

CHALMERS
UNIVERSITY OF TECHNOLOGY

Emulation and Design of Polar Codes for PUF-based Key Generation

Downloaded from: https://research.chalmers.se, 2026-11-06 02:35 UTC

Citation for the original published paper (version of record):
Wang, X., Larsson-Edefors, P. (2026). Emulation and Design of Polar Codes for PUF-based Key Generation. IEEE Workshop on Signal Processing Systems, SiPS

N.B. When citing this work, cite the original published paper.

© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, or reuse of any copyrighted component of this work in other works.

(article starts on next page)

Emulation and Design of Polar Codes for PUF-based Key Generation

Xu Wang and Per Larsson-Edefors
Chalmers University of Technology, Gothenburg, Sweden
xuwang@chalmers.se

***Abstract*—Polar codes have gained increasing attention for physical unclonable function (PUF)-based key generation due to their strong error-correction performance at short block lengths and low hardware complexity when decoded using sequential successive cancellation (SC). However, reliable PUF-based key generation requires low block-error rates (BLERs) down to 10^{-12}. Designing polar codes to satisfy such reliability requirements necessitates extensive evaluations. Additionally the inherently sequential nature of SC decoding leads to prohibitively long simulation runtimes. To address the runtime challenge, we propose a high-speed FPGA-based emulation platform featuring a deeply-pipelined and fully-unrolled FastSSC decoder which achieves a throughput of one codeword per cycle. With a 200-MHz clock, we can complete 10^{10} emulation runs in approximately 50 seconds and 10^{13} runs in about 14 hours. We perform comprehensive design space explorations to identify $\mathcal{P}(2048, 64)$ with 5-bit quantization as a code that satisfies the stringent reliability constraints of PUF-based key generation. The code achieves a BLER below 10^{-12} and 10^{-9} for an input bit-error probability of 0.15 and 0.25, respectively. In addition, we analyze the tradeoff between hardware complexity, the number of SRAM PUF source bits, and the maximum tolerable input bit-error probability under BLER requirements of 10^{-9} and 10^{-12}, providing practical design guidelines for ASIC implementation.**

***Index Terms*—Physical unclonable function, Key generation, Polar code, FPGA emulation, Ultra-low BLER.**

I. Introduction

Polar codes can achieve the symmetric capacity of memoryless channels under successive-cancellation (SC) decoding [1] and they are adopted in the 3GPP 5G NR standard. But beyond communication applications, polar codes have also gained increasing attention in cryptography applications, such as physical unclonable function (PUF)-based key generation [2]–[4]. This is because, first, polar codes have high error-correction performance at relatively short block lengths, which makes it possible to minimize the required number of SRAM PUF source bits, and, second, SC decoding lends itself to sequential decoding, which leads to low hardware complexity.

However, designing polar decoders for PUF-based key generation is challenging due to stringent reliability requirements. In contrast to communication systems, PUF-based applications require ultra-low block error rates (BLERs), often on the order of 10^{-6} to 10^{-9} [5], and even as low as 10^{-12} [6]. Meeting such requirements necessitates a careful selection of code parameters, including block length N and code rate R, as well as hardware constraints such as the number of quantization bits q for log-likelihood ratios (LLRs). This leads to a large design space with interdependent parameters, where performance depends not only on these design parameters but also on the input bit-error probability p, assuming the key generation process is modeled as a binary symmetric channel (BSC) [3], [4], [7].

Consequently, identifying suitable codes requires extensive performance evaluation across multiple dimensions, making exhaustive exploration highly challenging on conventional CPU-based simulation platforms. Achieving a BLER under 10^{-12} requires over 10^{13} emulation runs, motivating the use of FPGA-based emulation to accelerate evaluation. However, existing SC-based platforms are either sequential and too slow, with runtimes in excess of 10 days [8], [9], or have limited scalability, with maximal block lengths of $N = 1024$ [10].

To address these limitations, we propose a high-speed FPGA-based emulation platform which integrates a deeply-pipelined fully-unrolled FastSSC decoder [11] and enables *one-codeword-per-cycle* processing. Synthesized at 200 MHz, the proposed platform achieves a throughput of 2×10^8 codewords per second, completing 10^{10} and 10^{13} emulation runs in 50 seconds and 14 hours, respectively.

The proposed emulator enables us to perform extensive design space exploration under stringent BLER constraints, targeting BLER $< 10^{-12}$ at $p = 0.15$ and BLER $< 10^{-9}$ at $p = 0.25$. To the best of our knowledge, this work is the first to identify polar code configurations $\mathcal{P}(2048, 64)$ with $q \geq 5$ that meet these stringent requirements. Additionally, we also present two tradeoff studies, for BLERs below 10^{-12} and 10^{-9}, between hardware complexity, SRAM PUF source bit count, and the maximum tolerable p, thereby providing ASIC design guidelines for PUF-based key generation systems.

