Maximum-Likelihood Decoding of Device-Specific Multi-Bit Symbols for Reliable Key Generation

Meng-Day (Mandel) Yu*†§, Matthias Hiller‡, Srinivas Devadas§
*Verayo, Inc., San Jose, CA, USA
myu@verayo.com
†COSIC / KU Leuven, Belgium
‡Institute for Security in Information Technology / Technische Universität München, Germany
matthias.hiller@tum.de
§CSAIL / MIT, Cambridge, MA, USA
devadas@mit.edu

***Abstract*—We present a PUF key generation scheme that uses the provably optimal method of *maximum-likelihood* (ML) detection on *symbols* derived from PUF response bits. Each device forms a noisy, device-specific symbol constellation, based on manufacturing variation. Each detected symbol is a letter in a codeword of an error correction code, resulting in *non-binary* codewords.**

We present a three-pronged validation strategy: i. mathematical (deriving an optimal symbol decoder), ii. simulation (comparing against prior approaches), and iii. empirical (using implementation data). We present simulation results demonstrating that for a given PUF noise level and block size (an estimate of helper data size), our new symbol-based ML approach can have orders of magnitude better bit error rates compared to prior schemes such as block coding, repetition coding, and threshold-based pattern matching, especially under high levels of noise due to extreme environmental variation. We demonstrate environmental reliability of a ML symbol-based soft-decision error correction approach in 28*nm* FPGA silicon, covering -65°C to 105°C ambient (and including 125°C junction), and with 128-bit key regeneration error probability $\leq$ 1 ppm.

***Index Terms*—Physical Unclonable Function, Maximum-Likelihood, Non-Binary Codewords, Soft-Decision, Key Generation**

I. Introduction

Silicon Physical Unclonable Function (PUF) [8] [9] circuits exploit CMOS manufacturing variation in security applications, allowing a silicon device to be authenticated based on a challenge/response evaluation of difficult-to-replicate manufacturing variation. Since evaluation of a PUF response is *physical*, PUF response bits are noisy and noise increases with environmental variation. For a *key generation* use case, the basic PUF circuit needs error correction to account for evaluation noise; this requires *helper data*.

Figure 1 offers a codeword-centric view of PUF error correction, using code-offset helper data [6] [12]. During provisioning, keying material KM is used as input to an error correction code (ECC) encoder to produce a codeword CW. The codeword is bit-wise XOR'ed with a reference query of the PUF response RS, and stored as helper data HD. During regeneration, HD enables the return of a noisy query of the PUF response RS' back to a codeword with noise CW'; the regenerated PUF response is transformed into a form where ECC decoding can be applied to remove noise. In effect, PUF is used to "store" KM using chip-unique manufacturing variation where KM can be hashed/down-mixed into a key (hash not shown in Figure).[1]

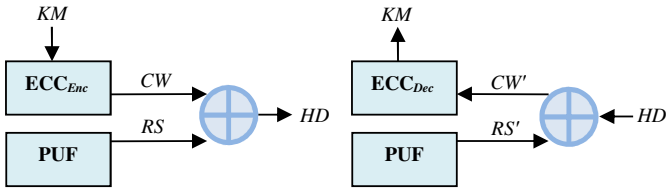

Fig. 1. PUF error correction using code-offset for provisioning (left) and regeneration (right): HD relates RS' to CW' to allow error decoding.

It is natural to think of codewords as composed of binary values vs. symbols from a higher-order alphabet. In our work, we perform PUF error correction using symbol-level (vs. bit-level) recovery and non-binary codewords. Our proof-of-concept *Maximum-Likelihood Symbol Recovery* (MLSR) implementation reduced bit errors to $\sim 0.01\%$ at a 125°C key regeneration junction temperature (provisioning at room temperature), and produced a soft-decision metric that allows a simple soft-decision decoder to "mop up" remaining errors. Our soft-decision symbol parity decoder also uses ML decoding and corrects the most-likely corrupted symbol based on the least distance between the most-likely and 2^{nd}-most-likely symbols, reducing the $\sim 0.01\%$ bit failure rate to show an empirical key regeneration failure rate $\leq 1 \times 10^{-6}$.

A. Contributions

The main contributions of the current work are:

- first to use ML symbol-based decoding on PUF bits
- first PUF error correction scheme capable of *orders of magnitude* bit error improvement over the standard repetition coding and BCH approaches

[1]Figure 1 assumes an externally chosen secret [12]. Using the syndrome construction [6], the secret (i.e., KM) can be internally derived, from RS.

