Reliable and Efficient PUF-Based Key Generation Using Pattern Matching

Zdenek (Sid) Paral*, and Srinivas Devadas†
*Verayo, Inc., †Massachusetts Institute of Technology
*sparal@verayo.com, †devadas@mit.edu

***Abstract*—We describe a novel and efficient method to reliably provision and re-generate a finite and exact sequence of bits, for use with cryptographic applications, e.g., as a key, by employing one or more challengeable Physical Unclonable Function (PUF) circuit elements. Our method reverses the conventional paradigm of using public challenges to generate secret PUF responses; it exposes response patterns and keeps secret the particular challenges that generate response patterns.**

The key is assembled from a series of small (initially chosen or random), secret integers, each being an index into a string of bits produced by the PUF circuit(s); a PUF unique pattern at each respective index is then persistently stored between provisioning and all subsequent key re-generations. To obtain the secret integers again, a newly repeated PUF output string is searched for highest-probability matches with the stored patterns. This means that complex error correction logic such as BCH decoders are not required. The method reveals only relatively short PUF output data in public store, thwarting opportunities for modeling attacks.

We provide experimental results using data obtained from PUF ASICs, which show that keys can be efficiently and reliably generated using our scheme under extreme environmental variation.

I. Introduction

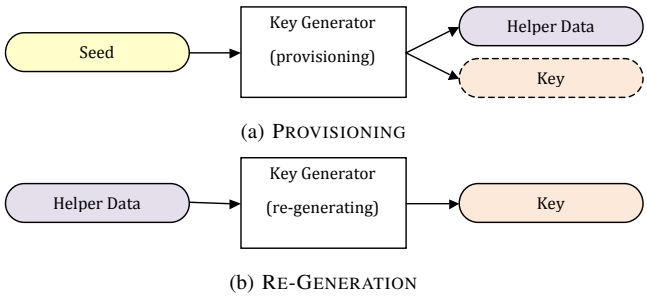
An important aspect of improving the level of trustworthiness of semiconductor devices, semiconductor based systems, and the semiconductor supply chain is enhancing physical security. Not only do we want semiconductor devices to be resistant to computational attacks, but also to physical attacks. Physical Unclonable Functions (PUFs) [1] [2] are becoming a useful tool in this regard.

Silicon PUFs generate signatures based on device manufacturing variations which are difficult to control or reproduce. Given a fixed challenge as input, a PUF outputs a response that is unique to the manufacturing instance of the PUF circuit. These responses are similar, but not necessarily bit exact, when regenerated on a given device using a given challenge, and are expected to deviate more in Hamming distance from a reference response as environmental parameters (for example, temperature and voltage) deviate between provisioning and re-generation. This is because circuit parameters, such as delays, do not vary uniformly with temperature and voltage.

There are two broad classes of applications for PUFs [2] [3] [4] [5] [6]. In certain classes of authentication applications, the silicon device is authenticated if the regenerated response is "close enough" in Hamming distance to the provisioned response. Errors in PUF responses are forgiven up to a certain threshold. In these authentication applications, challenges are never repeated to prevent replay attacks. However, the PUF has to be resistant to software model building attacks (e.g., learning attacks described in [7] [8]) in order to be secure. Else, an adversary can create a software model or clone of a particular PUF. A second class of applications is secret key generation. In conventional usage of a PUF as a key generator, only a fixed number of secret bits need to be generated from the PUF. These bits can be used as symmetric key bits or used as a random seed to generate a public/private key pair in a secure processor [9]. However, in order for the PUF outputs to be usable in cryptographic applications, the noisy bits need to be error corrected, with the aid of helper bits, commonly referred to as a syndrome. The greater the environmental variation a PUF is subject to, the greater the possible difference (noise) between a provisioned PUF response and a re-generated response.

This conventional method of PUF key generation using PUF response bits as secret keys has been explored in many publications including [2] [10] [11] [12] [13]. Error correction has to be secure, robust and efficient. The security concern is the leakage of secret bits through the syndrome or helper bits. Dodis et al give a general framework for secure error correction for biometric data in [14]. An Index-Based Syndrome (IBS) coding scheme that is information-theoretically secure is described and implemented in [13]. Robustness requires that the number of corrected errors be equal to greater than the maximum number of bit-errors from the widest range of environmental variation expected. All of these proposed schemes require fairly heavyweight error correction logic, e.g., a BCH decoder that is capable of correcting several bit-errors in a 64-bit codeword.

