Masked Memory Primitive for Key Insulated Schemes

Zachary DiMeglio *, Jenna Bustami †, Deniz Gurevin *, Chenglu Jin ‡, Marten van Dijk ‡ and Omer Khan*
*University of Connecticut, Storrs, CT USA
†University of California, Berkeley, CA USA
‡CWI, Amsterdam, The Netherlands

*{zachary.dimeglio, deniz.gurevin, khan}@uconn.edu, †jennabustami@berkeley.edu, ‡{Chenglu.Jin, marten.van.dijk}@cwi.nl

***Abstract*—In practical security systems, it is difficult to keep secret keys protected against adversarial attacks. Key insulated schemes (KIS) are used to improve security by generating session keys that expire after a finite period of time. However, during the refresh period, side channels of the base can be observed, leaking keys during transfer. To counter this, the proposed masked memory primitive prevents these attacks while maintaining low latency and computational requirements. Using PUFs and polar coding, keys are safely stored in memory, allowing users to extract keys as needed while preventing machine learning based attacks against the system. A $(2048, 512)$ polar code construction is proposed for PUF and adversarial error rates of 0.1 and 0.25, respectively, allowing for accurate key reconstructions and sufficient security. Furthermore, a 3.54 - 5.49ms delay between key request and retrieval can be achieved, a $\sim$4.81$\times$ improvement over the state-of-the-art KIS implementation. It is shown that these keys can be reliably requested by a user with $\leq 10^{-6}$ failure probability, while an adversary is unable to obtain the key, even with state-of-the-art decoding techniques and PUF learning algorithms.**

I. Introduction

In computing systems where sensitive data is used, proper security is essential. Typically, this means that information is being protected with the use of a keyed scheme, such as authentication in remote attestation, or encryption for securing data. The security of keys is important, especially at the hardware level. Through the use of side channel analysis (SCA), characteristics such as execution timing and power analysis can reveal significant information about keys as they perform their cryptographic operations. Such SCA attacks are even implemented entirely in software. For example, exploiting permissions on Intel machines allow users to measure CPU power traces without physical access [1], while other attacks write to a machine's cache during encryption, observing when evictions occur [2].

Key insulated schemes (KIS) aim to alleviate this problem by spreading private keys across timed *sessions*, providing security guarantees even in the event of a leaked key [3]. Assuming S total sessions with up $t < S$ session keys that can be leaked from an insecure device, a (t, S) KIS scheme guarantees the security of the remaining $S - t$ session keys [3]. While KIS assumes perfect security of the base that "refreshes" the insecure device with new secret keys [4], a difficult problem is ensuring the security of this base in the presence of a side-channel attacker. Therefore, a method of obfuscating keys before they are stored is required. Of course, this would also require a way to un-obfuscate keys when they are requested by the insecure device.

To perform this obfuscation, one can *mask* the session keys with a random sequence. This involves XORing the key using the mask prior to storage, safely allowing it to reside in memory, then XORing it again upon retrieval to unmask it. This can be performed with a physically unclonable function (PUF), which is a device-unique circuit to generate random outputs (responses) for a given input (challenge), forming a challenge-response pair (CRP) [5], [6]. PUFs come with numerous benefits. First, since PUFs are queried whenever a response is needed, responses do not need to be stored, and the queries can be locked behind access control [7]. Second, PUF designs are impossible to replicate even if their architecture is known, due to their intrinsic randomness introduced at the fabrication level [8], [9]. Finally, PUF readouts are inexpensive to perform on the fly [10]. These properties make the PUF a promising way to mask the session keys of KIS, however, they still have difficulties that must be addressed.

One such difficulty is that the same challenge can produce different responses over time. These reliability errors caused by device aging, temperature changes, or voltage fluctuations cause a legitimate user to incorrectly unmask a session key if used directly [8], [11]. Another difficulty is that PUF side channel attacks, when combined with learning algorithms, can yield accurate PUF models able to predict CRPs [12]–[15]. The goal is to design a system that can update KIS keys on an insecure device without error, while keeping them shielded in the presence of an adversary.

PUF errors can be corrected with the use of a block error correction code (ECC), which encodes k data and $N - k$ parity bits into a codeword of length N to be transmitted across a noisy channel. Schemes such as fuzzy extractors [16] can be used, but may leak PUF response information, contributing towards the adversary learning model. Recently, new high-performance coding schemes have been discovered, particularly polar codes [17]. Polar coding schemes are well suited for masked memory, offering high configurability to

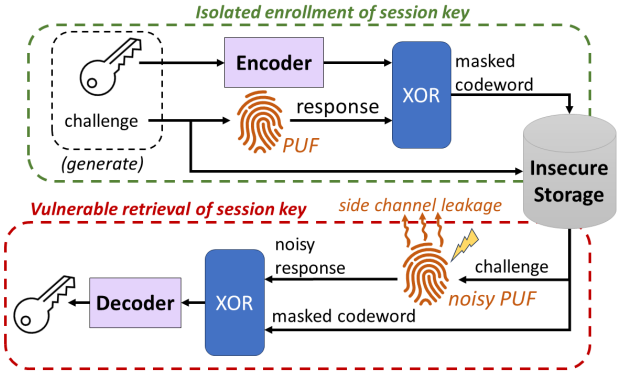

Fig. 1: High-level view of masked memory. The enrollment phase is assumed to be done in a secure environment offline.

explore reliable and secure code constructions for different settings [18], [19], while also having efficient decoding implementations that are suitable for correcting the high error rates of PUFs [20]–[22].

Construction of a *masked memory* is proposed, a primitive that reliably and securely updates KIS keys. First, strings of k random bits are used to generate session keys of length 128-bits. Then, these k bit strings are encoded with the polar transform [17], before being masked with unique PUF-generated responses. The challenges used to generate each response and the corresponding masked codewords are then stored at the insecure base as a tuple. When a session key is requested, the corresponding tuple is retrieved from memory, and the original key can be reproduced by using the challenge to get the masking response. The decoding process allows for errors caused by PUF noise to be corrected. An overview of the system is pictured in Figure 1. In this model, key retrieval occurs while an adversarial observer is present, with some assumptions made about their capabilities. Firstly, the adversary has access to PUF side channels, revealing a number of CRPs after key retrieval [12]. Next, a powerful modeling attack leveraging already known CRPs is used to predict new CRPs. Lastly, for a given coding scheme, the adversary chooses the best-known decoder without latency or resource limitations, and the implementation *does not* have to be the same as the main user. Considering these adversary capabilities, we design a coding scheme leveraging the properties of polar codes to ensure protection against this adversary.

For the main user renewing KIS keys, errors in CRPs are generated from PUF internal noise. These bit error rates can vary ranging from $\sim 5\%$ up to 25% [8], [23]. Denoting this error rate as p_m, we can also approximate the adversarial error rate, p_a, as the error rates produced in a predicted response. Because the adversary only has access to noisy response information, and PUF designs resilient to learning attacks have been shown [24], [25], it is enforced that $p_m < p_a$, as learning all CRPs from a PUF is infeasible for strong PUF designs [8], [26]. However, an advantage of the adversary is that for a given (N, k) coding scheme, no limitation on the decoding algorithm for the chosen parameters is enforced. Therefore, a coding scheme must be designed that can counter the decoder of adversary, while also ensuring the original key can be extracted correctly. Additionally, the minimal coding scheme that can provide reliable and secure key retrieval must be designed, so that an efficient decoding implementation with minimal latency may be achieved. To configure the desired coding scheme, the successive cancellation list (SCL) algorithm is evaluated, a state of the art polar decoding algorithm [22] which introduces an additional parameter, L. Increasing L can improve the error correction performance of a particular coding scheme, but requires additional latency overheads to do so, although some implementations to achieve increased performance have been realized [27].

