Haystack ciphers: White-box countermeasures as Symmetric encryption*

Alex Charlès[1] and Aleksei Udovenko[2]

[1] DCS, University of Luxembourg, Esch-sur-Alzette, Luxembourg,
alex.charles205@gmail.com
[2] SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg,
aleksei@affine.group

Abstract. In the area of white-box cryptography implementations, many existing protections are susceptible to attacks derived from physical cryptanalysis, which can be applied with minimal human effort and no prior design knowledge. The absence of a clear and comprehensive security model hinders the development of effective countermeasures against these attacks.

We introduce the Haystack ciphers, a formal model for the security of white-box countermeasures against such attacks. In this model, the countermeasures are represented simply as symmetric-key encryption schemes. We show that their *chosen-plaintext (IND-CPA)* security is closely related to the resistance of the countermeasures against computational trace-based attacks. Similarly, their *chosen-ciphertext (IND-CCA)* security is closely associated with the resistance against fault injection attacks in the white-box model. Secure Haystack ciphers constitute the next formal milestone for advancing white-box designs and countermeasures, the minimal requirement that is not currently clearly achieved but is plausibly feasible with available tools.

We review the white-box literature with respect to our model and bridge the gap between white-box and fault attacks, which are very powerful but were only partially considered in the white-box literature so far. We study known fault protections from the physical cryptography literature and present new fault attacks in the white-box setting, which raises the need and shapes the requirements for future secure countermeasures against fault attacks.

Keywords: White-box cryptography · Haystack ciphers · Security model · Symmetric cryptography · Fault injection · CPA · CCA

1 Introduction

In [Koc96], Kocher demonstrated that state-of-the-art asymmetric ciphers can be efficiently broken with access to side-channel information on the cryptographic hardware. Later, this approach was extended to symmetric cryptography [KJJ99]. A few years later, Chow, Eisen, Johnson, and van Oorshot

* Preliminary full version of the paper accepted at ASIACRYPT 2025, with major differences.

[CEJv02,CEJvO03] expanded the question by proposing a software implementation resistant to an attacker having *direct* access to the computation process, calling this model white-box cryptography. Studies mainly focused on protecting the DES and the AES block ciphers [DES77,DR98].

The initial white-box designs were based on protecting internal information with small nonlinear encodings and performing computations using a network of lookup tables. Various countermeasures were proposed based on this encoding principle, but all were broken [BGEC04,DRP13,LRD+14]. In 2015-2016, two works [SMdH15,BHMT16] showed that basic physical cryptanalysis methods can easily break white-box encoding-based countermeasures. These attacks, based on on Differential Fault Analysis (DFA) [BS97] and Differential Power Analysis (DPA) [KJJ99] respectively, require observing values of internal computations or injecting errors during computations and analyzing the faulty outputs.

Different families of countermeasures were introduced, some trying to hide the structure of the cipher itself using obfuscation techniques to prevent the attacker from studying only the part of interest of the implementation, others trying to protect each bit variable locally using masking schemes [BU18,SEL21] or shuffling methods [BU21]. A good example of a complex white-box attack is [GPRW20], in which the authors show how to reverse-engineer the obfuscation layer and then uncover key information locally from the white-box winning countermeasure of the 2017 WhibOx contest. Another example demonstrating a structural analysis and the utilization of fault attacks against the winning challenges of the 2019 WhibOx contest can be found in [TGCX23].

The attacker's possibilities in white-box are deep and broad, resulting in unclear adversary definitions and countermeasure security statements. In [BU18], the authors tried to classify the attacks and countermeasures in two categories: *structure hiding* and *value hiding*. The former encapsulates preventing an attacker from understanding the cryptographic implementation's code and structure. The latter represents the inability of an attacker to extract the sensitive intermediate information from the implementation. However, no precise definitions for these concepts were given.

To clarify the attacker possibilities and to simplify countermeasure studies, in this work, we propose the first mathematical model for the *value hiding* concept and consider white-box attacks that work without any knowledge of the obfuscated implementation structure and their associated countermeasures, which can also be called *trace-based* or *automated* attacks. Such attacks are often tried first, as they do not require human effort, so any white-box countermeasure must thwart them. We also emphasize that our work is not limited to cryptographic primitives, but to all stateless software implementations.