We propose a very different method of secret key generation using PUFs in this paper. Rather than using a fixed (and possibly) public challenge, while keeping the response bits secret, we reverse the paradigm and keep secret the particular challenges that generate exposed response bits. The secret key can be chosen at random. Roughly, our method works as follows: A PUF beginning from a fixed public challenge generates a string of response bits of length L. Choose a secret integer I of bit-size $\log_2(L)$ and treat it as an index into the string L. Beginning with this index, expose a W-length pattern ($W < L$) of PUF outputs and store it in *public* non-volatile storage. This is the provisioning step. During key re-generation, the PUF begins internally generating its output string, and comparison logic compares the PUF output to the stored pattern(s). If an approximate match with a number



(a) PROVISIONING

(b) RE-GENERATION

Fig. 1. Key Provisioning and Re-Generation

of mismatches equal to or less than T is found, then the associated index for the match is I with some probability. To generate a K-bit secret, we can run the above scheme $\frac{K}{\log_2 L}$ times.

We note that the "error correction" in the above scheme only requires comparison logic (similar to a PUF authentication scheme) and is very efficient from a hardware standpoint. The parameters L, W and T have to be chosen so the probability of a collision (i.e., a different index being returned) and the probability of no match (all patterns in the re-generated PUF output have more than T mismatches) are negligible under prescribed environmental variation. The security of the scheme is based on the assumption that it is hard to construct a model of PUF behavior given a limited number of challenge-response relationships.

We describe our scheme and its various parameters in Section II. Security considerations are the subject of Section III. Related work is discussed in Section IV. Experimental results using PUF chips are presented in Section V, and we conclude the paper in Section VI.

II. PATTERN MATCHING KEY GENERATOR

A. Basics of Key Generation

Figure 1 describes the basics of key generation.

At provisioning time, the key generator takes an externally provided (secret) Seed (entropy for the generated Key) and, using its embedded PUF, encodes this Seed into (public) Helper Data and a (secret) Key (Figure 1a). The Seed is only input once and may be discarded; the Helper Data is (publicly) stored for later use during Key re-generation; the (secret) Key is discarded.

A key generator must reliably produce an earlier provisioned Key, given the corresponding Helper Data. A PUF-based key generator combines the Helper Data with its unique, unclonable hardware function, so that only the presence of both the hardware circuit and the Helper Data leads to the correct Key, while the Helper Data alone does not reveal any usable information about the Key.

B. Architecture

The architecture of the Pattern Matching Key Generator (PMKG) is shown in Figure 2. Besides control logic, the PMKG consists of the following components:

- **Re-startable, Bi-modal Challenge Sequence Generator**: This is usually a linear-feedback shift register (LFSR) with an associated primitive polynomial. The sequence generator has a single input that affects the generated sequence.
- **One or more Challengeable Physical Unclonable Functions**: We use an arbiter PUF originally introduced in [15].
- **PUF Output Blender**: For security against modeling attacks, we require multiple Arbiter PUF outputs to be blended into a single bit. 4 bits are XOR'ed together corresponding to a 4-XOR Arbiter PUF [3].
- **Pattern Shift Register** of length W.
- **Tolerant Pattern Match Detector** fires if the pattern in the Pattern Shift Register is within the threshold T of the selected pattern in the Persistent Pattern Store.
- **Persistent Pattern Store and Pattern Selector**: Patterns for each round are stored in the Pattern Store during provisioning.
- **Additional Bi-modal Key Mixer(s)**: The key can be obtained directly from the index of the challenge sequence generator or after mixing.
- **Volatile Key Store** is an SRAM.

The various parameters in the PMKG are summarized in Table I.

Metric	Symbol	Example	Note
Key size	K	128	
PUF count	P	1	
Blender ratio	B	4	inp-bits/out-bit
Pattern width	W	256	
Round length	L	1024	
Round count	R	16	$R \geq \frac{K}{N}$
Match threshold	T	80	Tolerance
Secret index size	N	10	$N = \log_2(L)$
Clocks per round	CPR	5,120	$CPR = \frac{(L+W)\times B}{P}$
Clocks total	CT	81,920	$CT = R \times CPR$
Entropy size	E	160	$K \leq E = R \times N$
Total pat. size	S	4,096	$S = W \times R$

TABLE I
ARCHITECTURAL PARAMETERS IN THE PATTERN MATCHING KEY GENERATOR

The key generator works in rounds. A *round* is an instance of generating O bits ($O = L + W$) of continuous, blended PUF data; there are L possible patterns of width W found in such data. The position of a pattern is represented by its (zero-based index) I, which is N bits wide for binary power round lengths ($L = 2^N$).