II. Background

Ring oscillators [7] and SRAMs [2], [3] are two common types of PUF source. In this work, we assume a bias-free SRAM as PUF source.

A. Code-Offset Construction

Among key generation approaches for SRAM PUFs, code-offset construction [12] and syndrome construction [3], [12] are the two most commonly used ones [5]. In this work, we adopt code-offset construction, which consists of two phases: enrollment and key reconstruction, as illustrated in Fig. 1.

During enrollment, an initial SRAM response r is exclusive-ORed (XOR) with a randomly selected codeword c from an error-control code (ECC), producing the helper data h.

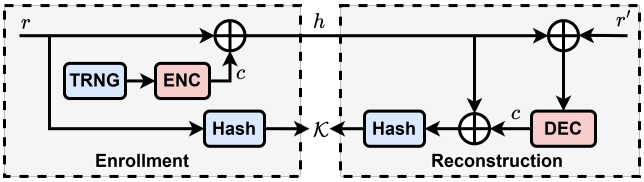
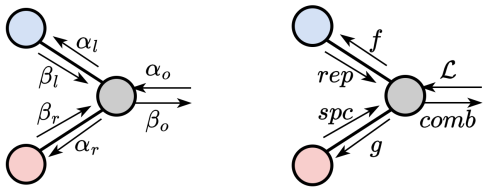


Fig. 1. Code-offset construction based key generation.



(a) (b)

Fig. 2. FastSSC-based polar tree showing message flow (a) and function transition (b) for a $\mathcal{P}(8,4)$ code.

The helper data is stored as public information and used for subsequent key reconstruction, in which a new response r' is observed. Due to noise and environmental variations, r' may differ from the enrolled response r and is modeled as $r' = r \oplus e$, where e denotes the error pattern. Combining r' with the stored helper data gives $c \oplus e$, which can be viewed as a corrupted codeword. Therefore, key reconstruction can be formulated as a channel-decoding problem, where the decoder (DEC) recovers c using the corresponding ECC decoding algorithm. In this work, PUF errors are modeled as a BSC with bit-flip probability p. To extract high-entropy keys, cryptographic hash functions are typically applied; however, the associated security aspects are beyond the scope of this work.

B. Polar Codes

Polar codes, denoted as $\mathcal{P}(N, K)$ with code rate $R = K/N$, are a class of capacity-achieving ECCs. Through channel polarization, a set of N identical channels is transformed into synthesized channels with varying reliabilities. Information bits are assigned to the most K reliable channels, while the remaining channels are fixed as frozen bits. Channel reliability can be estimated using density evolution [13].

In the encoding process, the input information bits are first mapped to their designated positions according to the frozen-bit vector. The resulting vector u is then multiplied by the generator matrix G_N to produce the encoded codeword $c = uG_N$. Here, G_N is constructed recursively using the Kronecker product of the binary kernel $F_2 = \begin{bmatrix} 1 & 0; 1 & 1 \end{bmatrix}$ such that $G_N = F_2^{\otimes n}$ for $N = 2^n$.

Decoding is more complex than encoding. Successive-cancellation (SC) decoding [1] represents a low-complexity decoding scheme, however, due to its bit-by-bit decoding nature, SC decoding incurs high latency. Several variants, e.g., simplified successive-cancellation (SSC) [14] and FastSSC [15], have been proposed to reduce latency by identifying subcode patterns and avoiding traversal of the full decoding tree. This work adopts FastSSC and supports four node types: a RATE-0 node with all frozen bits; a RATE-1 node with all information bits; a repetition (REP) node containing all frozen bits except for a single information bit at the end; and a single parity check (SPC) node containing all information bits except for a single frozen bit at the beginning.

C. Fast Simplified Successive Cancellation (FastSSC)

Figure 2 shows the FastSSC-based decoding tree for a $\mathcal{P}(8,4)$ code, comprising a REP node on the left and a SPC node on the right. Unlike communication systems that assume access to log-likelihood ratio (LLR) values from the channel, PUF-based systems typically provide only binary responses r. Therefore, root LLR values need to be determined by $\alpha_o = \mathcal{L}(r)$ within the decoder, where $\mathcal{L}$ is

$$\mathcal{L}(r[i]) = \log\left(\frac{1-p}{p}\right)(1 - 2r[i]). \tag{1}$$

In a hardware implementation, LLRs can be quantized to q-bit signed integer values in the range $[-2^{q-1}, 2^{q-1}-1]$, reducing both computational complexity and memory usage.

