates the need for the i.i.d. PUF output assumption in [21] by deriving syndrome security based on what cannot be learned by a machine-learning-equipped adversary. This new security framework based on limits of machine learning has the effect of reducing PUF complexity.[3] The machine-learning-based syndrome security framework differs from and complements the syndrome security arguments derived in [4] and [21]; for example, operating in the regime where machine learning classification error $\epsilon = 0.5$ is essentially equivalent in security to using an i.i.d. PUF output assumption. The current work is also one of few published works which contains results of an integrated ASIC PUF + ECC implementation under environmental stresses, and complements FPGA results obtained in [21].

1.3 Organization

We describe the implementations of k-sum PUFs with associated error correction in Section 2. Section 3 establishes the empirical viability of the lightweight error correction scheme with respect to stability results (against voltage, temperature, and aging) and implementation complexity. Section 4 uses the empirically viable building blocks, consisting of the lightweight error correction coder plus one or more 64-sum PUFs, to derive Secure Constructions.

2 PUFs with Lightweight Error Correction

In this section, we describe the FPGA and ASIC implementations of PUFs and the error correction schemes that we are evaluating in Section 3.

A simplified high-level block diagram is shown in Figure 1. The basic 64-sum PUF looks at the difference between two delay terms, each produced by the sum of 64 ring oscillator delay values. The challenge bit C_i for each of the 64 stages determines which ring oscillator is used to compute the top delay term, and which is used to compute the bottom delay term. The sign bit of the difference between the two delay terms determines whether the PUF produces a $'1'$ output bit or a $'0'$ output bit for the 64-bit challenge $C_0 \cdots C_{63}$. The remaining bits of the difference determine the confidence level of the $'1'$ or the $'0'$ output bit. The k-sum PUF can be thought of as a k-stage Arbiter PUF [11] with a real-valued output (as opposed to a single bit output) that contains both the output bit as well as its confidence level. This information is used by the downstream lightweight error correction block, using the indexing scheme described in [21], coupled with a Syndrome Distribution Shaper (to be described in Section 4) to minimize syndrome leakage. The indexing scheme uses index sizes between 4 bits (choosing best out of 16 output bits) and 5 bits (best out of 32). If a $'1'$ bit is

[3] Reducing PUF complexity is more difficult to achieve with an i.i.d. PUF output assumption, where an increase in syndrome length requires no less than a linear increase in the number of PUF elements.

to be encoded, the location of the maximum (out of 16 for a 4-bit index, and out of 32 for a 5-bit index) is chosen and written out as the Syndrome Word; alternatively, if a $'0'$ bit is to be encoded, the location of the minimum is chosen and written out as the Syndrome Word.

The 0.13μ ASIC implementation contains multiple banks of 64-sum PUFs, a lightweight error correction engine (including Indexing algorithm and Syndrome Distribution Shaping algorithm), universal hashing [10], cryptographic functions, and various other logic. Various Xilinx FPGA versions were created as the design evolved; the final FPGA version included all the functionality mentioned above for the ASIC.

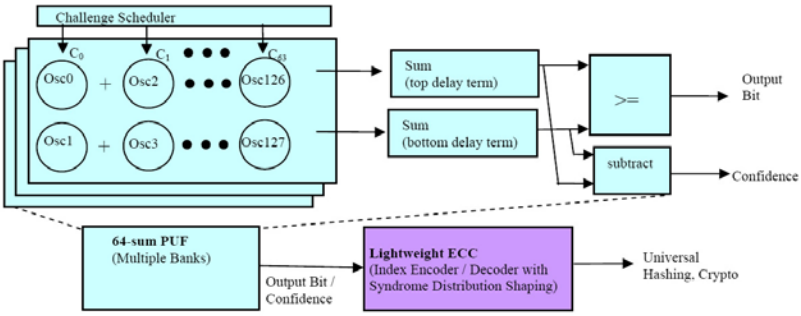

Fig. 1. Lightweight PUF Key Storage Block Diagram

3 Empirical Viability of Lightweight Error Correction

This section establishes the empirical viability of the lightweight error correction scheme, which is derived from the Index-only coding approach in [21], without the use of traditional ECC. The results in this section show a 75% reduction in error correction implementation complexity. Yet, when the indexing parameters are properly selected and applied in the context of the 64-sum PUF shown in Figure 1, stable PUF bits are derived under very extreme environmental conditions in FPGAs and ASICs.

3.1 Implementation Complexity

This section compares ECC complexity for three main classes of PUF error correction schemes using representatives from each:

1. Lightweight (Indexing only)
2. 2-stage ECC (Indexing + BCH63)
3. Large Block ECC (BCH255)

The analysis includes both encoder and decoder complexity and does not include I/O buffering, host interface logic, and other peripheral logic. Lightweight ECC has an implementation complexity that is estimated to be 75% smaller

than the two-stage scheme published in [21] (secure based on i.i.d. PUF output assumption) and an estimated 98% smaller than the single-stage scheme published in [18] (secure based on Dodis' framework). The results are summarized in Table 1 below. The SLICE utilization is minimal (1.2% of a modestly-sized Xilinx Virtex-5 LX50 SLICE count), containing only 69 registers.