Our contribution *(Haystack ciphers)* Our key contribution is the first security model for white-box countermeasures based on symmetric-key cryptography security notions. For a given countermeasure, we define a corresponding symmetric encryption scheme denoted by a *haystack cipher*. The key observation is that classic security notions for such schemes closely encompass the automated white-box attacks that we aim to formalize. More precisely, *indistinguishability*

under chosen-plaintext attacks (IND-CPA) of the haystack cipher establishes security against trace-based attacks; *indistinguishability under chosen-ciphertext attacks* (IND-CCA) establishes security against a wide range of fault-injection attacks. While this model is simplistic, it encapsulates state-of-the-art trace-based attacks and their countermeasures and provides a minimal requirement for white-box designs.

(Trace-based attacks in the CPA model) We define the Haystack ciphers of the countermeasures from the literature that aim to protect against the trace-based attacks, *e.g.* attacks that do not rely on the analysis of the structure of the implementation but solely on the information present in traces in an automated manner. To illustrate our model in a practical context, we begin by examining the linear decoding attack [GPRW20] against the ISW masking scheme [ISW03] and the code-based countermeasures [WMCS20].

(Fault attacks in the CCA model) We propose a first formal study of the fault attacks in the white-box model, assessing physical cryptanalysis fault attacks and countermeasures, and presenting new white-box fault attacks. Extending the work of [GPRW20], we show that an attacker can eliminate pseudorandom computations in an implementation in a complexity that depends on the countermeasure employed. We also demonstrate that forgery attacks in the CCA model represent undetected faults, such as ineffective fault attacks, as seen in SIFA [DEK+18]. We propose new white-box fault attacks to remove the redundancy of correcting schemes using Multi-Persistent Fault Attack (MPFA), and to break any code-based countermeasure using linear fault recoding (LFR) by injecting faults depending on the observed linear structure of the encoded output.

Limitations While our model gives a great understanding of adversary abilities and exposes necessary conditions for a countermeasure to be secure in the white-box setting, it does not prove the total security of an implementation in the white-box model. Firstly, our model does not assess implementation structure analysis tools and countermeasures. Furthermore, a Haystack cipher is defined only on the decoding function, while the corresponding gadgets are also essential to secure [BBD+16,BU21].

A supporting code implementing white-box countermeasures as Haystack ciphers and main attacks against them in SageMath [SD25] is available at [CU25][3].

2 Preliminaries and state-of-the-art

We will use the following notation throughout the paper. The letter $\mathbb{Z}$ denotes the set of integers, $\mathbb{Z}_{\geq 0}$ the set of nonnegative integers, $\mathbb{Z}_{>0}$ the set of positive integers. The matrix multiplication constant is denoted by ω ($\omega = \log_2 7 \approx 2.8$ is achieved by Strassen's method [Str69]). The finite field of order q is denoted by $\mathbb{F}_q$, and the n-dimensional vector space over $\mathbb{F}_q$ is denoted $\mathbb{F}_q^n$. The vectors are written in bold, e.g. $\boldsymbol{x} \in \mathbb{F}_q^n$. Addition element-wise in $\mathbb{F}_2^n$ is called XOR

[3] See also https://github.com/cryptolu/HaystackCiphers.

and is denoted by "$\oplus$" or simply "$+$", and multiplication element-wise is called AND and is denoted by "$\cdot$". The symbol "$\perp$" denotes a decryption failure (in the IND-CCA games, see Subsection 2.4) or a fault detection by a white-box countermeasure.

By $\mathrm{Shuffle}_{\boldsymbol{k}}$ we denote a function that takes as input a finite list of arbitrary size and shuffles the elements pseudorandomly based on the seed $\boldsymbol{k}$ and the length of the list, and returns the shuffled list. It is a deterministic algorithm when $\boldsymbol{k}$ and the input length are fixed. We assume uniform distribution of the shuffling permutations across different seeds $\boldsymbol{k}$. By $\mathrm{Shuffle}_{\boldsymbol{k}}^{-1}$ we denote the inverse of this function: $\mathrm{Shuffle}_{\boldsymbol{k}}^{-1} \circ \mathrm{Shuffle}_{\boldsymbol{k}} = \mathrm{Shuffle}_{\boldsymbol{k}} \circ \mathrm{Shuffle}_{\boldsymbol{k}}^{-1}$ is the identity mapping.