During provisioning, for each round, a secret index is selected. Blended PUF output bits of length W beginning from the appropriate index are loaded into non-volatile memory. Multiple bits are blended by the PUF Blender, for example, four PUF output bits (from a single or multiple PUFs) may be XOR'ed together to generate a blended PUF output bit. This blending is crucial to security and is discussed in Section III.

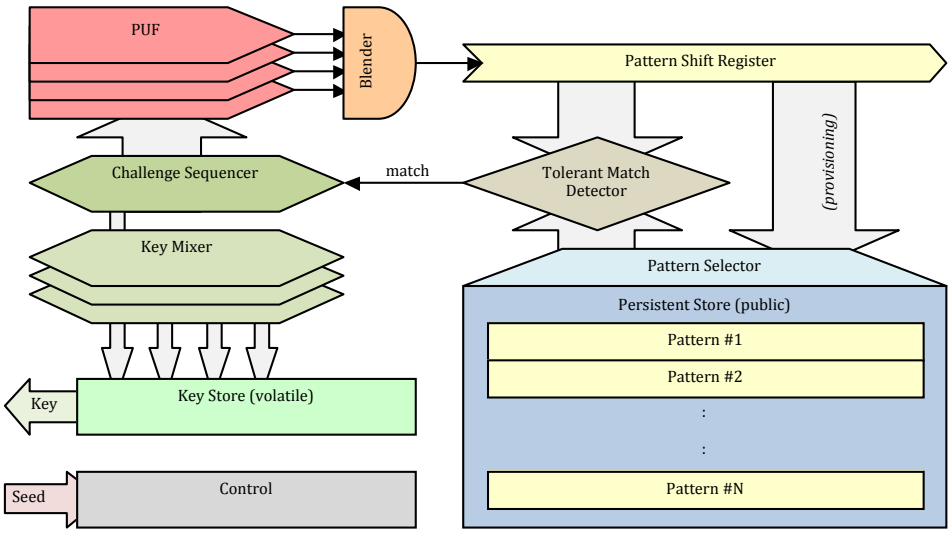


Fig. 2. Key Generator Architecture. This generator can be used to provision and re-generate a key.

Assume that the PMKG has been provisioned. During key re-generation, the PMKG works in multiple rounds, each consisting of a fixed-length challenge sequence. The challenge sequence generator is a linear feedback shift register (LFSR) with an associated primitive polynomial, and begins from a fixed challenge. PUFs generate response bits based on the applied challenge. The blended outputs are shifted into a pattern shift register and the Tolerant Match Detector matches the first pattern against the contents of the pattern shift register. If the number of mismatches is $\leq T$, the match signal is raised. At the end of the round, the index of the challenge that caused the match is loaded into the Volatile Key Store. If there is no match in a round, we have a failure, see Section II-D. We note that the PMKG takes exactly the same number of cycles and performs exactly the same number of operations each round to generate any key. Thus, it is less susceptible to differential power or timing analysis [16].

The match signal triggers a change in the challenge sequencer schedule as described in the next section.

C. Bi-Modality

The **match** signal in Figure 2 is raised when the PUF has generated a matching pattern. It is also used to "fork" the challenge sequence. This has several advantages:

1) Security is enhanced (cf. Section III). Since the index that is matched on is secret, each round makes the actual challenge sequence less and less traceable to an outsider/attacker, at a multiplicative rate of L per round.
2) It is consistent with running the challenge sequencer for a fixed number of cycles each round.
3) Forking in the challenge sequencer is set up in such a way that at the end of the round, the matching secret index can be deterministically derived from the LFSR contents.