Table 1. Three Classes of PUF Error Correction and Relative Complexities

Lightweight (This work)	**2-stage ECC** (From [21])	**Large Block** (From [18])
69 registers	471 registers	6400 registers (est. 16x)
1.2% SLICE count*(99/7200)	5% SLICE count*(393/7200)	65% SLICE count*

*Utilization of a modestly-sized Xilinx Virtex-5 LX50 device as a benchmark.

3.2 Stability

This section describes the performance of the Lightweight ECC with a 64-sum PUF. The results show that a 4-bit index is capable of achieving parts-per-million (ppm) level performance when provisioning is performed under nominal temperature and voltage (25°C, V_{nom}), and regeneration is performed under a fast-fast temperature-voltage corner (-55°C, V_{nom} + 10%) and a slow-slow temperature-voltage corner (125°C, V_{nom} - 10%). Figure 2 shows representative results for each corner, where a total of 1M+ error correction blocks using 4-bit indexing ran without errors for each corner using empirical data collected from Xilinx Virtex-5 FPGAs, with the 4-bit indexing post-processed in software using empirical PUF data. The data illustrates that ppm level stability is feasible, and better performance is achievable with either a larger index size (choosing best out of > 16) or using retry mechanisms if a failure is observed [21]. The average number of noisy bits is about 6 bits out of 63 in both cases, with the maximum number of noisy bits (for 1M+ blocks) at 9 out of 63 bits, and every single noisy bit was error corrected for all the cases that were run.

The empirical results also showed that under higher stress, a larger index size was required. For example, in the context of accelerated aging (Figure 3) where provisioning was performed at 25°C, 1.0V and regeneration at 125°C, 1.1V, an increase in index size by 0.25 bit is necessary (choosing best out of 20 instead of best out of 16) to achieve error-free performance. The analysis was performed using empirical PUF data from a Xilinx Virtex-5 FPGA device aged under high temperature and high voltage stress, with empirical PUF data extracted *in-situ* and 4.25-bit indexing (best out of 20) emulated as a post-processing step (in practice, this can be implemented using a 5-bit index, and choosing best out of 20 instead of best out of 32). Test parameters for accelerated aging were derived from *MIL-STD-883G Method 1005.8 Steady State Life* as well as accelerated aging parameters obtained from Xilinx. Specifically, 0.70eV activation energy was assumed, at a confidence level of 60% (same assumptions as those used by Xilinx). Over 80M+ blocks of PUF data were corrected, representing an accelerated life of 260+ years at 25°C and 20+ years at 55°C, with every single block error corrected using a 4.25-bit index for that entire dataset; this has an

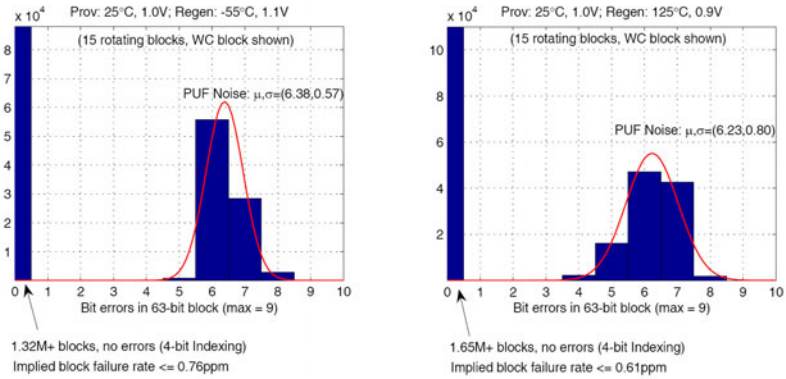

Fig. 2. Lightweight ECC performance, **WC Temperature / Voltage corners** (4-bit index). The right distribution in each plot is the PUF noise histogram before ECC, and the left distribution (at 0 errors) is the histogram after lightweight ECC.

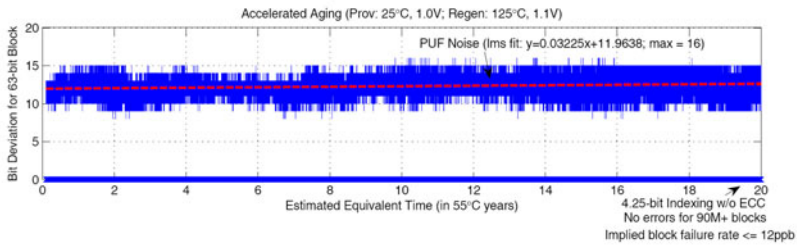

Fig. 3. Lightweight ECC performance, **Accelerated Aging** (4.25 bit index)

implied error rate of less than 12 parts per *billion*. As shown below, the average number of bits in error prior to indexing ranges from about 8 bits to 16 bits for a block size of 63 over 20 years at 55°C (or equivalently 260+ years at 25°C). The least mean square fit shows a slight upward slope of the PUF noise over this time. Yet with 4.25-bit indexing all the errors were corrected.