It is shown that for a $(2048, 512)$ polar code configuration with channel parameters $p_m = 0.1$, $p_a = 0.25$, and list sizes $L \geq 2$, main channel errors are extremely unlikely with an efficiently implemented decoder. Additionally, the modelled adversary using the same $(2048, 512)$ configuration and a decoder with a significantly larger list size ($L = 128$) is unable to leak the k information bits used to form the key in any of the trials. Even when given additional capabilities, such as an oracle that can verify the correctness of any arbitrary source word estimate, the adversary is still unsuccessful.

The contributions of this paper are the following:

1) A masked memory primitive for KIS is constructed using PUFs and polar codes, with our designed code construction yielding reliable key retrieval in an untrusted setting.
2) The same coding parameters are shown to be secure against an adversary model which has access to PUF side channel information and a powerful machine learning algorithm.

To realize the masked memory primitive, simulation is performed to emulate the main user and adversary in an insecure setting. SCL decoding is performed with an NVidia GPU, showing the masked memory primitive with a 3.54 - 5.49ms total latency cost on the KIS critical path, a $\sim 4.81\times$ speedup over the best-known KIS retrieval technique [28].

II. Background

A. Key Insulated Schemes

Key Insulated Schemes have been proposed to provide hard security guarantees while assuming that keys may be leaked over time [3]. In a typical keyed scheme, when the private key is leaked, the system cannot be recovered from the attacker. The entire cryptosystem must be purged, generating a new private key, and correspondingly forcing users to replace their public key. However, in KIS, rather than having a single key, S discrete keys are generated and assigned to a particular time period, called a session [3]. The device that uses these private keys for typical cryptographic operations is assumed to be insecure, but it is assumed that the key generator device is secure in *strong* KIS [4]. Even in strong KIS, however, the assumption is made that the key storage device is insecure and that renewing keys can potentially expose them.

The action of keys being refreshed opens up a new attack vector. Side channel information as the main device retrieves

a new session key may be used by an adversary to leak the key [28]. Our focus is on the need to store and retrieve keys from a vulnerable location securely, despite the presence of an adversary. If this is accomplished, the number of keys leaked across KIS sessions can be reduced.

B. PUFs

A particularly useful primitive to help accomplish secure key storage and retrieval is the PUF. PUFs leverage the uniqueness in chips during the fabrication process to generate unique "digital fingerprints" that can be used for inexpensive random sequence generation [5].

PUFs are well studied in the field of hardware security, and can generally be classified into either i) weak PUFs (wPUF) or ii) strong PUFs (sPUFs). The primary distinction we consider is the difference in challenge space; wPUFs support a limited number of challenges, while sPUFs have a very large quantity of CRPs. Additionally, by definition, the entire challenge-response space cannot be learned in sub-exponential time, making them more robust for secure applications [8], [9], [26]. Thus, the masked memory primitive requires an sPUF.

PUFs offer a way to mask session keys prior to enrollment, and subsequently unmask keys upon retrieval with low computation complexity and high entropy for different CRPs. However, one issue is that PUFs are still vulnerable to attacks, as their side-channel information is available as a byproduct of their operation [12]. Additionally, as physical analog devices, their outputs are not always consistent, and inherent noise can cause fluctuations in the response bits for the same challenge. This can be caused by temperature changes, voltage, device aging, and depends on the implementation used [8].

The noise from a PUF response can be modelled as a simple communication channel called the *binary symmetric channel* (BSC). Given the random variables X and Y, where $X \to Y$ across a noisy channel with alphabets $\mathcal{X}, \mathcal{Y} \in \{0, 1\}$, then in a BSC: $\Pr(X = Y) = 1 - p$ and $\Pr(X \neq Y) = p$, where p is the bit-flip probability. The channel is symmetric, since $\Pr(Y = 0|X = 1) = \Pr(Y = 1|X = 0)$. In the PUF, the bit flip probability p across successive CRP queries is nonzero and must be addressed, as every CRP is used at least twice (once during key enrollment, and once during key extraction). This is classically achieved by implementing error correction codes [8]. By encoding the key prior to PUF masking, the key can be reconstructed at a later point, without noise in the PUF response affecting reliability.

The work in [23] is used to provide an estimated average case PUF error probability. While PUF designs to achieve higher reliability exist, such as the ring-oscillator design proposed in [29], having a low, correctable error may be beneficial from a security standpoint. In fact, having some response bit variation can obfuscate CRP relationships to an adversary, decreasing the ability of a modelling attack to completely learn the behavior of the PUF [30]. By using an estimated PUF error rate of $p_m = 10\%$, an accurate PUF error approximation can be made that also works to inhibit attacks. However, depending on the specific PUF model being used, the value of p_m can be varied.

C. Error Correction Codes

Different ECCs operate on the same principle: insert redundancy to bits with an encoding algorithm, and then undo those operations with a decoding algorithm, so data can be faithfully reconstructed even when corrupted by noise. PUF response errors can be seen a noisy communication medium, so ECCs are leveraged to correct them [8]. The choice of ECC must resist attacks, while also providing reliable behavior to the main user. Flexibility in the code choice is desirable so that parameters can be tuned to balance between error correction, security, and latency.

One method of extracting keys in a noisy environment is to use a fuzzy extractor, which is explicitly designed to extract keys from noisy data (for example, biometric data) [16]. Differential sequence coding (DSC) is another approach often used in conjunction with other schemes to achieve high-performance decoding for noisy PUFs with low bias [31]. However, it is shown in some cases, such as when PUF bias is present, these schemes may lead to information leakage which can then be used to enhance the ML model [19], [32]. Additionally, as PUF reliability, machine learning accuracy, and application requirements can vary, a coding scheme that allows one to choose parameters more freely is desirable, allowing for designs more tailored to the system requirements.

High-performance coding schemes have emerged, such as turbo codes, Reed-Solomon (RM) codes, and polar codes, all of which are linear block error correction methods [33]–[35]. Polar codes are of particular interest since their encoding and decoding algorithms are efficient to implement, and when properly constructed, they outperform other coding schemes at similar rates [17], [36]. Polar codes have a clear advantage in their error correction ability, low decoder complexity, and variety of coding parameters. Several polar code decoding implementations have been proposed, such as belief propagation [37] and successive cancellation decoding [17]. However, focus is turned toward the best-performing decoder available: successive cancellation list (SCL) decoders [22], which extend the original decoder of Arikan [17] with efficient implementations designed [27].

Polar codes have also been used for reliable PUF usage, for example with SRAM-based PUFs, which have high error rates of 15-30% [20]. A security analysis of polar codes is also performed, as it is assumed that PUF information is leaked during a KIS key refresh to train a powerful adversarial ML model. The work in [19] also investigates polar codes to improve security with PUFs, however its focus is on reducing leakage due to PUF biasing.

D. Threat Model

Machine learning models have found success in predicting unknown PUF responses. Some of these models can be highly accurate, such as the attacks proposed in [14], in which some attacks can predict CRPs with $> 99\%$ given sufficient

training data. However, there are also drawbacks to these methods. Also concluded by [14], some PUF architectures render these attacks unusable, such as an 8-XOR length-512 arbiter PUF, which cannot be learned by machine learning models. Additionally, PUF architectures such as the multi-PUF design described in [24] are designed to be especially resilient to machine learning attacks, reducing their prediction accuracy $< 80\%$. Based on the results shown in [25], if the number of PUF CRPs is large, and there are many response bits, there is increased resilience against machine learning-based PUF attacks. Because the masked memory design uses an sPUF, there is a large CRP space that an adversary is unable to exhaustively learn. Additionally, since the PUF response acts as a mask for the codeword, there must be a significant number of response bits. So, based on these findings, the adversarial model is then estimated to have a predicted response error probability $p_a = 0.25$ with respect to the original PUF response from the isolated key enrollment phase.