- new method to help counter recent helper data attacks based on linear-XOR or linear-shift malleability

B. Organization

Section II provides background. Our MLSR method is described in Section III, and validated in Section IV using a three-pronged (mathematical, simulation, empirical) approach. Section V covers recent helper data manipulation attacks which MLSR helps to address. We conclude in Section VI.

II. Background

A. Related Work

PUF Key Generation. Silicon PUF key generation was first introduced using Hamming codes in [9] and more details were presented in [22]. More practical implementations that accounted for a wider range of environmental variation were described in [1] [16] [23] [19] [24] [17] [14] [11]. The recent results in [18] at 22*n*m have restricted temperature data (25°C to 50°C) that does not cover many operating conditions.[2]

Helper Data Manipulation. The issue of helper data manipulation was addressed for the biometric realm in [2] [7], with robust fuzzy extractors, but not so much in a manner specific to silicon PUFs. Their use of a helper data hash does not address recent helper data manipulation attacks in [13] [5]. A more recent work [11] also addresses helper data manipulation and in the context of a PUF but requires a hash function.

Maximum-Likelihood (ML). ML is a provably optimal decoding method used to recover bits transmitted over a noisy physical medium [20]. Repetition coding, which was used for PUF key generation in [1] [17] [14], uses a decoder that can be interpreted as ML with (degenerate) one-bit symbols. A recent publication [11] uses ML for 2^{nd} stage decode but not 1^{st}. We are the first to apply ML on multi-bit symbols, and to do so early on, during the 1^{st} stage, which results in orders of magnitude bit error improvements over standard methods.

B. PUF Building Blocks

Arbiter PUFs. Arbiter PUFs were introduced in [15] (Figure 2, top). PUF output is derived from a race condition formed by successive delay stages each comprising a crossbar switch. A challenge bit $x^i, 0 \leq i < k-1$, for each of the k stages determines whether the parallel path or cross path is active. Collectively, k challenge bits determine which path is chosen to create the transition at the top input to the arbiter latch, and similarly for bottom input. The difference comparison between the two delays determines whether the PUF produces a "1" vs. "0" response bit. Design layout has to be symmetrically matched such that random manufacturing variation would affect the response.

k-sum Ring Oscillator PUFs. In a k-sum PUF [24], each of k stages is replaced with a pair of identically manufactured ring oscillators (Figure 2, middle). Delay propagation is mimicked by digitally adding delay values for each stage depending on the challenge bit associated with that stage. Mathematically,

[2]Typical commercial (junction) temperature range: 0°C to 85°C; for industrial: -40°C to 100°C; and for military: -55°C to 125°C.

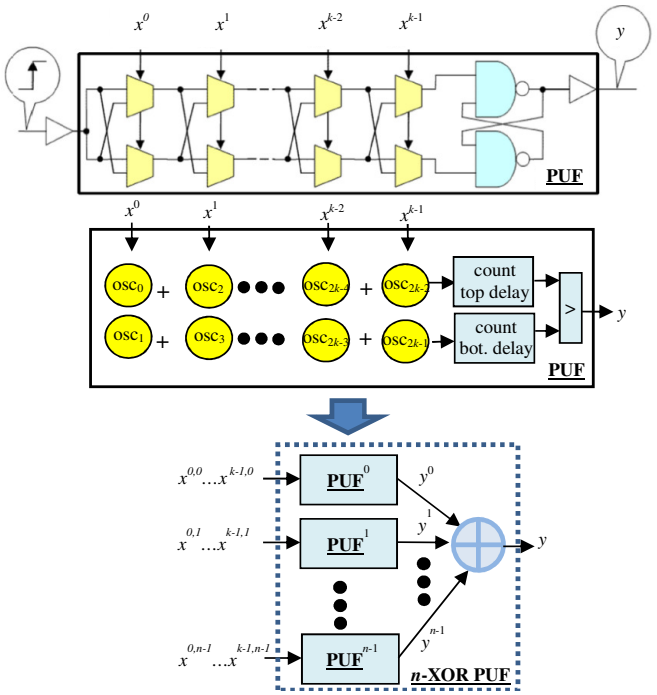

Fig. 2. Basic Arbiter PUF (top); k-sum Ring Oscillator equivalent (middle); n-XOR construction (bottom).

k-sum PUF and Arbiter PUF can be reduced to the same topology by a linear remapping of the challenge bits [21].

C. n-XOR Construction

The basic Arbiter PUF (and its equivalent k-sum PUF) can be modeled using machine learning attacks [21] when a *sufficient number* of challenge/response pairs (CRPs) is exposed. To scale up security against modeling attacks, XORs can be applied to outputs from n copies of these manufacturing variation building block primitives (Figure 2, bottom), to require more CRPs, computational power, time, etc. to model the composite n-XOR PUF.