As shown in Figs 2a and 2b, decoding begins with the computation of α_l using the f function. The corresponding β_l is then obtained using the rep function:

$$\beta_v[i] = \begin{cases} 0, & \text{if } \sum_{i=0}^{N_v-1} \alpha_v[i] \geq 0, \\ 1, & \text{otherwise,} \end{cases} \tag{2}$$

where $N_v = 2^s$ denotes the node length and s is the node stage; in this case $s = 2$. Based on β_l, α_r is calculated using the g function, followed by the determination of β_r using the spc function, where a parity bit is determined by

$$\beta_v[i] = \begin{cases} 0, & \text{when } \alpha_v[i] \geq 0, \\ 1, & \text{otherwise,} \end{cases} \tag{3}$$

$$\text{parity} = \bigoplus_{i=0}^{N_v-1} \beta_v[i]. \tag{4}$$

Then, according to the parity bit, β_v can be corrected by

$$j = \arg\min_i |\alpha_v[i]|, \tag{5}$$

$$\beta_v[j] = \beta_v[j] \oplus \text{parity}. \tag{6}$$

Given both β_l and β_r, the decoded message β_o is computed using the combine ($comb$) function, where $\beta_v[i] = \beta_r[i - N_v/2]$ for $i \geq N_v/2$, and $\beta_v[i] = \beta_l[i] \oplus \beta_r[i]$ otherwise.

For a node v at stage s with a $N_v = 2^s$ length, let $a = \alpha_v[i]$, $b = \alpha_v[i + N_v/2]$, and $c = \beta_v[i]$. The f and g functions can be implemented in a hardware-efficient manner according to:

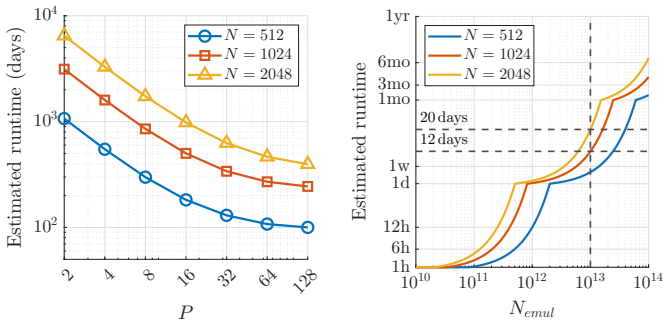
$$f(a,b) = \text{sign}(a)\ \text{sign}(b)\ \min\big(|a|, |b|\big),$$
$$g(a,b,c) = \big(1 - 2c\big)\, a + b.$$

Although RATE-0 and RATE-1 nodes are not included in this example, β_v is determined using the $r0$ and $r1$ functions, where $r0$ sets all bits to zero and $r1$ performs hard decisions.

D. Limitations of Existing FPGA-Based Emulation Platforms

Codes for reliable PUF-based key generation require design explorations in ultra-low BLER regimes. In stringent cases, this corresponds to a BLER $< 10^{-9}$ at $p = 0.25$ and a BLER $< 10^{-12}$ at $p = 0.15$, requiring at least 10^{13} emulation runs to observe sufficient error events. This poses a significant challenge for existing sequential SC-based emulation platforms [8], [9], which typically require over 10 days to obtain a single ultra-low-BLER evaluation point. These architectures reuse a limited number (P) of processing elements (PEs) to sequentially process each node, requiring L_{dec} cycles per codeword and limiting decoding throughput. While FastSSC-based decoding can reduce L_{dec}, the achievable speedup strongly depends on the frozen set and corresponding decoding tree. At a clock rate of f in Hz, the total emulation time for N_{emul} runs is given by $N_{emul} L_{dec}/f$ seconds.

To estimate L_{dec} for different FastSSC-based decoders, we evaluate the runtimes using the following assumptions: the f, g and $comb$ functions require $4\lceil N_v/4P \rceil$ cycles, the rep and $r0$ functions take $\lceil N_v/2P \rceil$ cycles, spc requires 2 cycles, while $r1$ is computed simultaneously with the f and g functions. Fig. 3a shows that, for $R = 1/16$ and $P = 4$ decoders, 10^{13} emulation runs require approximately 549 days to complete, even for the shortest $N = 512$ code. For $N = 1024$, the runtime exceeds four years, making exhaustive evaluation impractical.



(a) P_{dec} = 1 and N_{emul} = 10^{13} (b) P_{dec} = 20 & P = 128

Fig. 3. Runtime complexity of sequential SC decoding versus (a) processing-element parallelism (P) and (b) number of emulation runs (N_{emul}) for $p = 0.25$ and $R = 1/16$, assuming $f = 200$ MHz.