2.1 Generic automated traced-based and fault injection attacks

We now briefly review the state of the art of generic automated white-box attacks, a topic that originated in [SMdH15,BHMT16] and is now an active research trend in the field. In practice, many attacks from this section can and often are performed on software implementations, using specialized debugging tools [Teu25]. However, for analysis purposes, it is common to consider implementations in terms of Boolean circuits. We emphasize that many concepts presented here are agnostic of the computational model, and keeping the circuit model in mind mainly simplifies the presentation.

Definition 1. *A* Boolean circuit *is a directed acyclic graph where each non-initial node is called a* gate *and has an assigned Boolean operation to it (such as AND, XOR, NOT).*

Computational traces An adversary in possession of a white-box implementation may freely execute it on arbitrary inputs (plaintexts) and record all *intermediate computations*, such as memory accesses (reads/writes) in a program or values of all gates in a circuit. We will assume that intermediate computations can be linked across different executions (which is trivial in circuits); synchronization of computations is a different topic. As a result, one execution record can be represented by a bitstring called a *trace*. Naturally, each trace has an associated plaintext and ciphertext. We will denote by T the number of traces recorded by an adversary for further attacks. Each position in the trace is called a *node* and we may consider the sequence of all values recorded in a node across T different executions, which is a bit-vector of length T, called a *node vector*.

Selection function Once an attacker has generated traces of a protected white-box implementation, they aim to extract information about the key. Since the base algorithm and the plaintext are known, the attacker can, by enumerating possible values of a part of the key, predict a value that the white-box implementation will process (in some way) for the correct key part guess. Such a function is called a *selection function*. A classic example of a selection function is the output byte (or one of its bits) of the first-round Sbox of the AES block cipher [DR98,DR02]. Indeed, each of the 16 plaintext bytes is known to the attacker, it is being XORed with the unknown key byte, and is processed by the

AES Sbox. For each of the 16 bytes, brute-forcing every 256 key byte possibility gives the attacker the proper AES Sbox output only for the correct key byte guess.

Non-invasive attacks For each trace, the attacker can compute the selection function for each candidate of the key part, which is again a bit-vector of size T, called a *selection vector*. The set of all the selection vectors required to launch an attack is denoted by $\mathcal{K}$. Now, the attacker needs to create a distinguisher that takes the node vectors and the selection vectors as input and determines which of the selection vectors is related to the computations in some way (statistical, algebraic, etc.).

Such a distinguisher has a time complexity that depends on the number of node vectors considered and the parameters of the target countermeasure. For different flavors of attacks, the complexity can range from linear to polynomial or even exponential (*c.f.* [CU24], Table 1). Many attacks are impractical when run on full implementations (i.e., with all node vectors as the input) and are only run on small subsets of the trace vectors at a time (a so-called sliding window). Due to this constraint, we refer to them as *local attacks*.

Invasive attacks One can also inject one or more errors (called *faults*) into the implementation's computations to recover key parts, for instance by modifying some gates of a copy of the base program. As opposed to the physical cryptanalysis setting, faults in the white-box model are bit-precise and guaranteed to succeed; an attacker can perform multiple faults at the same time, and is not limited to the knowledge of the output of the implementation, but also has access to the node values and can observe the impact of the fault on all the nodes. Fault attacks are less studied in the white-box model, while they can bypass most countermeasures due to their effectiveness and simplicity of mounting.

A needle in the haystack To identify the key parts or locate the correct fault locations, the attacker must find node vectors related to the selection function, which are typically hidden within surrounding unrelated computations that serve as a source of randomness. By studying the cipher structure manually [GPRW20] or automatically [TGCX23], an attacker can potentially find small subsets of node vectors for the local attack to succeed. This technique is broadly called *data-dependency* analysis. Alternatively, an attacker may use a sliding window approach, assuming that the nodes related to each other are close to each other in the traces [CU24]. This is a typical assumption in generic automated white-box attacks, as it does not impose any requirements on the implementation. Resisting such attacks is already a challenging open problem, and in our work, we aim to gain a deeper understanding of it, particularly through its formal modeling.

2.2 Countermeasures and trace-based attacks

DCA and ISW Masking scheme The first non-invasive local attack, differential computational analysis (DCA), was introduced by [BHMT16] and is

derived from the classic differential power analysis (DPA) [KJJ99]. The idea is to compute the correlation between the node vectors and the selection vector candidates. Encoding-based countermeasures are biased [RW19], resulting in a successful attack. To thwart this attack, the node vectors need to be uncorrelated to the selection vectors, which can be achieved using the ISW masking scheme [ISW03].