The FPGA results are consistent with results from a 0.13μ ASIC implementation (Figure 4), which has multiple 64-sum PUFs as well as the lightweight encoding / decoding algorithm integrated into a single device. The results in Figure 4 show that under extreme voltage conditions, 4-bit indexing (best out of 16) results in a 2.5ppm block failure rate, whereas 5-bit indexing (best out of 32) results in error-free performance. The integrated ASIC device (unlike the FPGA results above which emulated the indexing with empirical PUF data as a post-processing step to help algorithmic derivation) does not allow for fractional index sizes.

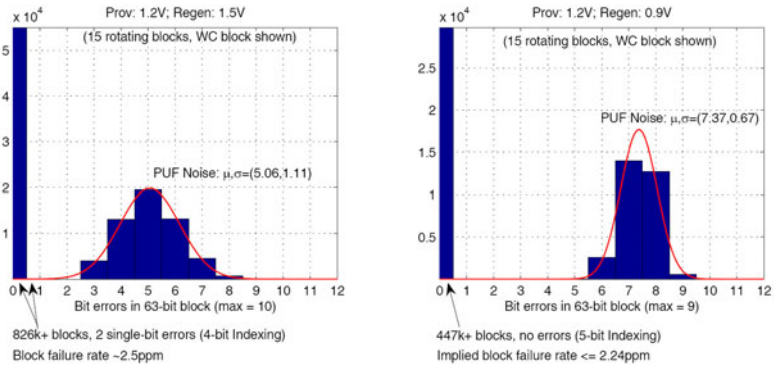

Fig. 4. 0.13μ ASIC with PUF + Lightweight ECC, **Extreme Voltage** performance. The right distribution in each plot is the PUF noise histogram before ECC, and the left distribution (near 0 errors) is the histogram after lightweight ECC.

4 Secure Constructions

The previous section demonstrated the empirical viability of a PUF + lightweight ECC combination. This section derives several Secure Constructions consisting of lightweight ECC and one or more 64-sum PUF blocks in the context of deriving a 128-bit key.[4] By adopting a machine-learning-based security argument instead of an i.i.d. PUF output argument, the number of ring oscillator pairs is reduced from 2520 to 640 (Secure Construction #1) or as little as 128 (Secure Construction #4). The PUF complexity reduction resulting from the machine-learning-based security argument is a result of the fact that each additional syndrome bit does not require a linear increase in the number of ring oscillators but instead relies on what cannot be learned about a challengeable physical system.

4.1 Unlearnable Bits

To determine what cannot be learned from a k-sum PUF, consider what is required to learn the delay differences of each pair of oscillators. A machine-learning-equipped adversary using a physical model of the PUF for learning starts with a model consisting of k parameters. The adversary also needs access to challenge/response pairs, for example, pairs consisting of k-bit challenges and 1-bit responses. Ruhrmair et al. in [14] derived an empirical equation relating the number of challenge/response (C/R) pairs N_{CRP}, number of parameters k, and the classification error rate ϵ as follows:

$$N_{CRP} \approx 0.5 \frac{k+1}{\epsilon}$$

[4] If additional key bits are required, then the entropy has to be increased, e.g., by doubling the number of 64-sum PUFs used for a 256-bit key.

The equation was derived using the best of results obtained from using Support Vector Machine (SVM), Logistical Regression (LR), and Evolution Strategy (ES) algorithms corresponding to an Arbiter delay PUF [11], including the case where $k = 64$. (While our PUF is not an Arbiter PUF per se, it has a very similar structure.) According to the equation, if k C/R pairs are known to the adversary for a k-parameter PUF, the adversary cannot do much better than guessing since the error rate $\epsilon = 0.5$. Intuitively, the results make sense; a k-parameter PUF would have at least k or more bits worth of parameter information (if each parameter is 1-bit, there would be k bits of information, and the parameter size likely needs to be a few bits for the machine learning to converge). As a result, if no more than k C/R pairs (each response is a single bit) are given out, no more than k bits of information are derived, and therefore the machine learning algorithm cannot infer much information.

4.2 Leaked Bits (LB)

We analyze several PUF Syndrome Coding algorithms, and describe their behavior with respect to *Leaked Bits*, the number of bits leaked per *Syndrome Word* for a particular Syndrome Coding algorithm, as defined below.

$$\mathbf{LB}(\underline{S}^{alg}) \equiv \mathbf{I}(\underline{S}^{alg}; \underline{M}^{\infty}) = \mathbf{H}(\underline{S}^{alg}) - \mathbf{H}(\underline{S}^{alg}|\underline{M}^{\infty})^5$$

where,

- $\underline{S}^{alg}$ is a random variable representing the Syndrome Word. Its variability comes from a particular syndrome coding algorithm used, denoted by the superscript alg.
- $\underline{M}^{\infty}$ is a random variable representing a PUF model that is perfect in predicting the PUF output bits (superscript ∞ denotes its perfect predicting ability). Its variability comes from PUF manufacturing variations.
- $\mathbf{H}$ is the *Shannon entropy* measure [2]

$$\mathbf{H}(\underline{X}) = -\sum_{x} p(x) \log_2 p(x)$$

  $\underline{X}$ is a random variable with a probability mass function $p(x)$, and the summation is taken over x over its entire alphabet
- $\mathbf{I}$ is the *mutual information* measure [2]

$$\mathbf{I}(\underline{Y}; \underline{X}) = \mathbf{H}(\underline{Y}) - \mathbf{H}(\underline{Y}|\underline{X})$$

  where $H(\underline{Y}|\underline{X}) = -\sum_{x} p(x) \sum_{y} p(y|x) \log_2 p(y|x)$. Note: $\mathbf{I}(\underline{Y}; \underline{X}) = \mathbf{I}(\underline{X}; \underline{Y})$.