To accelerate evaluation, more parallel PEs can be employed. However, as shown in Fig. 3a, emulation improvements run out of steam as P approaches 128, where the runtime still exceeds 100 days. Another approach is to parallelize several (P_{dec}) decoders, reducing emulation time to $N_{emul} L_{dec}/(P_{dec} f)$. Fig. 3b shows the relationship between runtime and N_{emul}, where $P = 128$ and $P_{dec} = 20$. Although the runtime of the $N = 512$ decoder is reduced to below one week, larger block lengths of $N = 1024$ and 2048 still require approximately 12 and 20 days, respectively, to complete 10^{13} evaluation runs. Consequently, exhaustive evaluation remains impractical given the large design space of polar codes with different parameter combinations.

To overcome these throughput limitations, unrolled SC-based decoders have been investigated. An unrolled SC-based decoder is integrated in [10], increasing the emulation throughput to one codeword per cycle and achieving 10^8 codewords per second at $f = 100$ MHz. However, the platform only supports evaluations up to $N = 1024$. This is because the unrolled SC decoder requires $N \log_2(N)$ PEs, resulting in a substantial lookup table (LUT) usage that limits scalability. In addition, the evaluation in [10] is based on an additive white Gaussian noise (AWGN) channel model, which may not accurately represent the error characteristics of SRAM PUF-based key generation systems.

III. FPGA-based Emulation Platform

To address emulator limitations, we develop and integrate a deeply-pipelined fully-unrolled FastSSC decoder in our emulation platform. With an $L_{dec} = 1$, the emulator processes *one codeword per cycle*. In addition, *multiple codewords per cycle* can be processed by parallelizing emulation chains. The platform consists of a software component and an FPGA-based hardware component. The software performs polar-code construction and RTL generation, with the generated designs subsequently moved to the FPGA system for evaluation.

A. Software Component

The software component receives the necessary parameters, including N, K, p, and q for the polar code, as well as N_{emul} to set the emulation count. Based on these parameters, the polar code is constructed using density evolution. The resulting frozen set of size $N-K$ is converted to a FastSSC-based polar decoding tree, which is then used as a blueprint to generate the corresponding RTL modules, registers and interconnects. In addition, the corresponding emulation blocks and the control logic are generated, with the emulation counter set to N_{emul}. Finally, the generated RTL code is fed to Vivado for synthesis and implementation, and used in the FPGAs for emulation.

B. FPGA Component

Our FPGA emulator models variations in PUF-based key generation as a BSC channel, assuming an SRAM as the entropy source. Fig. 4 shows the implementation of the entire processing chain, including random information generation, encoding, channel emulation, decoding, and error counting.

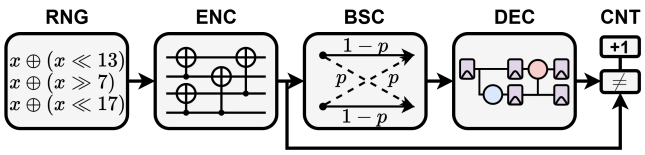


Fig. 4. Block diagram of the FPGA emulator component.

The random number generator (RNG) employs linear-feedback shift registers [16] to produce a K-bit information sequence u. The RNG exhibits a periodicity of $2^{WL}-1$, where WL is the wordlength of the bits generated each cycle. In this design, we choose $WL = 64$, supporting approximately 10^{19} unique generations, which is sufficient to evaluate BLER down to 10^{-12}. The information sequence is then encoded (ENC) according to the generator matrix G_N to get the encoded

codeword c. To reduce latency, the implemented encoder is fully parallelized by leveraging XOR chains.

Noise arising from environmental variations in SRAMs of PUF-based key generation can be modeled as a BSC, in which each bit flip occurs independently with a probability of p. To emulate this process in hardware, N parallel RNG-comparator pairs are employed, generating an N-bit error vector e per cycle. Each RNG produces pseudo-random numbers based on a unique seed, ensuring statistical independence across pairs. The generated number is then compared against a threshold defined as $\lfloor p \cdot (2^{64} - 1) \rceil$, which represents the probability of a bit flip scaled to the numeric range of the RNG. If the generated number falls below the threshold, the corresponding bit in the error vector e is set to 1; otherwise, it is set to 0. Finally, the e vector is XORed to the codeword c, producing the noisy codeword $r' = c \oplus e$.