III. Maximum-Likelihood Symbol Recovery

A. Adapting n-XOR PUF for Key Generation

The n-XOR PUF, in its native form, is an "open system" in that user and adversary alike have unrestricted access to its challenge/response interface. When used for *authentication*, the number of exposed CRPs is potentially very large, easily in the millions or billions, making the PUF susceptible to machine learning predictive attacks. Fortunately, in our specific *key generation* use case (similar to [19]), we generate a *fixed* number of keying bits from the PUF, thus we reveal only a *limited* number of response bits to the adversary, e.g., several thousand bits.

Our implementation uses 4-XOR (programmable to more) and exports $\leq$ 3000 response bits as helper data for a 128-bit key.[3] Multiple research groups with attacks spanning 3+ years [21] [4] require more than 3000 CRPs to break 4-XOR PUF.

[3]The entire challenge schedule, starting from the seed, is hardwired.

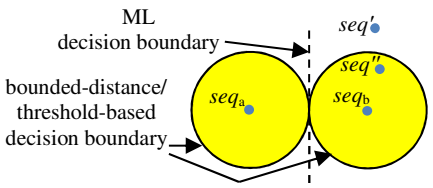

Fig. 3. ML vs. bounded-distance/threshold-based: seq'' decodes correctly for both approaches but seq' decodes properly only for ML.

B. ML vs. Bounded-Distance/Threshold-Based

We offer two explanations why ML has a lower error probability than a *bounded-distance* decoding approach associated with most PUF block decoders (e.g., BCH) [1] [23] [17], and equivalently for a *threshold-based* approach [19]:

1) Visual: Figure 3 shows two sequences, with the goal of deciding whether a received sequence belongs to seq_a or seq_b. In bounded-distance/threshold-based decode, seq'' decodes properly to seq_b, since it is *within* a certain preset distance or threshold represented by the circle around seq_b. On the other hand, seq' decodes improperly since it falls outside the circle. In ML, since seq' is closer to seq_b than to seq_a (i.e., falls to the right of dotted line), even though it falls outside the circle, it still decodes properly to seq_b.

2) Toy Example: Let $seq_a = 1010101$, and $seq_b = 1010010$. If a received sequence is 0101010, although 4 out of 7 bits have flipped with respect to seq_b, ML decodes to seq_b since received sequence is closer to seq_b than seq_a.

C. Device-Specific Symbols with Ease of ML Decode

We treat PUF response bits as l-bit sequences through which we encode and decode α-bit symbols. This is done for β processing blocks. Now, we establish some notation.

- PUF is a basic PUF building block (with no XOR), $\underline{\text{PUF}}^i$ is the i^{th} PUF instance.
- KM is keying material, HD is helper data ($l \cdot \beta$ bits).
- RS is a response sequence of l bits, and RS^{PUFi} is a l-bit response sequence from i^{th} PUF.
- s is an α-bit symbol. s' is the ML recovered version. s'' is the second-most-likely recovered version.
- $\boldsymbol{xresp^s}$ is the l-bit response sequence (helper data block) obtained after all RS^{PUF*} selected using s are XOR'ed.
- $\boldsymbol{xresp^?}$ is helper data block read back into the decoder.
- prime $'$ means a value derived during regeneration.
- c' is soft-decision confidence information for a symbol.

Now, we shall encode in a manner that yields a simple ML decoder. We give an α=4-bit example:

1) Symbol Encoder: HD is a function of symbol bits (derived from KM) and RS from multiple PUFs. Symbol bits s[0...3] each drive a mux select line of a two-bit to one-bit mux (Figure 4, top). The construction looks like n-XOR PUF except 2:1 muxes are used to select which PUF response RS^{PUF*} to use prior to the XOR function. The XOR'ed l-bit result, denoted $\boldsymbol{xresp^s}$, forms the helper data HD for one symbol. Encoder uses four 2:1 muxes and one XOR gate.