Mutual information $\mathbf{I}$ computes the amount of information shared between two random variables. For example, in a cryptographic encryption system, the amount of information shared between the Ciphertext (denoted CT) and Key (i.e., information leaked by the Ciphertext about the Key from an information-theoretic standpoint) is

$$\mathbf{I}(\underline{CT}^{alg};\underline{Key}) = \mathbf{I}(\underline{Key};\underline{CT}^{alg}) = \mathbf{H}(\underline{Key}) - \mathbf{H}(\underline{Key}|\underline{CT}^{alg}).$$

To determine the amount of information leaked by a Syndrome Word about the PUF, we use the same concept, and call the result Leaked Bits, as defined above.

We now describe several syndrome coding algorithms that have been published in open literature, and analyze their information leakage with respect to Leaked Bits. We also analyze the use of a new technique called *Syndrome Distribution Shaping* to reduce Leaked Bits while preserving the average error correction power.

Code-Offset. In the Code-offset Method [4], the syndrome bits generated correspond to the XOR mask for a sequence of PUF output bits required to form a valid error correction code codeword. Consider a simple example of a binary 3x repetition code. Let the random variable $\underline{B}$ be a bit we want to store in a PUF.[6] Let the random variable $\underline{O}$ represent a sequence of PUF output bits corresponding to the error correction word size (3-bits in the 3x repetition coding example). Let the random variable $\underline{S}$ represent the corresponding Syndrome Word (3-bits in the 3x repetition coding example). The valid code words are (000) and (111). If we want to store a bit $\underline{B} = 0$, the valid code word (000) is used. Alternatively, if we want to store a bit $\underline{B} = 1$, the valid code word (111) is used. To generate the syndrome, three PUF output bits $O = o_0 o_1 o_2$ are required. The syndrome $\underline{S}$ using Code-Offset is the XOR mask required to make the PUF output bits $\underline{O} = o_0 o_1 o_2$ a valid code word, i.e., $\underline{S} = \underline{C} \wedge \underline{O}$, where $\wedge$ is the bitwise XOR operator.

Now, let's compute Leaked Bits using this example. In the 3x repetition example above, $H(\underline{S}^{3x}) = \log_2(\#(\underline{S}^{3x})) = 3$ bits, where # operator is the cardinality of the random variable, or the number of possibilities that the random variable can take. This is the amount of uncertainty of the 3-bit syndrome *not conditioned on any other knowledge*. Recall that $\underline{M}^{\infty}$ represents a perfect PUF model in that it has perfect knowledge in predicting PUF output bit $\underline{O}$. Given a perfect PUF model, the uncertainty remaining in $\underline{S}$ reduces to the uncertainty as to whether the valid code word is (000) or (111). That is, $\mathbf{H}(\underline{S}^{3x}|\underline{M}^{\infty}) = \log_2(\#(\underline{B})) = 1$ bit. Putting it together:

$$\begin{aligned}\mathbf{LB}(\underline{S}^{3x}) \equiv \mathbf{I}(\underline{S}^{3x};\underline{M}^{\infty}) &= \mathbf{H}(\underline{S}^{3x}) - \mathbf{H}(\underline{S}^{3x}|\underline{M}^{\infty}) \\ &= \log_2(\#(\underline{S}^{3x})) - \log_2(\#(\underline{B})) = 3 - 1 = 2 \text{ bits.}\end{aligned}$$

Two bits of information are leaked for each Syndrome Word derived from a 3x repetition codeword.

[6] Here, we consider the case where the PUF is used as a generalized key-store: the keying bit $\underline{B}$ can come from a source external to the PUF chip (e.g., user chosen key), or it can be derived from a PUF on the same chip.