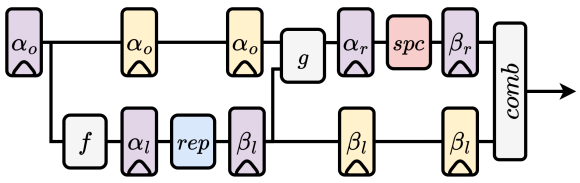


Fig. 5. Deeply-pipelined fully-unrolled FastSSC decoder for a $\mathcal{P}(8, 4)$ code.

The noisy codeword is fed to the decoder (DEC) for error correction. The decoder operates based on the FastSSC-based decoding tree and is implemented in a fully unrolled and deeply pipelined architecture [11], enabling a throughput of one codeword per cycle. Fig. 5 shows a block diagram of our decoder for a $\mathcal{P}(8, 4)$ code. Each function described in Fig. 2b is implemented by a dedicated unit and registers are inserted to reduce logic depth. In addition, separate registers (in yellow) are introduced to propagate and align data between pipeline stages. To synchronize α_o and β_l, four registers are integrated.

Among the four node types used in this design, REP and SPC nodes tend to exhibit the largest logic depth, particularly for nodes v with large lengths N_v, where $N_v = 2^s$. For REP nodes, as shown in eq. 2, the rep function involves summing N_v LLR values, resulting in an s-depth adder tree. SPC nodes, on the other hand, consist of an s-depth comparator tree to determine the minimum value and propagate its index, as shown in eq. 5.

TABLE I. Number of nodes at different code rates (R) for $N = 2048$.

R	Type	$N_v \in$				Total
		$(0, 8]$	$(8, 64]$	$(64, 256]$	$(256, 1024]$	
1/4	SPC	30	8	0	0	229
	REP	28	10	0	0	
1/8	SPC	28	5	0	0	189
	REP	25	6	2	0	
1/16	SPC	20	1	0	0	127
	REP	23	8	2	0	
1/32	SPC	9	0	0	0	85
	REP	8	10	0	1	

Communication systems typically employ code rates $R \geq 1/2$ and large-N_v SPC nodes are prevalent [15]. With the low code rates of PUF-based key generation, however, large-N_v REP nodes dominate the decoding tree. Categorized on node length N_v, Table I shows the number of REP and SPC nodes for $N = 2048$ and $p = 0.15$ codes with different code rates R. It should be noted that no SPC node exceeds a length of 64, and for $R = 1/32$, not a single SPC node exceeds a length of 8. In contrast, REP nodes with large N_v are observed as R decreases. For the lowest code rate ($R = 1/32$), a REP node reaches the maximum length of $N_v = 1024$. This leads to a 10-depth q-bit adder tree, with increasing delay as q grows. To mitigate this issue and maintain a high operating clock rate, pipeline registers are inserted between stages of the adder tree.

Finally, the decoded message is compared with the delayed reference codeword, and the error counter (CNT) is incremented whenever a mismatch occurs. The accumulated error count is then used to estimate the BLER. While this deeply pipelined and fully unrolled architecture enables high throughput, its hardware resources including registers grow quadratically with the block length of the code, with an additional dependence on code rate and frozen-bit locations.

C. Runtime Speedup

Our decoder accepts a new codeword every clock cycle. This reduces the runtime to N_{emul}/f seconds, or $N_{emul}/(fP_{dec})$ when decoder-level parallelism is employed, corresponding to a speedup of L_{dec} compared to conventional sequential decoding. For example, with $f = 200$ MHz, $N_{emul} = 10^{10}$ can be completed in approximately 50 seconds, while $N_{emul} = 10^{13}$ requires only around 14 hours. Notably, the runtime is independent of code parameters such as block length N, code rate R, and input bit-error probability p.

IV. Code Design Space Explorations

Our emulator enables efficient design space exploration under the stringent BLER requirements for PUF-based key generation. Specifically, we target BLER $< 10^{-12}$ at $p = 0.15$ and BLER $< 10^{-9}$ at $p = 0.25$, where a 128-bit secret key length is assumed. We explore the design space over parameters including block length $N \in \{512, 1024, 2048\}$, code rate $R \in \{1/16, 1/32\}$, number of quantization bits $q \in \{3, 4, 5, 6\}$, and input bit-error probability p from 0.15 to 0.25. All configurations are synthesized and implemented with a target clock frequency of 200 MHz.

To limit the design space, the code rate R is first set to 1/16, N is chosen from 512, 1024, and 2048, and q ranges from 3 to 6 bits. Fig. 6 shows the relationship between BLER and p for these configurations. As expected, by increasing N, we improve the error-correction performance as larger block lengths lead to stronger polarization. For a fixed code configuration, larger q leads to better BLER performance; however, no significant improvement is observed beyond 5 bits. Under the stringent BLER requirement of 10^{-12}, only five configurations can tolerate $p = 0.15$: $\mathcal{P}(1024, 64)$ codes with $q = 5$ and 6, and $\mathcal{P}(2048, 128)$ codes with $q = 4$, 5 and 6. However, no configuration is able to achieve a target BLER of 10^{-9} for $p = 0.25$.