We now elaborate on the forking functionality. Define $CS(c,a,f)$ as the challenge sequencing function with the starting challenge c, number of advancements a, and sequence-forking flag f. The forking flag f is cleared at the beginning of every round, and set upon finding a match between the round's pattern data and the current blended PUF data. The challenge sequencing is therefore split into two parts, one "before match" and "at and after match". Note that the "before match" part may be of zero length. If no match were found (a fault condition), the resulting challenge value would be composed as $c_{r+1}(no\_match) = CS(c_r, L, 0)$, for the sequence that started with the challenge c_r, advanced L times, with the forking flag cleared during the whole round. Under non-faulty conditions, a pattern match is made at some index I_r, setting the forking flag f for the rest of the round. The resulting challenge can be composed from the concatenated sequencing operations, $c_{r+1} = CS(c_{rm}, L - I_r, 1)$, where $c_{rm} = CS(c_r, I_r, 0)$.

Alternatively, the challenge sequence could be split into three parts, one "before match", one "at match", and one "after match", whereby the flag is only set in the single-advancement "at match" phase.

D. Failures and Reliability

The PUF output data are not fully repeatable, which is usually exaggerated by the blending function (e.g., XOR), and there is no guarantee that this key generator can always converge to the same key, despite and/or because of the forgiving nature of the noise-tolerant pattern matching logic. We have two possible failure conditions: pattern misses and pattern collisions.

a) Pattern miss: A miss occurs if the PUF generated data contains so much noise that it differs too much from the pattern block and the match detector does not fire at all during a round, which is detectable by the control logic at the end of each round. Misses are false negatives. Frequent misses indicate that the threshold T is set too low and should be increased.

b) Pattern collision: A collision occurs if the PUF generated data happens to come too close to matching a pattern block originated by a different secret index within the round. Collisions are false positives. A collision results in an incorrect recapture of the secret index and subsequent catastrophic divergence from the provisioned challenge schedule in case of the bi-modal challenge sequence generator. Unlike the pattern miss, it is undetectable at the control level. If collisions occur, it means that the threshold T is set too high.

The best defense against the above failures lies in choosing a sufficiently wide pattern (W), so that the probabilities of misses and collisions decrease to miniscule levels with appropriate choice of T. This is the approach we take in Section V.

In implementations where wide patterns can be traded for time, partial (miss) and full (collision) retrials with error detection can be employed. For example, a one way (hash) function slaved to the challenge sequencer produces a digest that is compared with a hash value stored at provisioning time; a match indicates a high probability of correct key generation. A narrow pattern retrial approach needs additional logical support at provisioning time, as the index choices must be discriminated for stability, and rejected if found unable to perform within an acceptable number of re-tries.

III. Security Considerations

A. Modeling Attacks

We are exposing response data of the PUF. In authentication applications, it is assumed that even given unlimited challenge-response pairs (CRPs), the adversary is unable to create the model of the underlying PUF or successfully predict the response for a hithherto unseen challenge. A circuit for which it is impossible to create a software model is called a Strong PUF. Recent work [8] has determined that several architectures that were previously considered Strong PUFs can be cloned via machine learning attacks.[1] One architecture that is currently resistant to machine learning attacks is a k-XOR n-stage Arbiter PUF with $k > 6$ and $n = 64$. The number of CRPs required to successfully attack k-XOR PUFs grows rapidly with k. For a 4-XOR Arbiter PUF with error-inflicted CRPs on a randomly-generated PUF structure (i.e., not a hardware PUF), over 30,000 CRPs are required, and for a 5-XOR 128-stage PUF structure over 100,000 CRPs are required. (Note that we cannot arbitrarily increase k in PUF designs, since the noise levels increase with k.)

In a PMKG, CRPs are not exposed directly, but the adversary knows all the details of the PMKG architecture including the beginning fixed challenge. The number of exposed response bits is the total pattern size, which can be assumed to be 4096 (cf. Table I). This is much smaller than the number of CRPs required for modeling, moreover, there is ambiguity in what challenges resulted in particular responses. We have two means of increasing the complexity seen by the adversary.

For small additional area, the number of PUFs P can be increased and the effective response size that is exposed *per* PUF can thus be reduced by a factor of P.

[1]This is similar to how cryptographic primitives are attacked and sometimes broken.

The second way is to not expose the challenge sequence schedule to the adversary. As described in Section II-C, the occurrence of a match at a particular index affects the challenge schedule in the subsequent round. Since the matching index is secret, constructing possible CRPs becomes more and more difficult with each passing round. This increases the ambiguity in the CRPs available to the adversary.

B. Provisioning Mode Security

We assume that the chip is provisioned with a secret and then the provisioning mode is disabled, for example, through an irreversible "fuse" operation. This is often assumed when PUFs are used to generate a fixed-length response that is used as a key (e.g., ring oscillator bits [3] or SRAM bits [17]).