Index-Based Syndrome (IBS) Coding. In Index-Based Syndrome coding [21], the Syndrome Word is an index lookup into a sequence of PUF output bits. Consider a simple example of a syndrome index with a width of 3 (i.e., $W = 3$, or a 3-bit index). $\mathbf{H}(\underline{S}^{3i}) = \log_2(\#(\underline{S}^{3i})) = 3$ bits. The 3-bit index takes on the value of the best out of $\#(\underline{S}^{3i}) = 8$ choices, requiring 8 PUF output bits $\underline{O} = o_0o_1o_2o_3o_4o_5o_6o_7$. If we want to store $\underline{B} = 0, \underline{S}^{3i} = \arg(\min_{j\in 0,1,\cdots,J-1}(o_j^r))$, where $J = 2^{W=3}$. (Superscript r of o^r denotes the real-valued output of the PUF, not just the binary $'1'$ or $'0'$ portion of the PUF output.) If we want to store $\underline{B} = 1, \underline{S}^{3i} = \arg(\max_{j\in 0,1,\cdots,J-1}(o_j^r))$. Now consider $\underline{M}^\infty$, which a perfect PUF model that predicts $\underline{O} = o_0o_1o_2o_3o_4o_5o_6o_7$ with perfect accuracy. Given a perfect PUF model, the uncertainty remaining in $\underline{S}$ reduces to the uncertainty as to whether the maximum or the minimum value is picked. That is, $\mathbf{H}(\underline{S}^{3i}|M^\infty) = \log_2(\#(\underline{B})) = 1$ bit. The amount of information leaked by $\underline{S}^{3i}$ about $\underline{M}^\infty$ is the Leaked Bits for $\underline{S}^{3i}$:

$$\begin{aligned}\mathbf{LB}(\underline{S}^{3i}) \equiv \mathbf{I}(\underline{S}^{3i};\underline{M}^\infty) &= \mathbf{H}(\underline{S}^{3i}) - \mathbf{H}(\underline{S}^{3i}|\underline{M}^\infty)\\ &= \log_2(\#(\underline{S}^{3i})) - \log_2(\#(\underline{B})) = 3 - 1 = 2 \text{ bits.}\end{aligned}$$

Two bits of information are leaked for each 3-bit index.[7]

Syndrome Distribution Shaping (SDS). Now, we present a new technique where we shape the syndrome distribution to minimize Leaked Bits while preserving, on average, error correction power. In its simplest form, the main idea is to enlarge the number of bits generated by $\underline{O}$ and to randomly select which of those bits would be used in forming a Syndrome Word and which would not be. As an example, visualize a case with a 3-bit Index-Based Syndrome. Let us order $o_0^ro_1^ro_2^ro_3^ro_4^ro_5^ro_6^ro_7^r$ from minimum to maximum, as $t_0^rt_1^rt_2^rt_3^rt_4^rt_5^rt_6^rt_7^r = \pi(o_0^ro_1^ro_2^ro_3^ro_4^ro_5^ro_6^ro_7^r)$, where π is a minimum-to-maximum sorting permutation. Here, t_0^r is the smallest $o_{j,j\epsilon 0\ldots J-1}^r$ value, t_1^r is the next smallest $o_{j,j\epsilon 0\ldots J-1}^r$ value, t_7^r is the largest $o_{j,j\in 0\ldots J-1}^r$ value. If there is equality in any of the comparisons, a random ordering among those is chosen.

Now, look at the unconditional probability of a particular 3-bit index value being selected. If we have no knowledge of the model or anything else, the probability of any index being selected is 1/8, i.e., $\mathbf{H}(\underline{S}^{3i}) = 3$ bits.

$$pr(o_0^r \text{ selected}) = 1/\#(\underline{S}^{3i}) = 1/8$$
$$\cdots$$
$$pr(o_7^r \text{ selected}) = 1/\#(\underline{S}^{3i}) = 1/8$$

Now, look at the probabilities conditioned upon $\underline{M}^\infty$, a perfect model, which allows us to sort the PUF output bits $t_0^rt_1^rt_2^rt_3^rt_4^rt_5^rt_6^rt_7^r = \pi(o_0^ro_1^ro_2^ro_3^ro_4^ro_5^ro_6^ro_7^r)$ and obtain:

$$pr(t_0^r \text{ selected}) = 1/\#(\underline{B}) = 1/2$$

[7] We are not making the assumption that PUF output bits are i.i.d., as in [21], under which the indices are provably secure.

$$
\begin{aligned}
pr(t_1^r \text{ selected}) &= 0 \\
&\cdots \\
pr(t_6^r \text{ selected}) &= 0 \\
pr(t_7^r \text{ selected}) &= 1/\#(\underline{B}) = 1/2
\end{aligned}
$$

Here, $\mathbf{H}(\underline{S}^{3i}|\underline{M}^{\infty}) = 1$ bit; either $o^r_{j,j\epsilon 0\ldots J-1} = t^r_0$ or $o^r_{j,j\in 0\ldots J-1} = t^r_7$ will be selected, depending on whether $\underline{B} = 0$ or $\underline{B} = 1$. Now, we want a randomization mapping that flattens the probability distribution while on the average preserving the error correction power. In the example above, a 3-bit index is used to choose the best PUF output value out of 8 choices. The distribution peaks on the ends: either the maximum or the minimum value will be selected. To flatten the distribution, one possibility is to use a 4-bit index, and randomly clobber or skip over half of the 16 choices such that the 4-bit index still chooses the best out of 8 values (same as the 3-bit index case).