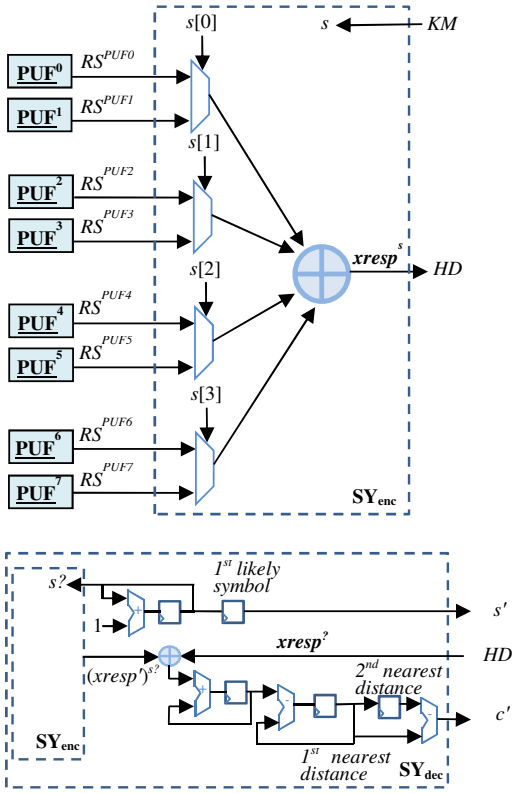

Fig. 4. 4-bit Symbol Encoder (top), 4-bit MLSR Symbol Decoder (bottom)

2) Symbol Decoder: Figure 4, bottom, depicts a symbol decoder, which contains a symbol encoder. The helper data from encoding $\boldsymbol{xresp^s}$ is read in as $\boldsymbol{xresp^?}$ since the symbol has yet to be recovered. To perform ML decoding, a 4-bit counter is used to iterate through all 16 symbols (indicated by $s?$), and the most likely symbol is stored. The symbol with minimum distance is found by performing bit-wise XOR of $\boldsymbol{xresp^?}$ (from HD) and candidate regenerated l-bit sequence $(xresp')^{s?}$ representing a symbol,[4] and incrementing a counter whenever a bit mismatch is seen (using one XOR gate and one $log_2(l)$-bit adder). A comparator (subtractor) is used to find 1^{st} nearest distance and 2^{nd} nearest distance (uses two stage buffering of $log_2(l)$ bits each), and their difference (using a subtractor) computes soft-decision confidence c'.

A 4-bit symbol encoder/decoder with l=128 bits requires 12 SLICES (440 gate equivalents) in a Xilinx Artix-7 FPGA.

D. 4-bit Symbol Encoder / Decoder Pseudo-Code

Here, $\oplus$ is bit-wise XOR, (.)? is an if-then-else statement, and wt is the Hamming weight operator counting the number of 1s. Note that tmp^* (temporary variable), RS^*, $\boldsymbol{xresp^*}$ are each l bits, and $dist$ (symbol distance) is $log_2(l)$ bits.

[4] ***bold italics*** indicate a helper data block; $(xresp')^{s?}$ is not a helper data block but an intermediate result used during symbol recovery.

Symbol Encoding:

$$\begin{aligned} tmp^a &= (s[0] == 1)?\ RS^{PUF1} : RS^{PUF0} \\ tmp^b &= (s[1] == 1)?\ RS^{PUF3} : RS^{PUF2} \\ tmp^c &= (s[2] == 1)?\ RS^{PUF5} : RS^{PUF4} \\ tmp^d &= (s[3] == 1)?\ RS^{PUF7} : RS^{PUF6} \\ \boldsymbol{xresp^s} &= tmp^a \oplus tmp^b \oplus tmp^c \oplus tmp^d \end{aligned}$$

Symbol Decoding:

$$\begin{aligned} dist^0 &= wt[\boldsymbol{xresp^?} \oplus RS'^{PUF0} \oplus RS'^{PUF2} \\ &\quad \oplus RS'^{PUF4} \oplus RS'^{PUF6}] \\ dist^1 &= wt[\boldsymbol{xresp^?} \oplus RS'^{PUF1} \oplus RS'^{PUF2} \\ &\quad \oplus RS'^{PUF4} \oplus RS'^{PUF6}] \\ &\vdots \\ dist^{15} &= wt[\boldsymbol{xresp^?} \oplus RS'^{PUF1} \oplus RS'^{PUF3} \\ &\quad \oplus RS'^{PUF5} \oplus RS'^{PUF7}] \end{aligned}$$

$$\begin{aligned} s' &= \underset{\hat{s}}{argmin}\ wt[dist^{\hat{s}}], \quad \hat{s} \in \{0, \ldots, 2^\alpha - 1\} \\ s'' &= \underset{\hat{s}}{argmin}\ wt[dist^{\hat{s}}], \quad \hat{s} \in \{0, \ldots, 2^\alpha - 1\} \backslash \{s'\} \\ c' &= dist^{s''} - dist^{s'} \end{aligned}$$

E. Concatenating Symbol Encoding/Decoding with ECC

As shown in the prior section, symbol encoder/decoder operations are quite simple. To incorporate symbol encoding/decoding into Figure 1, each codeword position, instead of being a single bit, is now an α-bit symbol as seen in Figure 5; there are β symbols in a codeword, processed as β processing blocks. The XOR function to produce helper data HD (Figure 1) is replaced by symbol encoder/decoder in Figure 4. Instead of having just one long response, there is now one l-bit RS for each of the β processing blocks and per PUF.