The same strategy can be employed with the PMKG as well. Note that the PMKG is both an encoder as well as a decoder and is used for provisioning as well as re-generation. In this case, an arbitrary Seed is input to the PMKG and we use a forkable schedule to further enhance security. The number of response bits exposed is $R \times W$ and, further, it is difficult for the adversary to compute challenge-response pairs. Forking during provisioning works the same as in re-generation regarding when the "forking flag" is set/cleared (either from the point of match to the end of round, or for one clock at match); only the condition setting it changes. During provisioning the comparisons are made on a running index counter versus the secret index of the current round; during re-generation, the comparisons are made on the response string versus the syndrome pattern for the current round. The challenge schedule must be identical during both provisioning and re-generations, including the "dead runs" from the match points to the ends of each round to thwart timing attacks.

IV. Related Work

A. Physical Unclonable Functions (PUFs)

Pappu [1] described Physical One-Way Functions implemented using microstructures and coherent radiation and described an authentication application requiring external measurement equipment. Gassend et al [2] coined the term Physical Unclonable Function, presented the first silicon PUF with integrated measurement circuitry, and showed how PUFs could be used for authentication as well as cryptographic applications. Many other silicon realizations of PUFs have been proposed (e.g., [17], [5], [4], [6]). For a more comprehensive review of PUF literature, see [18].

It has been shown that some proposed PUFs can be modeled or reverse-engineered (e.g., [7], [19]) precluding their use in unlimited authentication applications. State-of-the-art in numerical modeling attacks on PUFs (e.g., [8]) was discussed in Section III.

B. Error Correction

In a typical error correction setting for PUF, during an initialization phase, the PUF is evaluated for a set of challenges. Then a syndrome is computed based on the responses. The syndrome or helper data is public information which is

later sent to the PUF along with the challenges to perform correction on response bits. Equivalently, the syndrome can be stored locally on chip. Perhaps the earliest work that pointed out the requirement of error correction for cryptographic keys and used it on PUF responses was [10]. The work employed 2D Hamming codes for error correction. Later, a more realistic use of Bose-Chaudhuri-Hochquenghen (BCH) codes for error correction on PUF responses was proposed [9], [20]; however, the implementation cost and hardware overhead of this code is very high.

The use of repetition codes along with conventional syndrome generation using XOR masking was proposed for PUFs in [11]. Another error reduction technique proposed in [21] uses a longest increasing subsequence algorithm (LISA) to group ring oscillators on FPGAs such that resulting responses from comparing the frequency of ring oscillators within the same group does not change with temperature. [22] demonstrates the use of majority voting on delay-based PUF responses implemented on an FPGA to achieve noise reduction. The work in [4] applies the helper data algorithm described in [14], [11] to butterfly PUF responses, and further demonstrates classification of challenges into groups with different level of robustness. The responses to challenges in groups with higher level of robustness are less likely to be affected by noise and environmental variations. Assuming 0.78 bits of entropy for each PUF output bit, the algorithm requires 1500 PUF bits to derive a 128-bit uniformly distributed random key with a failure rate of 10^{-6}.

[12] describes the use of conventional soft-decision decoders. While a hard-decision decoder operates on typically 0 or 1 in a binary code, the inputs to a soft-decision decoder may take on a whole range of values in-between. Because of this extra information which indicates the reliability of each input data point, soft-decision coding yields a higher coding gain than their hard-decision counterparts.

Since the syndrome is public information, the adversary can derive bias information from the syndrome to tighten the search space to find the secret key. Information leakage via syndrome is a critical aspect of the error correction and has been addressed in [13]. This work further introduced index-based syndrome coding (IBS). IBS leaks less information that conventional syndrome coding methods such as code-offset construction using linear codes [14] or other variants that use bitwise XOR masking [11]. IBS generates pointers to values in a PUF output sequence so that the syndrome is not a direct linear function of PUF output bits and parity bits. In addition when IBS is used with PUFs with real-value outputs, it has a coding gain associated with soft-decision coding. IBS reduces the error correcting code implementation complexity by about $16\times$ to $64\times$. Error correction performed on responses acquired from ring oscillator PUFs implemented on Virtex-5 FPGAs under temperature variation from -55^oC to 125^oC at 1.0V $\pm$ 10% show an error rate below 1 ppm.