A masking scheme encodes every bit variable in the base implementation by n *shares* and replaces every bitwise operation by a *gadget*. Gadgets take as input the shares of the original inputs and output shares of the original outputs, possibly using additional randomness in the process. The *decoding function* of a masking scheme takes all of the shares of a variable and recovers the original variable. The ISW masking scheme encodes a bit variable x into ℓ shares $(x_1, \cdots, x_\ell)$, with $x = x_1 \oplus \cdots \oplus x_\ell$, such that $\ell - 1$ shares are chosen randomly. Therefore, given two encoded variables x shared over $(x_1, \cdots, x_\ell)$ and y shared over $(y_1, \cdots, y_\ell)$, a gadget, for instance, the gadget AND, will take as input these 2ℓ shares and will return $(z_1, \cdots, z_\ell)$, such that $z = x \cdot y = z_1 \oplus \cdots \oplus z_\ell$. Since $\ell - 1$ shares of ISW are chosen randomly, the correlation of each share with the original encoded variable is zero; therefore, schemes allow thwarting the DPA attack.

LDA and BU Masking Scheme Unlike in the side-channel field, a white-box cryptography attacker has access to noiseless traces. Therefore, for T traces, the node vectors can be considered as vectors in $\mathbb{F}_2^T$, allowing us to apply linear algebra.

In [GPRW20], the authors proposed the Linear Decoding Analysis (LDA) local attack. By denoting $\boldsymbol{a}$ the node vector of a node a, if ISW with ℓ shares has been employed, we know that there exist ℓ node vectors $(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_\ell)$ such that, for the correct selection vector $\boldsymbol{v}$, we have $\boldsymbol{x}_1 \oplus \cdots \oplus \boldsymbol{x}_\ell = \boldsymbol{v}$. So, given a subset of node vectors in a matrix M, and the selection vectors $(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_{|\mathcal{K}|})$, the LDA attack solves the linear system $M \cdot \boldsymbol{x} = \boldsymbol{v}_i$, for an unknown $\boldsymbol{x}$ and for $i \in (1, \cdots, |\mathcal{K}|)$. The attack succeeds for the correct selection vector if all of the node vectors corresponding to the shares of the selection vector are contained in the matrix M, even if M contains unrelated node vectors.

This attack has low complexity, so it was a threat to existing white-box countermeasures. In [BU18], the authors introduced a new masking scheme with the decoding function $x = x_1 \oplus x_2 \cdot x_3$. Since the decoding function is non-linear, it thwarts the LDA attack. One can observe that x is correlated with x_1, making it susceptible to the DCA attack. This is why the authors suggested combining their scheme with ISW by applying one on top of the other.

Higher-Order Attacks and Dummy-Shuffling Countermeasure Although the combination of ISW and BU masking schemes resists the DCA and LDA attacks [BRVW19], it is possible to enhance the DCA attack to compromise the ISW masking scheme. This leads to the so-called Higher-Order DCA (HODCA), which needs to have an order greater than or equal to the number of ISW shares while maintaining exponential time complexity in that order.

Increasing the number of shares of the ISW masking scheme is relatively cheap, making HODCA impractical due to its exponential time complexity. However, in [GPRW20], the author proposed a similar generalization of the LDA attack, called Higher Degree Decoding Analysis (HDDA). Although HDDA is exponential with its degree, the decoding function of BU is only a polynomial of degree 2, and HDDA of degree two can break it relatively efficiently.

As opposed to the ISW masking schemes, [BU18] did not propose any solutions to increase the degree of thwarting HDDA. Therefore, three years later, the same authors proposed in [BU21] a new countermeasure called Dummy Shuffling, creating s copies of the implementation and randomly choosing one of them to obtain the real input, while the others are fed random data. The authors showed that to break this countermeasure, HDDA should be of degree s, which, with enough slots, makes this countermeasure efficient against it. Once combined with ISW, the resulting countermeasure was conjectured to be resistant to state-of-the-art local attacks.

FLDA and SEL masking scheme The same year, in [SEL21], a new masking scheme was proposed, merging ISW with BU masking schemes. It encodes a bit variable x into $\ell + d$ shares $(x_1, \cdots, x_\ell, \tilde{x}_1, \cdots, \tilde{x}_d)$, such that $x = \bigoplus_{i=1}^{\ell} x_i \oplus \prod_{i=0}^{d} \tilde{x}_i$. HODCA must be of order ℓ and HDDA of degree d to break it. However, in [SEL21], only gadgets up to degree 3 are provided, meaning that the security is limited by a polynomial-time attack of degree less than 9. Moreover, the authors showed in [CU23] that high-degree instances of the SEL masking schemes were susceptible to an attack based on the Learning Parity with Noise problem. The following year, we introduced a new class of local attacks that broke the SEL masking scheme in polynomial time, regardless of which parameters ℓ and d are chosen [CU24].