ECC changes from being strictly binary to non-binary, operating on an alphabet of α bits per symbol. To support soft-decision ECC, symbol decoder also outputs confidence c', one per symbol computed using symbol distances.

As we shall show in Section IV.B, the symbol coding stage offers orders of magnitude better bit failure probabilities compared to conventional PUF approaches using BCH, repetition coding, or pattern matching, given the same PUF noise level applied to all these approaches and given roughly the same helper size estimate. To "mop up" residual errors, the concatenated ECC stage can be relatively simple. Our implementation supports a soft-decision single-parity-symbol decoder. The ECC decoder is also optimal in the ML sense, using c' values to determine the most-likely corrupted symbol and have it swapped if the parity symbol check fails.

IV. Validation

A. Mathematical: Deriving an Optimal Decoder

Optimal Symbol Decoding. A maximum-a-posteriori (MAP) decoder [20] is, by definition, the optimal decoder from the standpoint of minimizing bit error probabilities. In our context,

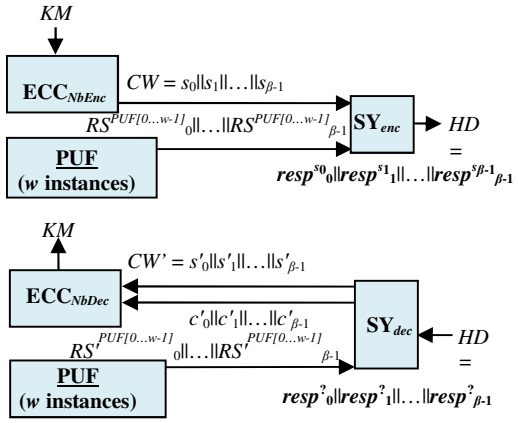

Fig. 5. Symbol-based provisioning (top) and regeneration (bottom).

MAP symbol decoder would maximize the probability of a correct symbol given a helper data (XOR'ed) block:

$$s' = \underset{\hat{s}}{argmax}\ Pr[\hat{s}|\boldsymbol{xresp^?}] \tag{1}$$

where $\hat{s} \in \{0, 1, \ldots, 2^\alpha - 1\}$. It is not clear what a direct implementation of this looks like and how complex it would be.

Assuming Uniform Symbol Distribution. We now derive an optimal symbol decoder *assuming* that symbols s are uniformly distributed[5] (s is random if KM is chosen at random). We can express the MAP decoder equation above in terms of Bayes' rule and derive a ML [20] decoder with a relatively simple implementation. Applying Bayes' rule and accounting for uniform distribution and that the helper data appears with a probability of 1, for $\hat{s} \in \{0, 1, \ldots, 2^\alpha - 1\}$:

$$\begin{aligned} Pr[\hat{s}|\boldsymbol{xresp^?}] &= Pr[\boldsymbol{xresp^?}|\hat{s}] \cdot Pr[\hat{s}]\ /\ Pr[\boldsymbol{xresp^?}] \\ &= Pr[\boldsymbol{xresp^?}|\hat{s}] \cdot 2^{-\alpha}\ /\ 1 \end{aligned} \tag{2}$$

So instead of maximizing $Pr[\hat{s}|\boldsymbol{xesp^?}]$ using MAP, we can alternatively maximize $Pr[\boldsymbol{xresp^?}|\hat{s}]$ using ML. However, during regeneration, the only way the decoder can represent $\hat{s}$ is by looking at $(xresp')^{\hat{s}}$ (i.e., we need the encoder function during the decoding process). So using ML, we maximize:

$$Pr[\boldsymbol{xresp^?}|\hat{s}\ ] = Pr[\boldsymbol{xresp^?}|(xresp')^{\hat{s}}] \tag{3}$$

This is equivalent to minimizing Hamming distance between helper data block and all possible regenerated choices:

$$s' = \underset{\hat{s}}{argmin}\ wt[\boldsymbol{xresp^?} \oplus (xresp')^{\hat{s}}], \tag{4}$$

$\hat{s} \in \{0, 1, \ldots, 2^\alpha - 1\}$. We have thus shown that our ML decoder (Eqn. (4)) is in fact an optimal decoder if we assume that the symbols s have uniform distribution, and in fact Eqn. (4) has a simple implementation as was shown in Sections III.C and III.D.