This distribution will not peak at the ends (and be zero elsewhere), as is the case for a 3-bit index, but will be flatter (i.e., more uniformly distributed). More generally, consider an independent bit generator with output $\underline{R}$ where $pr(\underline{R} = 1) = p$ and $pr(\underline{R} = 0) = 1 - p = q$. Now, consider a 4-bit index ($W = 4$), where each of the 16 values ($J = 2^W = 16$) have a probability of p of being clobbered (i.e., skipped or not used). On the average, if $p = 0.5$, 8 values out of 16 will not be clobbered, and the maximum or the minimum out of 8 will still be selected, thus giving on the average a similar error correction power as the 3-bit index case (which also selects the maximum or minimum out of 8). More generally,

$$
\begin{aligned}
pr(t_{j\epsilon 0\ldots J-1} \text{ selected}) &= 1/2(p^j q + p^{J-1-j} q) \\
pr(\text{none selected}) &= p^J
\end{aligned}
$$

Assume in this example that if all values are clobbered we randomly choose a value. Practically, this distinction does not make a difference in this example, since the probability is very small:

$$pr(t_{j\epsilon 0\ldots J-1} \text{ selected}) = 1/2(p^j q + p^{J-1-j} q) + p^J/J$$

Here, the amount of uncertainty remaining when applying Syndrome Distribution Shaping (SDS) given a perfect PUF model is:

$$\mathbf{H}(\underline{S}|\underline{M}^{\infty}) = \mathbf{H}(pr(t_{j\in 0\ldots J-1} \text{ selected}))$$

Now, compute the Leaked Bits of 4-bit SDS index:

$$
\begin{aligned}
\mathbf{LB}(\underline{S}^{W=4,p=.5}) &= \mathbf{I}(\underline{S}^{W=4,p=.5}; \underline{M}^{\infty}) \\
&= \mathbf{H}(\underline{S}^{W=4,p=.5}) - \mathbf{H}(\underline{S}^{W=4,p=.5}|\underline{M}^{\infty}) \\
&= \log_2(\#(\underline{S}^{W=4,p=.5})) - \mathbf{H}(pr(t_{j\in 0..J-1} \text{ selected})) \\
&= 4 - 2.98 = 1.02 \text{ bits}
\end{aligned}
$$

Note that a $W = 4$-bit SDS index with a clobbering rate of 0.5 has a similar average error correction capability as a $W = 3$-bit index-based syndrome (i.e., both, on the average, select the strongest out of 8), and yet the Leaked Bits has been reduced by 50%, from 2 bits to 1.02 bits. By expanding the number of PUF output bits $\underline{O}$ and randomly eliminating them so that on the average we select the index from the same number of choices (i.e., preserving on average the same error correction power), we have lowered the number of bits leaked via each Syndrome Word from 2 bits to about 1 bit. Moving on to $W = 5$-bit syndrome, and $p = 0.75$, to preserve on the average the same error correction power, and beyond we have the following results:

$$\begin{aligned}
\mathbf{I}(\underline{S}^{3i}, M^{\infty}) &= 2 \text{ bits} \\
\mathbf{I}(\underline{S}^{W=4,p=1/2}, M^{\infty}) &= 1.02 \text{ bits} \\
\mathbf{I}(\underline{S}^{W=5,p=3/4}, M^{\infty}) &= 0.80 \text{ bits} \\
\mathbf{I}(\underline{S}^{W=6,p=7/8}, M^{\infty}) &= 0.71 \text{ bits} \\
\mathbf{I}(\underline{S}^{W=7,p=15/16}, M^{\infty}) &= 0.67 \text{ bits}
\end{aligned}$$

Note that between $W = 3$ and $W = 6$ there is almost a 3x improvement. Beyond $W = 6$ there are diminishing returns. Alternative SDS algorithms include ones that yield an even lower Leaked Bits, while others guarantee a certain minimum number of un-clobbered choices available by using only one side of the un-clobbered binomial distribution.

4.3 Secure Construction Examples

Now, armed with a metric (LB) to determine the number of bits leaked from each Syndrome Word, we describe different methodologies to derive secure constructions using the 64-sum PUF as a building block. The methodology requires an assumption describing the relationship between the classification error ϵ and Leaked Bits (and more precisely, a Leaked Bits sum ΣLB). In other words, we are establishing secure constructions assuming an $(\epsilon, \Sigma\text{LB})$ machine-learning-equipped adversary. This adversary cannot produce a classification error better than ϵ given that a total leaked bits of ΣLB. For the numerical examples below, we use the machine learning results in [14] as a proxy for the relationship between classification error and a sum of Leaked Bits;[8] this use has been preliminarily affirmed by the authors of this work using SVM and Simulated Annealing methods, and shall be further developed as future work.

[8] Formally, Leaked Bits for raw PUF responses can be computed using a null Syndrome notation: $\mathbf{LB}(\underline{S}^{\text{null}}) \equiv \mathbf{I}(\underline{S}^{\text{null}}; \underline{M}^{\infty}) = \mathbf{H}(\underline{S}^{\text{null}}) - \mathbf{H}(\underline{S}^{\text{null}} | \underline{M}^{\infty}) = 1 - 0 = 1$ bit, since the unconditional entropy of $\underline{S}^{\text{null}}$ (raw PUF response bit) is $\mathbf{H}(\underline{S}^{\text{null}}) = 1$ bit, and when conditioned with a perfect PUF model, $\underline{S}^{\text{null}}$ is completely known, i.e., $\mathbf{H}(\underline{S}^{\text{null}} | \underline{M}^{\infty}) = 0$ bit. As such, each bit leaked in the context of Leaked Bits (as defined) can also be interpreted as a leak of one equation for the PUF system with a 1-bit outcome.