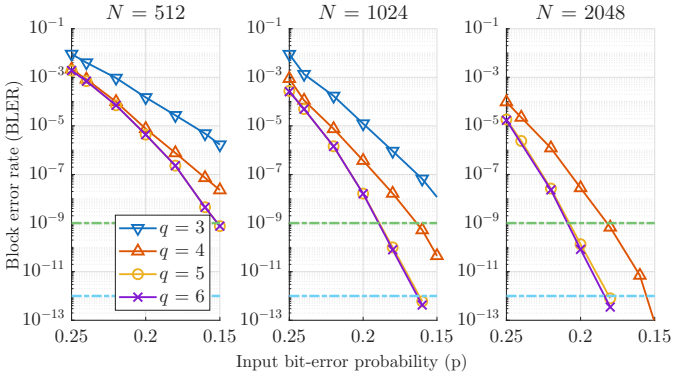


Fig. 6. Error-correction performance vs input bit-error probability for $R = 1/16$ and $N = 512$, 1024, and 2048. The horizontal green and light blue lines indicate the BLER requirement of 10^{-9} and 10^{-12}, respectively.

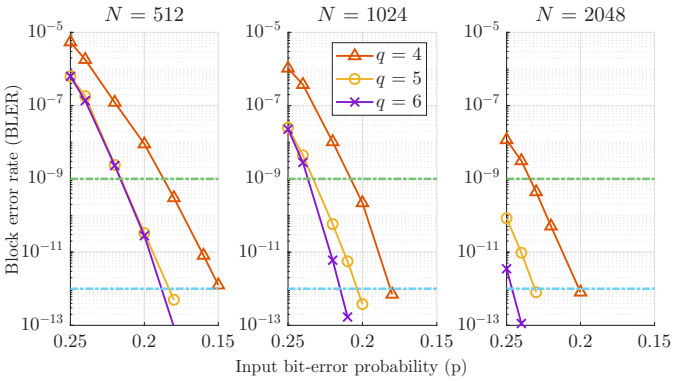


Fig. 7. Error-correction performance vs input bit-error probability for $R = 1/32$ and $N = 512$, 1024, and 2048. The horizontal green and light blue lines indicate the BLER requirement of 10^{-9} and 10^{-12}, respectively.

We can also improve the error-correction performance by reducing the code rate R, however, this comes at the expense of an increased number of PUF source bits. Fig. 7 presents codes with the same block lengths as in Fig. 6, but here R is reduced to 1/32 and $q = 3$ candidates have been removed. As anticipated, reducing the code rate improves the error-correction performance. For example, at $N = 2048$, the improvement exceeds three orders of magnitude for $q = 4$, and five orders of magnitude for $q = 5$ and 6. In addition, the 6-bit configurations begin to diverge from those of the 5-bit configurations as N increases, indicating that higher LLR resolution is more beneficial for larger block lengths and lower code rates. For the stringent BLER requirement of 10^{-12}, all configurations meet the target BLER at $p = 0.15$, except for the 4-bit $\mathcal{P}(512, 16)$ candidate. Notably, $\mathcal{P}(2048, 64)$ with $q = 5$ and 6 fulfill the BLER requirement of 10^{-9} for a stringent $p = 0.25$, resulting in an SRAM PUF source bit count of 4096.

V. Practical Tradeoff Analysis

In addition to error-correction performance, practical polar decoders for PUF-based systems are restricted in terms of hardware complexity and number of PUF source bits. To reduce area, polar decoders are typically implemented in a sequential decoding manner, reusing a limited number of PEs and storing intermediate data in memory. In this case, memory dominates area [2], [4]. For the following analysis, we adopt the memory configuration in [4], which requires $N(2q + 3)$ bits to store LLR values and binary messages. Taking this complexity into account, Fig. 8 shows design tradeoffs between memory size in bits, maximum tolerable bit-error probability p, and required number of PUF source bits, under a BLER requirement of 10^{-12}. Green and red markers represent code rates $R = 1/16$ and $R = 1/32$, requiring 2048 and 4096 PUF source bits, respectively.