All of the above works corresponded to using PUF *response* bits as secret key bits; here we use *indices* into sets of PUF challenges as the secret key bits, and the PUF response is exposed. Our security assumption is that we are using a PUF with the properties defined in [2] (and what has later been termed a Strong PUF [23] in the literature). If necessary, we can weaken this assumption to limit the adversary to only knowing a relatively small set of PUF response bits (cf. Section III).

V. Evaluation Using ASIC Data

We evaluate our PMKG on data obtained from [24] that includes 4-XOR and higher Arbiter PUFs. We focus on 4-XOR Arbiters in our experiments.

We first provide results on inter-chip and intra-chip variation of ten 4-XOR Arbiter chips in Figure 3. The PUF chips were provisioned at 25^{o}C and response re-generation was done between -25^{o}C and +85^{o}C in an TestEquity HalfCUBE (Model 105A) Oven with switching between -25^{o}C, +25^{o}C, and +85^{o}C. We note that the PUF is receiving power from an RFID reader and therefore there is voltage variation across provisioning and re-generation, but it cannot be quantified precisely. Figure 3a shows that the inter-chip variation is very close to 50%, and the average intra-chip variation is 5%. Figure 3b gives the false positive and false negative rates for various thresholds. If we choose a threshold of 80, the false positive and negative rates are both less than 1 part per billion (the point where the curves intersect is below 0.001 ppm).

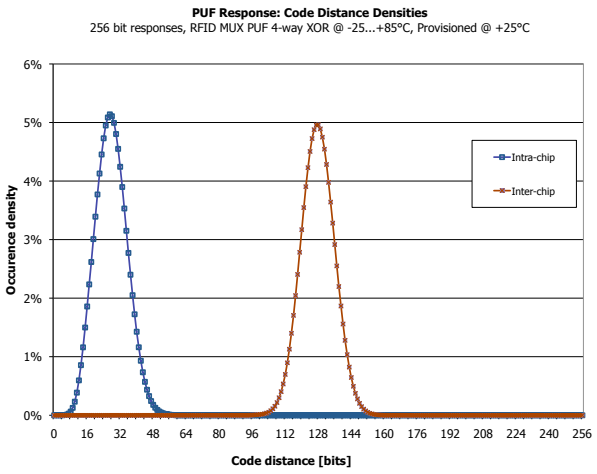
We next provide results on key provisioning and re-generation. Five 4-XOR Arbiter chips were provisioned at 25^{o}C and response re-generation was done between -25^{o}C and +85^{o}C with switching between -25^{o}C, +25^{o}C, and +85^{o}C. We used four settings for W and T as shown in Table II. For each of the four settings of W and T, keys were re-generated over 18,500 times across the temperature range.

Table II shows the number of trials required for successful key re-generation for the different settings. If key re-generation fails, either because of a pattern miss (false negative) or a pattern collision (false positive) in any of the rounds, we retry to a maximum of 20 trials. For example, $W = 96,\ T = 24$ resulted in only 14,003 out of 18,540 successful key re-generations in the first trial, and in 94 cases, 19 trails were not enough. On the other hand, $W = 256,\ T = 80$ was successful in the very first trial 100% of the time. This is consistent with Figure 3b.

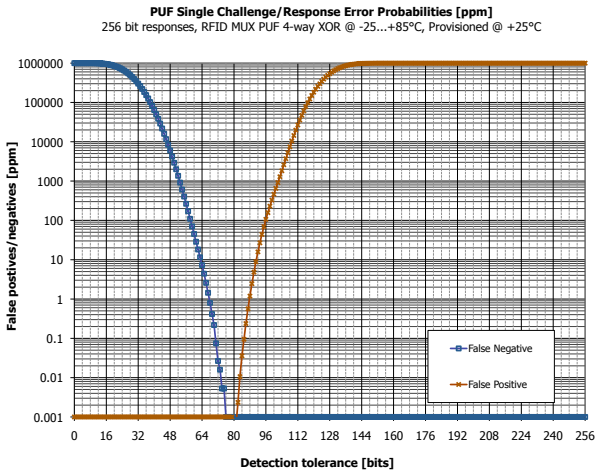
Our results indicate that we require $W \geq 128$, and the specific choice will depend on the trading off key generation time (including possible retries) for total pattern size.