Secure Construction #1: ΣLB *well within* $\epsilon = 0.5$. In this construction, we conservatively operate *well within* the regime of $\epsilon = 0.5$. Using the machine learning results in [14], we can choose to operate at a point where the Leaked Bits sum is no more than *half* the number of parameters in the PUF.

$$\Sigma LB \approx \frac{1}{2} 0.5 \left. \frac{k+1}{\epsilon} \right|_{\epsilon=0.5, k=64} = 32.5 \text{ bits per 64-sum PUF}$$

As an example, using a 6-bit index with a clobber rate of 5/8, the average error correction power is between 4 and 5-bit index (best out of 64 x 3/8 = 24 bits on the average). The LB is:

$$LB = \mathbf{I}(\underline{S}^{W=6,p=5/8}, \underline{M}^{\infty}) = 2.45 \text{ bits per Syndrome Word}$$

With $k = 64$ sum stages, we allow $\Sigma LB = k/2 = 32$ bits to be leaked per PUF; this translates to the use of 32 / 2.45 = 13 SDS indices. To generate a 128-bit key, we need ten (128/13) 64-sum PUFs, using 640 delay parameters to keep secret 128-bits worth of information (PUF complexity = 640/128 = 5). *It is likely a safe assumption that if less than $k/2$ equations are leaked from a k parameter PUF, a machine-learning-equipped adversary cannot learn much about the PUF since there remain $k/2$ degrees of freedom. This construction is formally equivalent to security obtained using an i.i.d. PUF output assumption, with a 2x margin on the certainty of $\epsilon = 0.5$.*

Secure Construction #2: ΣLB at $\epsilon = 0.5$ boundary. We remove the 2x margin on the certainty of $\epsilon = 0.5$, thus requiring half the number of PUFs. This construction is formally equivalent to security obtained using an i.i.d. PUF output assumption.

Secure Construction #3: *Challenge Modification on Block Boundary.* Here, we operate across multiple blocks in the range where $\epsilon \leq 0.5$, and compute the average min-entropy to account for cases where $\epsilon \neq 0.5$. Continuing the example from above, the first block of 26 SDS indices leaks 64 bits worth of information, but $\epsilon = 0.5$. Now, let's assume that the results of the first block (consisting of 26 data bits) are used to modify the challenge bits for the second block; that is, we use a *Challenge Modification Schedule* at the block boundary so that the 26 Syndrome indices for the second block cannot be used by the machine learning algorithm unless the first 26 bits are guessed or estimated correctly. The machine learning algorithm requires input / output sets, i.e., Challenge/Syndrome sets in our case, in order to train the delay parameters and the Challenges are known for the second block 0.5^{26} of the time. Now, let's compute average min-entropy of this chained scheme given that after the first block $\epsilon \neq 0.5$.

First, we recall the definition of min-entropy $\mathbf{H}_{\infty}(.) \equiv -\log_2(Pr_{max}(.))$ and average min-entropy $\tilde{\mathbf{H}}_{\infty}(\underline{X}|\underline{Y}) \equiv -\log_2(E_{y \leftarrow \underline{Y}}[2^{-\mathbf{H}_{\infty}(\underline{X}|\underline{Y}=y)}])$ using the definitions and notation from [4].

In our context: $\tilde{\mathbf{H}}_{\infty}(\underline{P}|\underline{CS}) \equiv -\log_2(E_{cs \leftarrow \underline{CS}}[2^{-\mathbf{H}_{\infty}(\underline{P}|\underline{CS}=cs)}])$ where $\underline{P}$ is a random variable predicting all $\#(P)$ PUF-derived bits, and $\underline{CS} = cs$ a *subset* of the available Challenge/Syndrome sets used for regenerating $\underline{P}$. The subset is

based on the number of Challenge/Syndrome sets that is known to the adversary *at any one time* as the result of the Challenge Modification Schedule.

Now let's consider the case where two blocks are generated using a 64-sum PUF, with syndrome modification at the block boundary. There is $\gamma = 0.5^{\#(P)/2}$ probability that the syndrome for the second block is useful; this is for the case where the 26 bits of the first block are guessed or estimated correctly.

$$\begin{aligned}\tilde{\mathbf{H}}_\infty(\underline{P}|\underline{CS}) &\equiv -\log_2(E_{cs\leftarrow\underline{CS}}[2^{-\mathbf{H}_\infty(\underline{P}|\underline{CS}=cs)}]) \\ &= -\log_2\{[(1-\gamma)\max(\epsilon, 1-\epsilon)^{\#(P)}]|_{\epsilon=0.5,\#(P)=52,\gamma=0.5^{26}} \\ &+ [\gamma\max(\epsilon, 1-\epsilon)^{\#(P)}]|_{\epsilon=0.25,\#(P)=52,\gamma=0.5^{26}}\} = 47.51 \text{ bits}\end{aligned}$$

The number of PUFs required is reduced to three (128 / 47.51) for 128-bit secret (PUF complexity = 3 x 64 / 128 = 1.5), if we assume that the machine learning result in [14] is a good proxy in estimating ϵ.