[5]If symbols are not uniform, optimality of ML decoder gets impacted; security is compensated by generating more symbols s before a down-mixing hash.

B. Simulation: Analytical and Monte Carlo Results

Deriving MLSR error probability. The output bit error probability of a scheme is an important performance measure to evaluate it. We analyze MLSR with a sequence (helper data block) length l and alphabet size $z = 2^{\alpha}$. Let (Hamming) distance between two random reference sequences be d. The regenerated sequence is interpreted as one of the reference sequences plus noise.

The correct symbol is detected, if less than $\lfloor \frac{d-1}{2} \rfloor$ errors occur in the d positions where two sequences a and b differ. For a distance of d_a between the regenerated sequence and sequence a, and d_b between the regenerated sequence and sequence b, an error in the sequence positions with identical values would increase both distances by 1 such that $d = d_a - d_b$ still remains constant and the decision is not affected.

Therefore, for an input PUF bit error probability p and a distance d, a detection error p_e between symbols a and b occurs with

$$p_e(d) = 1 - \sum_{i=0}^{\lfloor \frac{d-1}{2} \rfloor} \binom{d}{i} p^i (1-p)^{d-i} \tag{5}$$

Depending on l, distances d_i occur with a probability

$$\Pr[d = d_i] = \binom{l}{d_i} \cdot 2^{-l} \tag{6}$$

Summing over all possible distances and their error probabilities gives an average overall error probability for a detection error between any two unknown symbols of

$$p_{pair} = 2^{-l} \sum_{d=0}^{l} \binom{l}{d} p_e(d) \tag{7}$$

For z symbols, the correct symbol could be decoded to any other of the $z-1$ symbols, which gives an overall symbol error probability p_S of

$$p_S = (z-1) \cdot 2^{-l} \sum_{d=0}^{l} \binom{l}{d} p_e(d) \tag{8}$$

ML vs. Bounded-Distance/Threshold-Based. We demonstrate the efficiency of ML vs. other ECC methods with an equivalent block size (which is a first order estimate of helper data size) and a fixed PUF noise level. Figure 6 shows three scenarios, namely embedding 7 information bits into a block size of ~ 64 (red), 8 bits into a block size of ~ 128 (blue) and 9 bits into a block size of ~ 256 (black).

For each scenario, we compare MLSR($\nu = l$,$\kappa = \alpha$) (solid line) to a Rep(ν,$\kappa = 1$,τ) code (dotted line) and a BCH(ν,κ,τ) code (circles), where ν is the block size, κ is the number of information bits per block, τ is the number of correctable errors. For MLSR, key bit error probabilities are derived from symbol error probabilities according to Eqn. 8, and assuming that for an α-bit symbol, in average half the bits in a symbol are flipped if symbol recovery fails. The solid dots represent simulation results that demonstrate that the analytic approach is accurate. To achieve similar rates for the

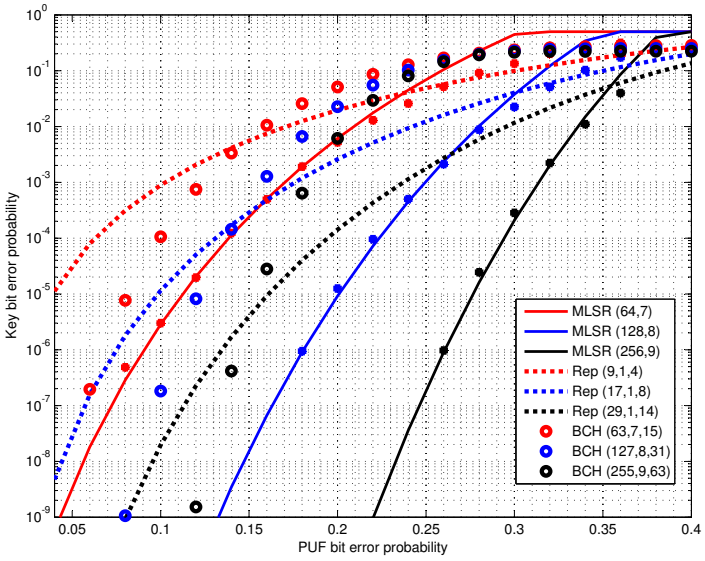

Fig. 6. Key regeneration bit error probabilities (y-axis) over different PUF noise level (x-axis) for MLSR, Repetition and BCH codes

Repetition code for a fair comparison, we divide a block into one sub-block for each information bit. Note that bounded-minimum-distance decoded BCH code with maximum error correction (which corrects $\sim 25\%$ bit flips in this setting) is equivalent to a threshold-based pattern matching approach [19] with comparison threshold set to 0.25.