VI. Conclusions

We have presented a viable method of PUF-based key generation that is notable for low clock latency and hardware requirements: only a PUF, registers, bit-comparison, and threshold computation logic are required. The generation of keys can be made faster and the security level raised by increasing the number of PUFs. In order for the exposed responses to not be a security hazard we had to use a 4-XOR arbiter PUF. Future work will focus on the development of new delay PUF structures that are hard to model and have less intrinsic noise than a 4-XOR arbiter PUF.



(a) VARIATION



(b) FALSE POSITIVES AND NEGATIVES

Fig. 3. (a) Intra-chip and Inter-chip variation for 4-XOR Arbiter PUFs. (b) False Positive and False Negative Rates.

Settings	**Total**	**Trials**																
W, T	**Keygen**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16-19	20
96, 24	18535	14003	2500	865	355	231	126	85	66	36	38	37	23	17	13	7	39	94
128, 30	18535	18347	140	22	11	7	3	1	1	2	1							
192, 54	18537	18534	3															
256, 80	18540	18540																

TABLE II
KEY RE-GENERATION RESULTS. KEYS WERE PROVISIONED AT 25^oC AND RE-GENERATED BETWEEN -25^oC AND $+85^o$C.

REFERENCES

[1] R. Pappu, "Physical one-way functions," Ph.D. dissertation, Massachusetts Institute of Technology, 2001.

[2] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *Computer and Communication Security Conference*, 2002.

[3] G. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Design Automation Conference (DAC)*, 2007, pp. 9–14.

[4] S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "Extended abstract: The butterfly PUF protecting IP on every FPGA," in *Hardware-Oriented Security and Trust (HOST)*, 2008, pp. 67–70.

[5] D. Holcomb, W. Burleson, and K. Fu, "Initial SRAM state as a fingerprint and source of true random numbers for RFID tags," in *Proceedings of the Conference on RFID Security*, 2007.

[6] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight secure PUF," in *International Conference on Computer Aided Design (ICCAD)*, 2008, pp. 670–673.

[7] D. Lim, "Extracting Secret Keys from Integrated Circuits," Master's thesis, Massachusetts Institute of Technology, may 2004.

[8] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *ACM Conference on Computer and Communications Security (CCS)*, 2010, pp. 237–249.

[9] G. E. Suh, "AEGIS: A Single-Chip Secure Processor," Ph.D. dissertation, Massachusetts Institute of Technology, Aug 2005.

[10] B. Gassend, "Physical Random Functions," Master's thesis, Massachusetts Institute of Technology, Jan. 2003.

[11] C. Bösch, J. Guajardo, A. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2008, pp. 181–197.

[12] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2009, pp. 332–347.

[13] M.-D. M. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Design and Test of Computers*, vol. 27, pp. 48–65, 2010.

[14] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: how to generate strong keys from biometrics and other noisy data," in *Advances in Cryptology - Eurocrypt*, 2004.

[15] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Delay-Based Circuit Authentication and Applications," in *Symposium on Applied Computing (SAC)*, 2003.

[16] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Lecture Notes in Computer Science*, vol. 1666, pp. 388–397, 1999. [Online]. Available: citeseer.nj.nec.com/kocher99differential.html

[17] Y. Su, J. Holleman, and B. Otis, "A 1.6pJ/bit 96 (percent) stable chip ID generating circuit using process variations," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2007, pp. 200–201.

[18] A. Sadeghi and D. Naccache, Eds., *Towards Hardware-Intrinsic Security: Foundations and Practice*. Springer, 2010.

[19] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Testing techniques for hardware security," in *International Test Conference (ITC)*, 2008, pp. 1–10.

[20] G. Suh, C. O'Donnell, and S. Devadas, "AEGIS: a Single-Chip secure processor," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 570–580, 2007.

[21] C. Yin and G. Qu, "LISA: maximizing RO PUF's secret extraction," in *Hardware-Oriented Security and Trust (HOST)*, 2010, pp. 100–105.

[22] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA PUF using programmable delay lines," in *IEEE Workshop on Information Forensics and Security*, 2010, p. in press.

[23] J. Guajardo, S. Kumar, G. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2007, pp. 63–80.

[24] S. Devadas, E. Suh, S. Paral, R. Sowell, T. Ziola, and V. Khandelwal, "Design and Implementation of PUF-Based "Unclonable" RFID ICs for Anti-Counterfeiting and Security Applications," in *Proceedings of 2008 IEEE International Conference on RFID (RFID 2008)*, May 2008, pp. 58–64.