It is noted that no work has been able to successfully incorporate the employed coding scheme along with the PUF to build an enhanced ML model that somehow considers both parity information along with the PUF CRPs. So, when considering the adversary, the machine learning model is completely separate from the error correction model [38].

After the adversary has predicted a PUF response, in order to extract the original key, the unmasked codeword must be decoded properly. Because masked memory uses polar codes, an adversary who is attempting to decode a masked codeword is constrained to the use of polar decoding algorithms. It is noted, however, that i) the adversary knows the polar code block length and information indices, meaning a proper decoder can be implemented and used by the adversary, ii) the decoder used by the adversary is not necessarily the same as the one used by the main channel, and iii) for a given secret key, the adversary is not present during its enrollment, as that can be done in an offline or isolated environment. More specifically, the implication of ii) is that a different, more powerful decoding algorithm may be implemented on the leaked masked key, so long as the code block length and parity bits are the same. Additionally, the implication of iii) is that the adversary cannot directly access the PUF and obtain the response for a leaked challenge, and must rely on the machine learning model to provide the estimated response.

III. Masked Memory Primitive

The goal of masked memory is to improve the security of KIS keys stored in an insecure location when they are retrieved by a user. Difficulties include reliable error correction, protection from adversaries, and reasonable latency for implementation of the system. Whenever a user needs a key refreshed, they are able to access the masked memory and request the session i key as such: the tuple $\{c, X\}_i$ which contains the PUF challenge and masked key for session i, respectfully, is extracted from the insecure memory. This process is shown in Figure 2.

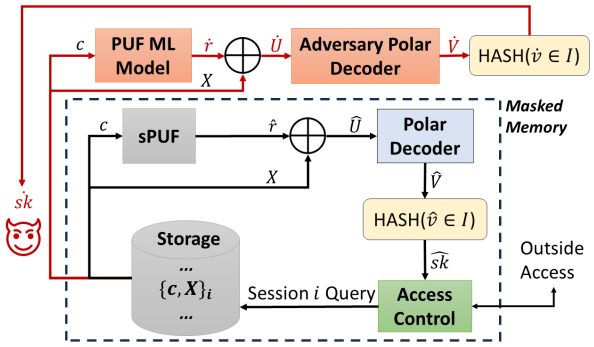

Fig. 2: Process of a key being requested from the masked memory in the presence of an adversarial channel.

For the user, $\hat{r}$ is extracted from the PUF using c, where $\hat{r}$ is the extracted noisy response that was used to mask the key, with each response bit having an error probability of p_m. Then, the noisy codeword is produced via $\hat{U} = \hat{r} \oplus X$. The extracted key $\hat{sk}$ is estimated after the recovered data bits at the output of the decoder, $\hat{v} \in \mathcal{I}$, are hashed down to the original key length. We require that $\hat{sk} = sk$, the original enrolled key.

At the same time, an adversary observes the requested c and X through the insecure memory side channels. The goal of the adversary is the same as the main channel; to attempt the reconstruction of sk using the acquired tuple. Although the adversary does not have direct access to the PUF model, it is assumed that the PUF machine learning model has been trained using previously observed challenge-response pairs $\{c_j, r_j\}_{j=1}^{i-1}$ to form a response estimate $\{c_i, \dot{r}_i\}$ as shown in Figure 3. The adversary estimated response $\dot{r}$ produced by the ML model is used to unmask X, providing the adversary with $\dot{U}$ to be decoded. After the adversary decodes, the recovered data bits, $\dot{v} \in \mathcal{I}$ are hashed into $\dot{sk}$, the adversary key estimate. It is crucial to note here that the adversarial decoder is not necessarily the same implementation as the main channel, however, the block length and data channels will be the same. The adversary is assumed to have the best

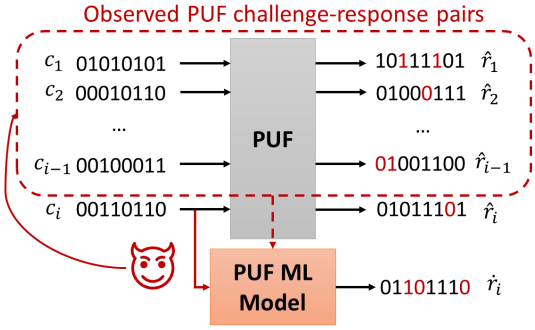

Fig. 3: An adversary observes challenge-response pairs as keys are requested from masked memory.

decoding implementation for a (N, k) polar code, along with the construction of the source code known. The SCL decoder is used by the adversary, which has an additional parameter, L, to determine the most probable transmitted source word V out L different estimates at each stage in the algorithm [22]. When

choosing the coding scheme, it should be ensured that for a choice of N and k, the adversary cannot properly decode even as L grows significantly. Parameters that inhibit the adversary decoder sufficiently are required, while also allowing the main user to decode properly.

The tradeoff between reliability and latency can also be analyzed for the main channel parameters depending on the application. For example, in encryption applications where the reliability of decoding is essential, it is critical that reliability in our scheme is as robust as possible for error correction, at the cost of increased latency. For applications such as attestation, an implementation may forego some reliability for faster decoding if re-transmission would not be as costly. Such a latency analysis is not performed for the adversary, as latency is not seen as a factor for an adversary with ample resources.

The channel and error models used are discussed in Section III-A. Then, the use of polar codes in masked memory is detailed in Section III-B. Finally, the masked memory implementation is discussed, along with the metrics used to analyze performance in Section III-C.

A. Channel and Error Models

Whenever a session key is initially stored in memory, a challenge is used to invoke the PUF black box to receive the response, modeled generally as $r = f(c)$, where r is a response, f is the PUF model, and c is the challenge [8]. It is assumed that the PUF can produce enough bits in r to completely mask the codeword of length N. For subsequent requests, when a session key is requested, internal noise from the PUF affects the reliability of responses, resulting in $\hat{r} = f'(c)$, where $f' \approx f$ is the same PUF with added noise. A response error is then defined as $r_i \neq \hat{r}_i$, where i is the response bit index, r_i is the mask bit, and $\hat{r}_i$ is the unmask bit. A uniform discrete error distribution is assumed across all indices $i \in N$ where $\Pr(r_i \neq \hat{r}_i) = p_m$. Furthermore, as the error on each index is independent of other indices, the BSC is used to model the PUF noise process, giving $\Pr(r_i = \hat{r}_i) = 1 - p_m$. f' acts as the noisy BSC channel for the legitimate user, causing incorrectly unmasked keys if not addressed. Therefore, there is a stringent requirement that decoded secret keys cannot contain any errors, otherwise, key retrieval fails.

The adversary is unable to directly access the PUF f, but can make a series of challenge-response observations to develop a machine learning model which is also modelled as a BSC. The machine learning model is capable of estimating responses to arbitrary challenges c as $\dot{r} = g(c)$, where $\dot{r}$ is the adversary response estimate, g is the ML PUF model, and c is the same challenge used by the legitimate user. It should be noted that based on the wiretap channel model [39], the accuracy of the machine learning model is bounded by f', in that $\Pr(r_i \neq \dot{r}_i) = p_a$ and $\Pr(r_i = \dot{r}_i) = 1 - p_a$ with $p_m < p_a \leq \frac{1}{2}$.

A channel measure that is considered is the *channel capacity*, determined by the PUF error rate. The channel capacity defines a theoretical limit to the ratio of information to total data that may be transmitted across a noisy channel without error [40].