Comparing curves of same color shows significant performance increase through using ML. For example, for MLSR(128,8) (solid blue curve), at a PUF noise factor of p = 0.15 (x-axis), MLSR has a bit error probability of 1×10^{-8} (y-axis) *prior to* any soft-decision decoding to mop up residual errors, while both repetition and BCH (set to maximum correction of 25%) have error probabilities at least $10,000\times$ higher.

C. Empirical: Reliable Key Generation at 28nm

Design Under Test. We implemented our design in 28*n*m Xilinx Artix-7 FPGA silicon, using *k*=64-sum PUFs [24] in a n=4-XOR construction (in an ASIC implementation this can be replaced by 64-stage Arbiter PUFs). Each symbol is configured as α = 8 bits. To derive a 128-bit key, we treat 22 symbols as keying material, producing a 176-bit value that is down-mixable to a 128-bit key (we assume a 1.3 pre-hash expansion as was used in [1]). Our design contained ability to perform single-parity-symbol soft-decision decoding, where we appended a single 8-bit parity column to the 22 8-bit columns (symbols); this is equivalent to β = 23 symbols stored as helper data for a 128-bit key. Each symbol is mapped to a l=128-bit 4-XOR'ed response sequence, taking up $l \cdot \beta$ = 2944 bits in the helper data. In terms of raw 4-XOR PUF noise, compared to a reference at 25°C ambient, error rates are 22% at -65°C ambient, 24% at 105°C ambient / 125°C junction, and 15% at 25°C ambient.

TABLE I
ENVIRONMENTAL STRESS TESTING, 28*nm* FPGA SILICON

Prov	Regen	Key Regen Cycles	After MLSR	After SD
25°C	-65°C	3.4×10^5	11 Failures (3.24×10^{-5})	0 Failures ($<<$ 2.94ppm)
25°C	105°C	1.2×10^6	3562 Failures (2.95×10^{-3})	0 Failures ($<$ 0.83ppm)
1.00V	1.05V	1.7×10^6	0 Failures	0 Failures ($<<$ 0.59ppm)
1.00V	0.95V	8.0×10^5	0 Failures	0 Failures ($<<$ 1.25ppm)

Key Regeneration Under Environmental Stress. Table 1 shows results of key generation stress testing from one device. Another two devices underwent similar tests (covering same extreme conditions) but ran for a smaller number of key regeneration cycles to confirm these results. Results show a 128-bit key regeneration failure rate $\leq 1 \times 10^{-6}$ in that more than a million key regenerations were run but no failures were seen after soft-decision decode (SD) for the most stressful condition of 105°C ambient / 125°C junction.

V. LINEAR-XOR AND LINEAR-SHIFT ATTACKS

Recently, there have been several papers attacking PUF key generators by manipulating helper data.

1) Code-offset: XOR Malleability: [13] took advantage of linear-manipulative properties of code-offset helper data to launch an attack, since XOR'ing helper data with another codeword causes a deterministic differential codeword shift.

2) Pattern Matching: Shift Malleability: [5] crafted an attack noting that a left/right shift in pattern matching helper data produces an "adjacent" helper data in a predictable fashion.

The attacks in [13] [5] manipulated helper data based on a codeword structure or shift pattern that is *common to all devices*. We have resistance to such attacks since our symbol-to-symbol helper data relationships are determined by manufacturing variation *unique to each device* vs. relying solely on algorithmic attributes common to all devices. It is not obvious how to manipulate a helper data block to cause it to converge predictably to a different symbol on a particular device.

VI. CONCLUSION

We presented a ML symbol coding method that can outperform, with orders of magnitude lower bit error probabilities, prior PUF key generation approaches such as BCH, repetition coding, and pattern matching. We further concatenated symbol recovery with soft-decision ECC to eliminate residual errors.

Future work include reconciling our mathematical, simulation, and empirical silicon results with theoretical symbol-based extensions [3] to the fuzzy extractor that are under development.

ACKNOWLEDGMENT

This work was partly funded by the Bavaria California Technology Center (BaCaTeC), grant number 2014-1/9.

REFERENCES

[1] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, P. Tuyls, "Efficient Helper Data Key Extractor on FPGAs," Workshop on Cryptographic Hardware and Embedded Systems (CHES), 2008, LNCS vol. 5154, pp. 181-197.