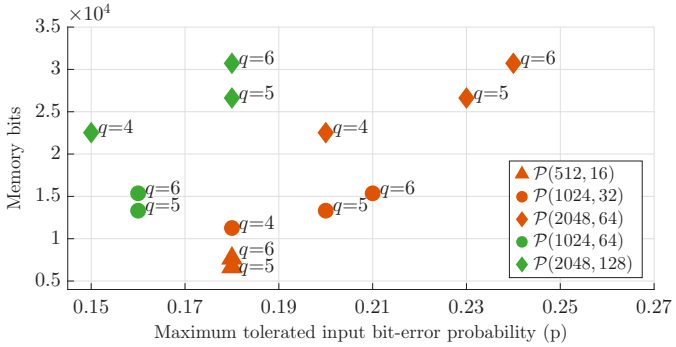


Fig. 8. Hardware complexity (memory bits) of different decoder configurations versus the maximum tolerated input bit-error probability p under the stringent target BLER of 10^{-12}. Green and red markers indicate configurations requiring 2048 and 4096 PUF source bits, respectively.

In general, tolerating higher p requires lower code rate R, larger values of N and q, resulting in an increase in the number of memory bits. This trend is evident for $R = 1/32$ configurations. However, for a low code rate $R = 1/16$, increasing q beyond 5 does not lead to an increasing tolerance for higher p, which means that $q = 5$ provides a favorable performance-complexity tradeoff for these configurations. For a 2048-bit PUF source, a maximum p of 0.18 can be handled by a 5-bit $\mathcal{P}(2048, 128)$ decoder, requiring a memory of 26,624 bits. In contrast, leveraging the same size of memory and quantization bits, $\mathcal{P}(2048, 64)$ can push p to 0.23, at a cost of a doubling of PUF source bits. Among all configurations, the 6-bit $\mathcal{P}(2048, 64)$ code achieves the highest tolerable $p = 0.24$, requiring a memory of 30,720 bits.

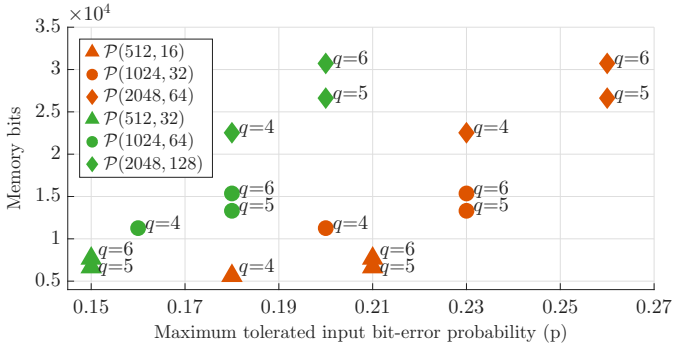


Fig. 9. Hardware complexity (memory bits) of different decoder configurations versus the maximum tolerated input bit-error probability p under the stringent target BLER of 10^{-9}. Green and red markers indicate configurations requiring 2048 and 4096 PUF source bits, respectively.

Figure 9 illustrates the tradeoffs when the BLER requirement is relaxed to 10^{-9}. In this regime, increasing quantization precision beyond $q=5$ provides no further improvement in the maximum tolerable p. Therefore, $q=5$ provides the best performance-complexity tradeoff. For a 2048-bit PUF source, a 5-bit $\mathcal{P}(2048,128)$ decoder achieves the maximum tolerable input bit-error probability with $p=0.20$, requiring a memory size of 26,624 bits. With the same memory size and block length, reducing code rate R to 1/32 increases the maximum tolerable p to 0.26. However, this requires increasing the number of PUF source bits to 4096.

VI. FPGA Resource Usage

While a deeply-pipelined fully-unrolled architecture enables high-speed processing, it requires significant FPGA resources, thereby limiting the range of code configurations that can be implemented within the available FPGA resources. To maintain a high operating clock rate and accelerate emulation, configurations with $N=512$, 1024, and 2048 are implemented and emulated on an XCVU190 FPGA.

TABLE II. Resource usage for an emulation chain based on a 5-bit $\mathcal{P}(2048,64)$ decoder implemented at 200 MHz.

Component	# LUT	# FF	# CARRY8
RNG	72	128	0
ENC	2,069	2,048	0
BSC	216,770	133,120	8,192
DEC	52,741	49,075	769
CTL etc.	22,144	20,357	19
# Total usage	293,796	204,728	8,980
Utilization (%)	27.4	9.5	6.7

Table II presents the resource usage of a single-chain ($P_{dec}=1$) emulator based on a 5-bit $\mathcal{P}(2048,64)$ decoder with $p=0.25$. Clearly, LUT resources are the limiting factor. Among all the components, the BSC unit with its 2048 parallel RNG-comparator pairs is the component that requires the most overall resources. The second-largest block is the decoder itself. These two blocks dominate resource usage and would become the primary constraints as the block length N increases. Using a 200-MHz clock, the emulator sustains a throughput of more than 400 Gbps.