Using the binary entropy function, given as:

$$H(p) = -p\log_2(p) - (1-p)\log_2(1-p) \tag{1}$$

with p as the channel bit-flip probability, the channel capacity of a BSC is:

$$C(x) = 1 - H(x) \tag{2}$$

which gives the capacities for both the main and adversarial channels.

Another measure is the *secrecy capacity* of the system, which depends on the reliability of the two channels. This provides an upper bound on the amount of information that can be transmitted *both* reliably and securely across a communications channel in the presence of a wiretap channel, so long the degradation of the adversary channel with respect to the main channel is known, as in our case [39]. Using Equation 2, the secrecy capacity of the PUF channel is given as [18]:

$$C_s = C(p_m) - C(p_a) \tag{3}$$

This measure gives the theoretical maximum code rate that can be used for reliable communication, while also being secure [39]. In Section IV-A, C_s is used as an upper bound for choosing parameters to configure the masked memory.

B. Construction of Masked Memory with Polar Codes

Polar codes are used to construct the proposed masked memory as the method of error correction since they can be efficiently implemented, have high error correction performance, and have secure constructions [17], [18], [36], [36]. The basic polar code concepts are introduced in the following subsections.

1) Polar Code Construction: Polar codes encode a k bit message into a vector $U \in \{0,1\}$ of length N. The remaining $N-k$ bits provide parity for error correction, giving the code rate as $R = \frac{k}{N}$ [35]. Higher rates lead to better error correction in noisy environments, as additional bits dedicated to error correction are used. In any of the coding considerations, $R < C_s$ is chosen, as choosing parameters exceeding the secrecy capacity holds no guarantees on leakage in the presence of a wiretap channel [41]. Before encoding, k information bits and the $N-k$ parity bits are constructed into a source word V. We denote the k information bit indices as belonging to $\mathcal{I}$, with the $N-k$ parity bit indices (also called the frozen set as they are set to '0') as $\mathcal{F}$ [17].

The members of $\mathcal{I}$ are chosen using the Bhattacharyya parameter [35], density evolution [42], or channel independent construction [43], each having their own variation in performance and implementation complexity. The goal of each construction method is to order the reliability of each channel from most to least reliable, assign the first k of these channels to $\mathcal{I}$, with the $N-k$ remaining indices assigned to $\mathcal{F}$. The capacity of channels in $\mathcal{I}$ approach 1, while the capacity of channels in $\mathcal{F}$ approach 0 when $N \to \infty$ with $R \to C(p)$ for a BSC [17]. Different constructions have tradeoffs between

computational complexity and reliability accuracy. Because of the low construction complexity compared to their performance, we choose the original Bhattacharyya parameter method originally introduced by Arikan [17].

2) Polar Code Encoding: After constructing the source word, V, by interleaving the k information and $N-k$ parity bits, the codeword U is generated as:

$$U = V\mathbf{G}^{\oplus \log_2(N)} \tag{4}$$

Where $\mathbf{G}$ is the polar code kernel given by

$$\mathbf{G} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{5}$$

and $\log_2(N)$ is the Kronecker power of the kernel [35].

This encoding is used to form the masked codeword as $X = U \oplus r$. Later on, when requested by a user, X can be unmasked and subsequently decoded to allow the retrieval of the original key.

3) Polar Code Decoding: After a session key is requested by the user, X is unmasked with the noisy PUF response, forming $\hat{U} = X \oplus \hat{r}$. The original key is then estimated using a decoding algorithm. Two widely used decoding algorithms used for decoding are i) successive-cancellation (SC) decoders [35] and ii) belief propagation (BP) decoders [37]. Both types of decoders have $\mathcal{O}(N \log N)$ decoding complexities. The primary difference is that BP decoding is an iterative algorithm, which can be performed in parallel effectively, at the cost of decoding performance. On the other hand, SC decoders are more serialized, but yield better error correction.

SC decoding recursively attempts to estimate the i^{th} channel of the source word V, v_i, based on the following likelyhood estimate [35]:

$$\hat{v}_i = \begin{cases} 0, & v_i \in \mathcal{F} \\ 0, & \frac{\Pr(\hat{U}, \hat{V}_1^{i-1} | v_i = 0)}{\Pr(\hat{U}, \hat{V}_1^{i-1} | v_i = 1)} > 1 \\ 1, & \text{otherwise} \end{cases} \tag{6}$$

Likelihood ratios for each bit are computed, and a hard decision is made about the estimated source word bit $\hat{v}_i$ at any given stage. Finally, each estimated bit $\hat{v} \in \mathcal{I}$ is extracted, and hashed down to form the 128-bit key estimate $\hat{sk}$. Note that because of the recursive structure of this decoder, it can be difficult to exploit parallelism at the different stages [27].

The adversary goes through the same process as the main channel to attempt the reconstruction of the key, except instead of unmasking X with a PUF, the machine learning model with increased error is used, resulting in a more difficult decoding process. $\dot{U} = X \oplus \dot{r}$ is formed, followed by decoding $\dot{U} \to \dot{V}$, and finally hashing down the bits $\dot{v} \in \mathcal{I}$ into $\dot{sk}$.

Incorporating SC *list decoding* has been shown to improve the performance over the bare SC decoder [22]. The L most likely source words at each step are selected to proceed. At the end of the algorithm, the L best estimates for the transmitted source word remain, with the highest probability word chosen as the output [22]. Equation 7 shows how the final estimate is selected:

$$\hat{V} = \underset{\hat{V}_l}{\arg\max}\{\{\Pr(\hat{U}|\hat{V}_l)\}_{l=1}^{L}\} \tag{7}$$

with $\hat{V}_l$ as the l^{th} generated possible source word.

The caveat here is that as an $\mathcal{O}(LN \log N)$ algorithm, additional latency is experienced for better decoding performance [27]. While this tradeoff is carefully considered for the main channel, we assume that an adversary can choose an arbitrarily large L since they are not bounded by latency constraints.

Also, as L increases, the error correction performance approaches an upper bound [22]. Consider the most likely transmitted source word to usually be among the first L_{max} estimates. Then, increasing $L > L_{\text{max}}$ would have no effect on decoding performance. So long as the adversary is unable to decode properly past this bound, we will have a similar amount of list decoding failures for $L > L_{\text{max}}$ as $L = L_{\text{max}}$. Literature suggests this bound to be around $L_{\text{max}} = 32$, however, it is also found that sometimes the transmitted source word can exist somewhere in the list, despite it not being the *most* likely as provided by Equation 7 [22]. We consider an adversary that has an *oracle* $\Omega(\cdot)$ that can do the following:

$$\Omega(\dot{V}) = \begin{cases} 0, & V \neq \dot{V} \\ 1, & V = \dot{V} \end{cases} \tag{8}$$

That is, the adversary can determine if an arbitrary vector is the original source word. Using $\Omega(\cdot)$, it is assumed the adversary will choose to greedily verify each L source word estimate produced by the SCL decoder, even if it is not the estimate ultimately returned. This could realistically be done by first extracting the information bits and performing the hash to form $\dot{sk}$, as is normally done. Then, depending on the application, $\dot{sk}$ could be verified. For example, in the case of authentication, a public key could be tested for compatibility with $\dot{sk}$. In the case of encryption, a ciphertext could be decrypted with sk, and if correct, the plaintext should be interpretable.

In either case, although Equation 8 provides information about the correctness of a particular source word estimate, no other information (i.e., Hamming distance) is revealed. This idea is used in the subsequent sections.