[2] X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, A. Smith, "Secure Remote Authentication Using Biometric Data," Eurocrypt 2005, LNCS vol. 3494, pp. 147-163.

[3] R. Canetti, B. Fuller, O. Paneth, L. Reyzin, A. Smith, "Key Derivation From Noisy Sources With More Errors Than Entropy," IACR Cryptology ePrint Archive, 2014/243.

[4] J. Delvaux, I. Verbauwhede, "Side Channel Modeling Attacks on 65*n*m Arbiter PUFs Exploiting CMOS Device Noise," IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2013.

[5] J. Delvaux, I. Verbauwhede, "Attacking PUF-Based Pattern Matching Key Generators via Helper Data Manipulation," The Cryptographers' Track at the RSA Conference (CT-RSA), 2014.

[6] Y. Dodis, R. Ostrovsky, L. Reyzin, A. Smith, "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data," 2008 ed (first published in Eurocrypt 2004, LNCS vol. 3027, 2004, pp. 523-540).

[7] Y. Dodis, B. Kanukurthi, J. Katz, L. Reyzin, A. Smith, "Robust Fuzzy Extractors and Authenticated Key Agreement from Close Secrets," IACR Cryptology ePrint Archive, 2010/456.

[8] B. Gassend, D. Clarke, M. van Dijk, S. Devadas, "Silicon Physical Random Functions," ACM Conference on Computer Communication Security (CCS), 2002.

[9] B. Gassend, "Physical Random Functions," MS Thesis, MIT, 2003.

[10] M. Hiller, D. Merli, F. Stumpf, G. Sigl, "Complementary IBS: Application Specific Error Correction for PUFs," IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2012.

[11] M. Hiller, M. Weiner, L. Rodrigues Lima, M. Birkner, G. Sigl, "Breaking through Fixed PUF Block Limitations with Differential Sequence Coding and Convolutional Codes," International Workshop on Trustworthy Embedded Devices (TrustED), 2013.

[12] A. Juels, M. Wattenberg, "A Fuzzy Commitment Scheme," ACM Conference on Computer and Communications Security (CCS), 1999.

[13] K. Karakoyunlu, B. Sunar, "Differential Template Attacks on PUF Enabled Cryptographic Devices," IEEE International Workshop on Information Forensics and Security (WIFS), 2010.

[14] V. van der Leest, B. Preneel, E. van der Sluis, "Soft Decision Error Correction for Compact Memory-Based PUFs Using a Single Enrollment," Workshop on Cryptographic Hardware and Embedded Systems (CHES) 2012, LNCS vol. 7428, pp. 268-282.

[15] D. Lim, "Extracting Secret Keys from Integrated Circuits," MS Thesis, MIT, 2004.

[16] R. Maes, P. Tuyls, I. Verbauwhede, "A Soft Decision Helper Data Algorithm for SRAM PUFs," IEEE International Symposium on Information Theory (ISIT), 2009.

[17] R. Maes, A. van Herrewege, I. Verbauwhede, "PUFKY: A Fully Functional PUF-based Cryptographic Key Generator," Workshop on Cryptographic Hardware and Embedded Systems (CHES), 2012, LNCS vol. 7428, pp. 302-319.

[18] S. Mathew, S. Satpathy, M. Anders, H. Kaul, A. Agrawal, S. Hsu, G. Chen, R. Parker, R. Krishnamurthy, V. De, "A 0.19pJ/bit PVT Variation-Tolerant Hybrid Physically Unclonable Function Circuit for 100% Stable Secure Key Generation in 22nm CMOS," IEEE International Solid-State Circuits Conference (ISSCC), 2014.

[19] Z. Paral, S. Devadas, "Reliable and Efficient PUF-based Key Generation Using Pattern Matching," IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2011.

[20] J. Proakis, "Digital Communications," 3rd Ed. 1995.

[21] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, J. Schmidhuber, "Modeling Attacks on Physical Unclonable Functions," ACM Conference on Computer and Communication Security (CCS), 2010.

[22] G. Suh, "AEGIS: A Single-Chip Secure Processor," PhD thesis, MIT, 2005.

[23] M. Yu, S. Devadas, "Secure and Robust Error Correction for Physical Unclonable Functions," IEEE Design and Test of Computers, vol. 27, no. 1, pp. 48-65, Jan./Feb. 2010.

[24] M. Yu, D. M'Raihi, R. Sowell, S. Devadas, "Lightweight and Secure PUF Key Storage Using Limits of Machine Learning," Workshop on Cryptographic Hardware and Embedded Systems (CHES), 2011, LNCS vol. 6917,pp. 358-373.