Secure Construction #4: *Challenge Modification on Syndrome Word Boundary.* Here, the challenge schedule is deviated once per Syndrome Word (e.g., index) vs. once per block. Due to space constraints, the derivation is omitted. Results for different number of blocks extracted per 64-sum PUF are below:

- 1 Block: $\tilde{\mathbf{H}}_\infty(\underline{P}|\underline{CS})|_{\#(P)=26} = 26$ bits
- 2 Blocks: $\tilde{\mathbf{H}}_\infty(\underline{P}|\underline{CS})|_{\#(P)=52} = 52$ bits
- 3 Blocks: $\tilde{\mathbf{H}}_\infty(\underline{P}|\underline{CS})|_{\#(P)=78} = 70$ bits

Two PUFs (128/70) are required for a 128-bit secret (PUF complexity = 1).

5 Conclusions

A PUF-based key storage is built using a lightweight ECC, without the use of traditional error correction techniques, and one or more 64-sum PUFs. The ECC complexity is low, with a register count of 69 for the encoder / decoder core, yet producing robust environmental stability results on FPGAs and ASICs. To our knowledge, this is the first time an integrated key generator ASIC implementation has been evaluated. We presented a new security argument that relies on what *cannot* be learned from a machine learning perspective, allowing a large reduction in PUF complexity. Future work includes further validation and refinements of the machine learning results in [14], applying the machine learning security method to XOR'ed PUFs (that are more difficult to learn), and methods to de-rate the ϵ vs. leaked bits curve to account for side channel leaks.

References

1. Bösch, C., Guajardo, J., Sadeghi, A.-R., Shokrollahi, J., Tuyls, P.: Efficient Helper Data Key Extractor on FPGAs. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 181–197. Springer, Heidelberg (2008)
2. Cover, T., Thomas, J.: Elements of Information Theory, 2nd edn. (2006)

3. Devadas, S., Suh, E., Paral, S., Sowell, R., Ziola, T., Khandelwal, V.: Design and Implementation of PUF-Based 'Unclonable' RFID ICs for Anti-Counterfeiting and Security Applications. In: Proc. RFID 2008, pp. 58–64 (May 2008)
4. Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data (2008)
5. Gassend, B.: Physical Random Functions, Master's Thesis, EECS, MIT (2003)
6. Gassend, B., Clarke, D., van Dijk, M., Devadas, S.: Silicon Physical Random Functions. In: Proc. ACM CCS, pp. 148–160. ACM Press, New York (2002)
7. Guajardo, J., Kumar, S., Schrijen, G., Tuyls, P.: FPGA intrinsic pUFs and their use for IP protection. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007)
8. Holcomb, D., Burleson, W., Fu, K.: Initial SRAM State as a Fingerprint and Source of True Random Numbers for RFID Tags. In: Conf. RFID Security (2007)
9. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
10. Krawczyk, H.: LFSR-based hashing and authentication. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 129–139. Springer, Heidelberg (1994)
11. Lim, D.: Extracting Secret Keys from Integrated Circuits, MS Thesis, MIT (2004)
12. Maes, R., Tuyls, P., Verbauwhede, I.: A Soft Decision Helper Data Algorithm for SRAM PUFs. In: IEEE ISIT 2009. IEEE Press, Los Alamitos (2009)
13. Ruhrmair, U.: On the Foundations of Physical Unclonable Functions (2009)
14. Ruhrmair, U., Sehnke, F., Solter, J., Dror, G., Devadas, S., Schmidhuber, J.: Modeling Attacks on Physical Unclonable Functions. In: Proc. ACM CCS (October 2010)
15. Sehnke, F., Osendorfer, C., Sölter, J., Schmidhuber, J., Rührmair, U.: Policy gradients for cryptanalysis. In: Diamantaras, K., Duch, W., Iliadis, L.S. (eds.) ICANN 2010. LNCS, vol. 6354, pp. 168–177. Springer, Heidelberg (2010)
16. Skorobogatov, S.P.: Semi-Invasive Attacks: A New Approach to Hardware Security Analysis. Univ. Cambridge, Computer Lab.: Tech. Report (April 2005)
17. Su, Y., Holleman, J., Otis, B.: A 1.6pJ/bit 96 (percent) Stable Chip ID Generating Circuit Using Process Variations. In: ISSCC 2007, pp. 200–201 (2007)
18. Suh, G.: AEGIS: A Single-Chip Secure Processor, PhD thesis, EECS, MIT (2005)
19. Suh, G., Devadas, S.: Physical Unclonable Functions for Device Authentication and Secret Key Generation. In: DAC 2007, pp. 9–14 (2007)
20. Vapnik, V., Chervonenkis, A.: On the uniform convergence of relative frequencies of events to their probabilities. Theory of Prob. and its App. (1971)
21. Yu, M., Devadas, S.: Secure and Robust Error Correction for Physical Unclonable Functions. IEEE D&T 27(1), 48–65 (2010)