C. Implementation of Polar Codes based Masked Memory

For the masked memory coding scheme, if a 128-bit key is considered, then the constraint $k > 128$ is needed to ensure that the key can be hashed down from the information bits. Similarly, schemes with other key sizes (i.e., 256 or 512-bit key based schemes) can also be Incorporated into masked memory, with the only limitation being that the $k >$ "key size" constraint is met. The code parameter considerations are the choice of N, R, and L. Choosing larger N, or block length, shows more "ideal" polar code behavior in that channels tend to polarize better at the cost of increased decoder complexity and latency for the main channel. The choice of R, or code rate, is also important, as for $R \ll C_s$, main channel

decoding performs well, but lots of information is leaked to the adversary. For $R \approx C_s$, main decoder reliability suffers, but with less adversary accuracy. In each case, increasing L can decrease errors at the cost of latency in the main channel, noting that performance is not much improved beyond L_{max}.

The strategy is then to find a combination of N and R that can meet sufficient reliability conditions for the main channel using a minimal L, while also meeting sufficient security conditions for the adversary channel with a very large L. To determine the optimal parameters for our masked memory setting, decoding block error rate (BLER) is used for both main and adversarial channels, while the estimated channel entropy is used for the adversarial channel. For the main channel, the latency of the decoder is also considered. Each metric is discussed next.

1) Block Error Rate: The BLER is defined for both the main and adversary channels to determine the overall effectiveness of the decoder in each respective channel. The vector Z is defined as the concatenation of each data bit $v \in \mathcal{I}$ (i.e., a vector of the original k information bits of V that hash down to sk). Then, $\hat{Z}$ and $\dot{Z}$ denote the main and adversary channel estimates of these bits from decoding $\hat{U} \rightarrow \hat{V}$ and $\dot{U} \rightarrow \dot{V}$, respectively.

The BLER is considered as:

$$\text{BLER} = \frac{\text{\# of trials where } a \neq b}{\text{total \# of trials}} \tag{9}$$

with $a = Z$, and

$$b = \begin{cases} \hat{Z}, & \text{for the main channel} \\ \dot{Z}, & \text{for the adversary channel} \end{cases}$$

For the main channel, this number should be low ($\leq 10^{-6}$), while for the adversary, we should always have block errors. The error patterns within the incorrectly decoded vectors can then be further analyzed.

2) Adversary Channel Entropy: As evident in [18], specific bits, z_i, have better reliability than others depending on their location in the source word (polar code construction tries to exploit this) [17], [42], [43]. This is also true for the adversarial channel. Using Equation 1, the approximated adversary entropy of a particular bit channel after decoding is:

$$y(i) = H(\Pr(z_i \neq \dot{z}_i)) \tag{10}$$

which can be used to analyze how secure the scheme is. As mentioned, a strong adversary will exhaustively attempt each SCL with Equation 8, so each of these L outputs must be considered.

Equation 10 can then be extended to SCL decoding. Letting $y_l(i)$ be the channel entropy for the i^{th} channel in the l^{th} SCL estimate, then ideally:

$$y_1(i) \approx y_2(i) \approx \cdots \approx y_L(i) \tag{11}$$

so that an adversary cannot use any produced estimate to their advantage. A subset of the adversary channels is defined as $\mathcal{D} \subseteq \mathcal{I}$ in which $D = \{j \in \mathcal{I} \mid y(j) \geq 1 - \delta\}$, with δ bringing a particular index entropy to an acceptable level.

Because the adversary has access to an oracle which can accurately verify the correctness of an arbitrary decoded word, an adversary may choose to use the existing L SCL candidates to form an *enhanced* list of SCL candidates $L' > L$. The additional $L' - L$ candidates can be formed by using a-priori information about the source word, (i.e., the relative reliability of each channel from $\mathcal{I}$) to change bits, or by attempting to merge the existing L candidates in a meaningful way. It may also be considered that the L' candidates can be sorted from most-to-least likely, allowing a systematic way to test each candidate with Equation 8. Let $\mathcal{L}$ be this ordered set of L' candidates.

If it is assumed that δ is sufficiently small, then a known $|\mathcal{D}|$ bits will appear as completely random to the adversary. If it is also assumed that there is no correlation between these channels for each of the candidates in $\mathcal{L}$, then the approximation in Equation 11 should hold. Also, the channels in $\mathcal{D}$ of the additional $L' - L$ candidates should be uncorrelated equally like the rest of the original L SCL candidates.

If there are $|\mathcal{D}|$ channels that have approximately full entropy, the adversary is correct with a probability of $\leq 2^{-|\mathcal{D}|}$. This can be verified with Equation 8. If the oracle returns 0, there is no additional information gained about the correct candidate, so the 2^{nd} most likely estimate is tested in a similar fashion. Similar to the first case, the probability of being correct should be $\leq 2^{-|\mathcal{D}|}$ provided i) the assumption about the L generated SCL source words being uncorrelated holds and ii) the indices in $\mathcal{D}$ truly have high entropy. This process is repeated for the remaining candidates in $\mathcal{L}$ the same way. The adversary then has an overall success probability of $L' \cdot 2^{-|D|}$ with an $O(L'N \log N)$ complexity, which is infeasible to compute for an exponentially increasing L'. We validate the set $\mathcal{D}$ experimentally, and show that the different candidates produced by the SCL decoder are uncorrelated.

Furthermore, it is noted that if the key is extracted from the correct source word by means of a cryptographic hash function, as is our case, where k bits are hashed down to 128 with $128 < |\mathcal{D}| - \log_2 L'$, then the adversary should just attempt to guess the session key with probability 2^{-128}. Security now reduces to that of the cryptographic hash function.

3) Latency Considerations: Analyzing latency enables masked memory implementation tailored for certain applications. Recall that the primary applications we consider are authentication in remote attestation and encryption. In the case of remote attestation, if we consider a decoding failure on the main channel, (i.e., when we retrieve our key from masked memory, there is an error), we would realize such an error existed due to the authentication failing, and simply re-request authentication. Besides the additional time required to re-attempt authentication, there would be no other consequence of failing the additional authentication. Thus, from a latency perspective, it might be beneficial to implement a smaller error correction code that requires less time to decode, and that has an acceptably low probability of error.

In the case of encryption, however, such a relaxation on the reliability is not permissible. If the decoding of the stored

key were to fail, which was then used to decrypt user data, there would be no way to verify its integrity. This would have serious consequences if this incorrect data were then used throughout a program. As a result, our masked memory implementation would require much harder requirements for encryption, with the tradeoff of longer decoder latency in the critical path.

IV. Methodology

A. Error Simulation

We model the performance of the system given in Figure 2 via simulation with an open-source Python polar code implementation[1]. The simulator is modified to automate the collection of bit error statistics for the different configurations, such as channel-dependent error rates to estimate channel entropy across each trial. Note that the main and adversary channel error probabilities are modifiable to model different scenarios. For instance, more aggressive adversarial models can be tested by decreasing p_a, or less reliable PUF architectures can be modeled with an increased p_m.

The process for masked memory simulation is as follows: for key enrollment, random vectors Z of length k are first generated to simulate the information bits, before being constructed into the source word V (using the Bhattacharyya polar code construction) and encoded to a length N block, U. We then find $X = U \oplus r$, where r is another random vector of length N to simulate a PUF-generated mask. Extraction is simulated by inverting this process. A uniform 10% noise is injected across the bits of r to model PUF noise and form $\hat{r}$, which forms $\hat{U} = X \oplus \hat{r}$, and is subsequently decoded with the k estimated information bits extracted into $\hat{Z}$ from the decoded source estimate $\hat{V}$.

For the adversary, a uniform 25% error rate is instead applied to r to form $\dot{r}$, modeling the additional noise induced by the ML model, and forming $\dot{U} = X \oplus \dot{r}$. $\dot{U}$ is then decoded, extracting the adversary information bit estimates into $\dot{Z}$. In all cases, the SCL decoder is used in the simulations (where a list size of $L = 1$ just indicates the original successive cancellation algorithm). The model is tested with different parameters (code block size, rate, and list size) to examine the tradeoff between latency, error correction performance, and security.