VII. Conclusion

This work presents a high-speed FPGA-based emulation platform for exploration of polar codes targeting PUF-based key generation. By employing a deeply pipelined and fully unrolled FastSSC decoder, the proposed platform processes one codeword per cycle. At 200 MHz, this corresponds to a throughput of 2×10^8 codewords/s and, for $N=2048$, a data rate of over 400 Gbps. This enables efficient BLER evaluations under ultra-strict reliability requirements, completing 10^{10} and 10^{13} emulation runs in 50 seconds and 14 hours, respectively.

Leveraging this platform, we conduct comprehensive design space explorations and identify a 5-bit $\mathcal{P}(2048,64)$ code that satisfies stringent reliability requirements of BLER $<10^{-12}$ at $p=0.15$ and BLER $<10^{-9}$ at $p=0.25$. This results in a requirement of 4096 PUF source bits for generating a 128-bit key, assuming a code-offset construction for the key generation. Furthermore, under BLER targets of 10^{-12} and 10^{-9}, we study design tradeoffs between hardware complexity, PUF source bit counts, and maximum tolerable input bit-error probabilities. The results reveal that, for a BLER requirement of 10^{-9}, increasing quantization precision beyond 5 bits provides no further improvement in the maximum tolerable p. In contrast, the ultra-low 10^{-12} regime still benefits from increasing quantization precision up to 6 bits, albeit at the expense of increased memory size for storing intermediate values. Furthermore, while lower code rates improve reliability, they also increase the required number of PUF source bits.

Acknowledgment

This project was financially supported by the Area of Advance ICT, Chalmers.

References

[1] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, pp. 3051–3073, 2008.

[2] C. Kestel, C. Frisch, M. Pehl, and N. Wehn, "Towards more secure PUF applications: A low-area polar decoder implementation," in *IEEE 35th Int. System-on-Chip Conf. (SOCC)*, 2022.

[3] B. Chen, T. Ignatenko, F. M. J. Willems, R. Maes, E. van der Sluis, and G. N. Selimis, "A robust SRAM-PUF key generation scheme based on polar codes," in *IEEE Global Communications Conf.*, 2017.

[4] X. Wang and P. Larsson-Edefors, "Reliable and area-efficient polar decoders for SRAM PUF-based key generation," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2026.

[5] K. Lata and L. R. Cenkeramaddi, "FPGA-Based PUF designs: A comprehensive review and comparative analysis," *Cryptogr.*, vol. 7, 2023.

[6] N. Bijnens and V. van der Leest, "SRAM PUF: The secure silicon fingerprint." https://www.synopsys.com/articles/sram-puf-secure-silicon-fingerprint.html, July 2024. Accessed: 2026-05-08.

[7] R. Maes, A. V. Herrewege, and I. M. R. Verbauwhede, "PUFKY: A fully functional PUF-based cryptographic key generator," in *Workshop on Cryptographic Hardware and Embedded Systems*, 2012.

[8] J. Tong, H. Zhang, L. Huang, X. Liu, and J. Wang, "An asymmetric adaptive SCL decoder hardware for ultra-low-error-rate polar codes," *16th Int. Symp. on Wireless Communication Systems (ISWCS)*, pp. 532–536, 2019.

[9] C. Xiong, Y. Zhong, C. Zhang, and Z. Yan, "An FPGA emulation platform for polar codes," *IEEE Int. Workshop on Signal Processing Systems (SiPS)*, pp. 148–153, 2016.

[10] J. Wüthrich, A. Balatsoukas-Stimming, and A. P. Burg, "An FPGA-based accelerator for rapid simulation of SC decoding of polar codes," *IEEE Int. Conf. on Electronics, Circuits, and Systems*, pp. 633–636, 2015.

[11] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "237 Gbit/s unrolled hardware polar decoder," *Electronics Letters*, vol. 51, no. 10, pp. 762–763, 2015.

[12] Y. Dodis, R. M. Ostrovsky, L. Reyzin, and A. D. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *IACR Cryptol. ePrint Arch.*, vol. 2003, p. 235, 2004.

[13] R. Mori and T. Tanaka, "Performance and construction of polar codes on symmetric binary-input memoryless channels," in *IEEE Int. Symp. on Information Theory (ISIT)*, pp. 1496–1500, 2009.

[14] A. A. Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Commun. Lett.*, vol. 15, pp. 1378–1380, 2011.

[15] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE J. Sel. Areas Commun.*, vol. 32, pp. 946–957, 2013.

[16] G. Marsaglia, "Xorshift RNGs," *J. Statistical Software*, vol. 8, no. 1, pp. 1–6, 2003.