To find an upper bound of code rate for simulation, C_s is used. Using Equation 3 with $p_a = 0.25$ (as determined in Section II-D) and $p_m = 0.1$ (as determined in Section II-B), $C_s = 0.342$ is computed. As such, the rates of $R = \frac{1}{5}$, $R = \frac{1}{4}$, and $R = \frac{1}{3}$ are measured for block lengths of $N = 1024$ and $N = 2048$. Various list sizes $L = 1, 2, 4, 8, 16$ are also tested in order to find the SCL performance for each configuration. Certain configurations are then chosen to test $L = 128$.

Initially, 10^5 trials are performed with each configuration to obtain baseline results. Equation 9 is then used to determine the average BLER of each channel for each configuration. Configurations that show an adversary BLER of 1 for each trial are then tested at $L = 128$ to analyze if the decoding failures persisted at an exaggerated L. Additionally, those that have main BLER $< 10^{-6}$ and adversary BLER $= 1$ are chosen for 10^6 trials, and further analyzed at $L = 128$ for the adversary.

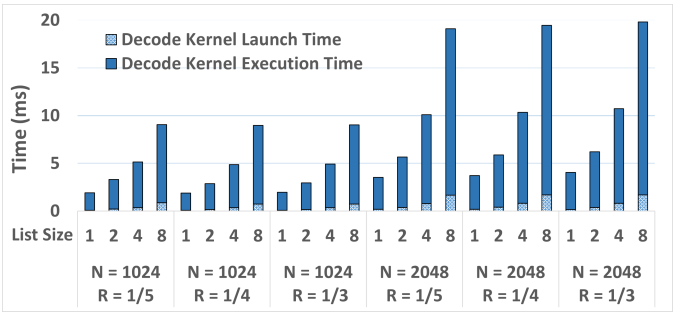

Fig. 4: Main channel latency of polar code GPU kernel for each configuration.

The adversary is considered for further analysis with the best N and R configurations at $L = 128$. Each channel's entropy is approximated by Equation 10 across all 10^6 experiments. Additionally, experiments are performed to analyze the the $L-1$ remaining candidates, counting the number of errors in each candidate and producing the minimum and maximum number of bit errors produced for a given run.

B. Latency Simulation

After evaluating the error correction settings for the main and adversary channels, the decoding scheme is evaluated for implementation costs using a GPU implementation of polar encoders and decoders. The Nvidia Quadro RTX 6000 GPU is used for evaluations with the polar codes library[2] [44].

The decoding latency is measured using the NVidia Nsight profiler. Each experiment is run individually for a given N, R, and L configuration. The latency measurements are reported using the time spent launching a new decoding kernel, and the kernel execution time spent in decoding algorithm in the GPU. The kernel launch time is included since a practical implementation of the decoder with a GPU in masked memory will need to invoke the kernel for each request.

V. Experimental Results

The experimental results obtained are summarized in Table I. Columns that are white denote configurations that have an adversary BLER $< 100\%$ (i.e., the adversary successfully decoded all k information bits perfectly in some trials), failing the security condition. Yellow configurations are shown to be secure, however have worse reliability than the $\leq 10^{-6}$ failure rate metric. Finally, the green configuration of $N = 2048$ with $R = \frac{1}{4}$ meet *both* the reliability and security conditions for the simulated trials, and are further analyzed according to the methodology. The reported errors are based on the vector Z, the k encoded data bits that hash down to the secret sk. In Figure 4, the latency impact for each configuration is shown for increasing list sizes (L). For the same N and L, not much latency variation is seen across each R. This is because changing the rate simply modifies the ratio of k to $N - k$

[1] https://github.com/RQC-QApp/polar-codes

[2] https://nvlabs.github.io/sionna/api/fec.polar.html

Block Length	N = 1024						N = 2048					
Rate	1/5		1/4		1/3		1/5		1/4		1/3	
List Size	Main	Adv.	Main	Adv.	Main	Adv.	Main	Adv.	Main	Adv.	Main	Adv.
1	0%	99.355%	0.007%	99.999%	2.127%	100%	0%	99.985%	0.0005%	100%	0.79%	100%
2	0%	97.552%	0.004%	99.998%	0.478%	100%	0%	99.909%%	0%	100%	0.205%	100%
4	0%	95.156%	0.003%	99.985%	0.254%	100%	0%	99.676%	0%	100%	0.111%	100%
8	0%	92.435%	0.003%	99.972%	0.209%	100%	0%	99.283%	0%	100%	0.126%	100%
16	0%	89.686%	0.003%	99.960%	0.149%	100%	0%	98.801%	0%	100%	0.100%	100%
128	-	-	-	-	0.199%	100%	-	-	0%	100%	0.119%	100%

TABLE I: BLER of main and adversary channels for different configurations.

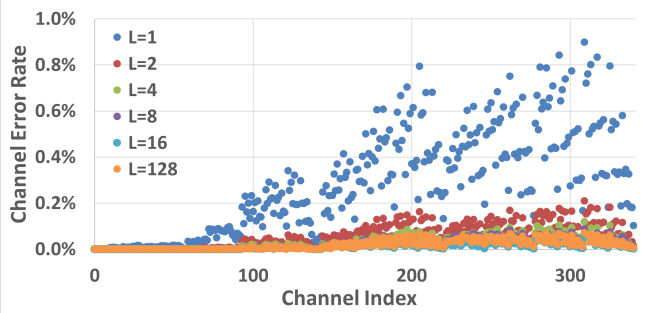

Fig. 5: Main channel error patterns for $N = 1024$, $R = \frac{1}{3}$.

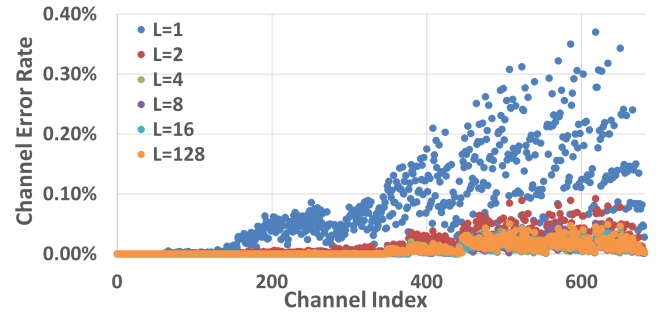

Fig. 6: Main channel error patterns for $N = 2048$, $R = \frac{1}{3}$.

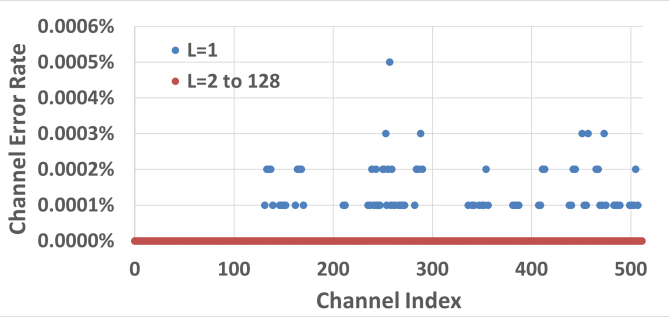

Fig. 7: Main channel error patterns for $N = 2048$, $R = \frac{1}{4}$.

bits. A total of $N \cdot L$ likelihoods are still computed, with the only difference being the decoder can automatically set a bit $i \in \mathcal{F}$ to '0' as in Equation 6, since the set $\mathcal{F}$ is known a-priori. The main contributors to increased latency are N and L for a given decoder configuration since the SCL decoder complexity is $\mathcal{O}(LN \log N)$. For $N = 1024$ vs. $N = 2048$, the latency is $\approx 2\times$, which is expected as SC decoding is an inherently serial algorithm. Increasing L allows the decoder to exploit some parallelism between branches, so rather than linear growth, slightly better latency is observed (for example, with $N = 2048$ and $R = \frac{1}{4}$, latency is $3.72ms$ at $L = 1$ and $19.45ms$ for $L = 8$, around $5.23\times$).

However, the cost of increasing L is still evidently significant. Therefore, minimizing L for the main channel while retaining the decoding efficacy must be prioritized. Exemplifying this is $N = 2048, R = \frac{1}{4}$ with $L = 1$ vs. $L = 2$. Increasing the list size effectively reduces the BLER in the main channel to 0% from 0.0005%, but with $1.71\times$ latency overhead. For applications where main BLER below 10^{-6} is needed, this tradeoff is worth using in the main channel.

A. Choosing Configurations for Masked Memory

For potential candidates highlighted in Table I, the bit error patterns of $\hat{Z}$ for each are shown in Figures 5, 6, and 7. While the primary metric of interest when analyzing main channel error is the BLER, these error patterns reveal additional insights. Firstly, the $R = \frac{1}{3}$ configurations shown in Figures 5 and 6 visualize the reliability of certain channels for various L. In both cases, the low channel error rates tend towards the lower indices, while larger channel indices see higher error rates. This is expected when constructing polar codes using the Bhattacharyya parameter, as can be seen in [17]. Another insight here is that past $L \approx 8$, larger list sizes do not further improve decoding, suggesting that $L = 8$ may be a practical bound for these configurations considering the latency and error correction tradeoff. Conversely, the additional reliability gained from increasing the list size from $L = 1$ to $L = 2$ is significant, consistent with the finding of Table I. So for an application where a number of block errors may be acceptable (such as in lightweight authentication protocols where re-attempts may be allowed, or even expected), choosing $L = 2$ shows a greater factor of error correction relative to the additional latency paid.

For the case of Figure 7, while some errors are observed at $L = 1$, increasing $L > 1$ resulted in no main channel BLER for this configuration, suggesting $L = 2$ as the realistic upper bound. For $L = 1$, out of the 5 trials resulting in BLER, some of the error channels are fairly predictable. For example, channel $i = 257$ resulted in an error for every BLER that was seen, suggesting that flipping this bit when encountering an error for this channel would correct it with high probability. Similar logic can be followed for other channels that saw repeated errors. While in a real setting, selecting a $L > 1$ would be more practical, it is interesting to observe this distribution across the channels.

The focus then shifts on the green column showing the

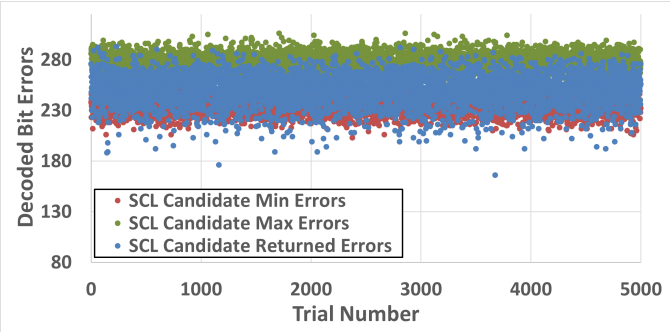

Fig. 8: Trial bit errors for the $N = 2048, R = \frac{1}{4}, L = 128$ parameters of adversary.

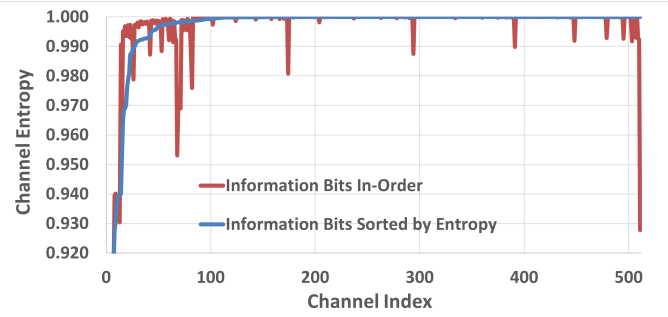

Fig. 9: The entropy of each data channel for the adversary.

$N = 2048$ and $R = \frac{1}{4}$ setting, or the $(2048, 512)$ polar code configuration. Considering the $L = 2$ case of the $(2048, 512)$ code for the main channel, an error rate of $\leq 10^{-6}$ can be achieved with a latency overhead of $5.89ms$ for decoding. This includes the time required to launch the decoder, which accurately models the scenario of a new KIS session key being requested at a certain period.

For the $L = 128$ case of the adversary, the error rates of the SCL decoder are analyzed. Three measurements are made: (1) The number of errors produced by the returned SCL estimate, which is the most likely transmitted source out of the L SCL paths. (2) The *minimum* number of errors produced by any of the remaining $L - 1$ candidates that are not ultimately returned. (3) The *maximum* number of errors produced by any of the remaining $L - 1$ candidates that were not ultimately returned. Experimental results are shown in Figure 8. It is seen that the SCL decoder has a significant number of channel bit errors when a block error is encountered. Out of $5k$ trials, the mean number of errors in the returned SCL estimates is 249.72, translating to a 47.88% error rate out of 512 data bits. Interestingly, the variance of the minimum and maximum error candidates not ultimately returned had a variance of 56.57 and 57.71, respectively, while the variance of the actual returned SCL estimate was 214.43. This suggests that when the SCL algorithm fails, and the other $L - 1$ less likely candidates are searched, there is a predictable range of channel errors that can be expected.

Although there are some SCL candidates in red that have the minimal number of errors in a particular trial, the overall best trials are those that come from the actual returned SCL estimate, as shown in Figure 8. However, even in these cases where the SCL algorithm performs relatively well, the actual source word is unrecoverable to the adversary, with each trial having ≥ 166 errors in the best case. Clearly, even with the oracle of Equation 8 available to the adversary, the properly decoded source word will never be found, resulting in an unsuccessful adversary.

Considering the polarization property of polar codes, the entropy of each decoded channel for the adversary is shown in Figure 9. While certain channels have lower overall entropy to the adversary (and thus are easier to decode), these most reliable channels do not jeopardize the security of the key. If an adversary knows the most reliable bit positions (i.e., those with the least entropy), the channels with the *most* entropy $\mathcal{D}$ must be determined to guarantee the scheme is secure, even if the adversary can leak bits in $\bar{\mathcal{D}} \cap \mathcal{I}$ with increased certainty.

Considering $\mathcal{D}$ as our set of channels that each has an entropy $\geq 1 - \delta$, and $\delta = 0.001$ for instance, sorting the channels by their entropy in Figure 9 gives 419 bits $\in \mathcal{D}$. As these bits appear completely random to an adversary, there are a total of $\approx -\log_2(L \cdot 2^{-419}) = 419 - \log_2(L)$ bits that need to be guessed, reducing the security to that of the hash as shown in Section III-C2.

These two additional results show that adversaries experiencing block errors cannot use the SCL decoder candidates, or the channel properties of polar codes themselves, to assist with decoding for $N = 2048, R = \frac{1}{4}$ with $L = 128$. It is also shown that for these same N and R, the main channel can achieve sufficiently low error with low list sizes.

Comparison to State-of-the-art [28]: Compared to KIS implementations where keys are generated on-the-fly with an overhead of $26.6ms$ [28], the proposed masked memory requires the additional overhead of decoding, a total of $5.49ms$, as S session keys can be pre-generated and securely stored in memory. This makes masked memory a practical method for the secure and reliable retrieval of keys.

VI. Conclusion

We have introduced the masked memory primitive, which utilizes a strong PUF and polar codes to improve the security of refreshing keys in KIS with low latency. After estimating the PUF and machine learning model parameters, masked memory is evaluated experimentally. It is found that for the polar code scheme of $(2048, 512)$ and $L = 2$, the main channel can effectively decode and recover all k information bits used to form session keys. It is further shown that for an adversary, this same scheme with $L = 128$ does not allow for proper decoding, resulting in significant block error rates. Furthermore, the adversary gains no advantage by searching through all SCL decoder candidates, with adversary channels having sufficient entropy among all source word estimates to inhibit decoding. This results in the strong security of session keys as they are retrieved from memory.

VII. Acknowledgments

This research was supported by the National Science Foundation under Grants No. 1929261 and 1950600.

References

[1] M. Lipp, A. Kogler, D. F. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "Platypus: Software-based power side-channel attacks on x86," *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 355–371, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:227222948

[2] N. Lawson, "Side-channel attacks on cryptographic software," *IEEE Security & Privacy*, vol. 7, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:16125684

[3] Y. Dodis, J. Katz, S. Xu, and M. Yung, "Key-insulated public key cryptosystems," in *Advances in Cryptology—EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques Amsterdam, The Netherlands, April 28–May 2, 2002 Proceedings 21*. Springer, 2002, pp. 65–82.

[4] ——, "Strong key-insulated signature schemes," in *Public Key Cryptography*, 2003.

[5] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 148–160.

[6] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[7] C. Jin, W. Burleson, M. van Dijk, and U. Rührmair, "Programmable access-controlled and generic erasable puf design and its applications," *Journal of Cryptographic Engineering*, vol. 12, no. 4, pp. 413–432, 2022.

[8] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," *Proceedings of the IEEE*, vol. 102, pp. 1126–1141, 2014.

[9] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Techniques for design and implementation of secure reconfigurable pufs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, pp. 5:1–5:33, 2009.

[10] S. Tajik, E. Dietz, S. Frohmann, J.-P. Seifert, D. Nedospasov, C. Helfmeier, C. Boit, and H. Dittrich, "Physical characterization of arbiter pufs," in *Cryptographic Hardware and Embedded Systems–CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings 16*. Springer, 2014, pp. 493–509.

[11] R. Maes and I. Verbauwhede, "Physically unclonable functions: A study on the state of the art and future research directions," *Towards Hardware-Intrinsic Security: Foundations and Practice*, pp. 3–37, 2010.

[12] G. T. Becker and R. Kumar, "Active and passive side-channel attacks on delay based puf designs," *IACR Cryptol. ePrint Arch.*, vol. 2014, p. 287, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:875095

[13] A. Mahmoud, U. Rührmair, M. Majzoobi, and F. Koushanfar, "Combined modeling and side channel attacks on strong pufs," *IACR Cryptol. ePrint Arch.*, vol. 2013, p. 632, 2013.

[14] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," *IACR Cryptol. ePrint Arch.*, vol. 2010, p. 251, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:13961278

[15] Y. Yu, M. Moraitis, and E. Dubrova, "Profiled deep learning side-channel attack on a protected arbiter puf combined with bitstream modification," *Cryptology ePrint Archive*, 2020.

[16] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. D. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *SIAM journal on computing (Print)*, 2004.

[17] E. Arkan, "A performance comparison of polar codes and reed-muller codes," *IEEE Communications Letters*, vol. 12, 2008.

[18] H. Mahdavifar and A. Vardy, "Achieving the secrecy capacity of wiretap channels using polar codes," *IEEE Transactions on Information Theory*, vol. 57, pp. 6428–6443, 2010.

[19] Y. Bai and Z. Yan, "A secure and robust key generation method using physical unclonable functions and polar codes," *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 254–259, 2019.

[20] B. Chen, T. Ignatenko, F. M. Willems, R. Maes, E. van der Sluis, and G. Selimis, "A robust sram-puf key generation scheme based on polar codes," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[21] Y. Bai and Z. Yan, "Physical unclonable functions with improved robustness based on polar codes," in *2017 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2017, pp. 1–6.

[22] I. Tal and A. Vardy, "List decoding of polar codes," *2011 IEEE International Symposium on Information Theory Proceedings*, pp. 1–5, 2011.

[23] R. Maes, "An accurate probabilistic reliability model for silicon pufs," in *Cryptographic Hardware and Embedded Systems-CHES 2013: 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings 15*. Springer, 2013, pp. 73–89.

[24] Q. Ma, C. Gu, N. Hanley, C. Wang, W. Liu, and M. O'Neill, "A machine learning attack resistant multi-puf design on fpga," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 97–104.

[25] H. Kareem and D. Dunaev, "Machine learning vulnerability assessment of ring oscillator physical unclonable functions," *2023 International Conference on Control, Automation and Diagnosis (ICCAD)*, pp. 1–5, 2023.

[26] U. Rührmair, H. Busch, and S. Katzenbeisser, "Strong pufs: models, constructions, and security proofs," *Towards Hardware-Intrinsic Security: Foundations and Practice*, pp. 79–96, 2010.

[27] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Transactions on Signal Processing*, vol. 61, pp. 289–299, 2013.

[28] D. Gurevin, C. Jin, P. H. Nguyen, O. Khan, and M. van Dijk, "Secure remote attestation with strong key insulation guarantees," *IEEE Transactions on Computers*, 2023.

[29] M. Gao, K. Lai, and G. Qu, "A highly flexible ring oscillator puf," in *Proceedings of the 51st annual design automation conference*, 2014, pp. 1–6.

[30] Y. Gao, S. F. Al-Sarawi, and D. Abbott, "Physical unclonable functions," *Nature Electronics*, vol. 3, no. 2, pp. 81–91, 2020.

[31] M. Hiller, M.-D. Yu, and G. Sigl, "Cherry-picking reliable puf bits with differential sequence coding," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 9, pp. 2065–2076, 2016.

[32] M. Hiller and A. G. Önalan, "Hiding secrecy leakage in leaky helper data," in *Workshop on Cryptographic Hardware and Embedded Systems*, 2017.

[33] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proceedings of ICC'93-IEEE International Conference on Communications*, vol. 2. IEEE, 1993, pp. 1064–1070.

[34] S. B. Wicker and V. K. Bhargava, *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

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

[36] B. Tahir, S. Schwarz, and M. Rupp, "Ber comparison between convolutional, turbo, ldpc, and polar codes," *2017 24th International Conference on Telecommunications (ICT)*, pp. 1–7, 2017.

[37] E. Arıkan, "Polar codes: A pipelined implementation," in *Proc. 4th ISBC*, vol. 2010, 2010, pp. 11–14.

[38] M. van Dijk and C. Jin, "A theoretical framework for the analysis of physical unclonable function interfaces and its relation to the random oracle model," *Journal of Cryptology*, vol. 36, no. 4, p. 35, 2023.

[39] A. D. Wyner, "The wire-tap channel," *The Bell System Technical Journal*, vol. 54, pp. 1355–1387, 1975.

[40] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[41] A. D. Wyner and J. Ziv, "A theorem on the entropy of certain binary sequences and applications-i," *IEEE Trans. Inf. Theory*, vol. 19, pp. 769–772, 1973.

[42] R. Mori and T. TANAKA, "Performance of polar codes with the construction using density evolution," *IEEE Communications Letters*, vol. 13, 2009.

[43] Y. Zhou, R. Li, H. Zhang, H. Luo, and J. Wang, "Polarization weight family methods for polar code construction," *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, pp. 1–5, 2018.

[44] J. Hoydis, S. Cammerer, F. A. Aoudia, A. Vem, N. Binder, G. Marcus, and A. Keller, "Sionna: An open-source library for next-generation physical layer research," *arXiv preprint arXiv:2203.11854*, 2022.