2.3 Fault attacks and countermeasures

Physical cryptanalysis fault attacks In [BS97], the authors proposed injecting a fault in a hardware device to observe its impact on the output, a Differential Fault Attack (DFA). If the fault was injected at the correct position, some key parts could be retrieved depending on the difference observed. Later, fault techniques affecting the output of a cryptographic primitive were referred to as effective attacks. Multiple other effective fault attacks were later introduced [PQ03,LSG+10,LRD+12,FJLT13].

Moreover, it was shown that successfully injecting a fault that results in a zero output difference also gives information on the key [Cla07]; such attacks are referred to as ineffective fault attacks. The idea to use statistical analysis was generalized in [DEK+18], expanding the potential of ineffective fault attacks.

Physical cryptography countermeasures Countermeasures against fault attacks can take multiple forms on the hardware setup [BBB+22]. However, we will focus on countermeasures based on computational redundancy to fit the

white-box context. To thwart an injected fault, a countermeasure needs to detect it. For instance, code-based masking [WMCS20] replaces bit variables with codewords of a *error detection code* and verifies if a fault has been detected. Similarly, *time redundancy with comparison* [BCN+06] performs computation twice to observe if their results match.

Once a fault is detected, a countermeasure can perform a *correction* or an *infection*/rejection approach. The former consists of correcting the fault, so the output of the cipher remains unchanged. In contrast, the latter consists of expanding the fault so the output does not carry any useful information. In our analysis, if a fault is detected, the implementation's output is denoted by $\perp$.

White-box fault attacks In the white-box model, few studies explored the fault attack methodology besides seminal works [SMdH15], while it remaining a threat to white-box countermeasures. Indeed, similar to the trace-based attacks, faults can be performed without human resources or design knowledge. In the WhibOx capture the flag challenges of the CHES conference of 2017 and 2019, the JeanGrey tool[4], implementing the DFA attack using multi-faulting, showed to be very efficient against some of the submitted algorithms [AT20]. Similarly, the authors of [GPRW20] showed that it is possible to remove pseudorandomness from a white-box implementation using faults. It was also revealed that faulting enables the identification of the main slot of the Dummy Shuffling countermeasure [GRW20]. Finally, in [EGMC25, Chapter 2], a double fault was proposed to retrieve the locations of the linear shares of ISW. In this work, thanks to our Haystack model, we will define these fault attacks more precisely and show new ones against physical cryptography countermeasures in the white-box model.

2.4 Symmetric-key Security

Our new white-box model is based on the classic indistinguishability security notions and games, which we recall now.

Definition 2 ([BN00]). *A symmetric encryption scheme is a tuple of stateless algorithms (K, E, D). The randomized key generation algorithm K takes no input and returns a key k. The randomized encryption algorithm E takes as input a key k and a plaintext p to return a ciphertext c. The deterministic decryption algorithm D takes as input the key k and a string c to return either the corresponding plaintext p or the symbol $\perp$. We require that $D_k(E_k(p)) = p$ with probability one for all p, where the probability is over the choice of k and the randomness of E.*

We will need the following security notions: indistinguishability under chosen-plaintext attack (IND-CPA), indistinguishability under chosen-ciphertext attack [BDJR97,BN00] (non-adaptive - IND-CCA1, adaptive - IND-CCA2), and authenticated encryption (IND-CCA3, as in [Shr04]). Informally, in the corresponding games exposed in Section A, an adversary has access to *encryption oracle*

[4] https://github.com/SideChannelMarvels/JeanGrey

and has the goal of winning the indistinguishability challenge: the adversary can submit two messages, obtain the encryption of one of them and guess which one it was. In the chosen ciphertext games, the adversary has additional access to the *decryption oracle*. In IND-CCA1, the decryption was only accessible before the challenge was started. In IND-CCA2, the decryption can also be called afterward, except that the challenge ciphertexts cannot be queried. In IND-CCA3 (also known as authenticated encryption), producing any new valid ciphertext not created by the oracle also qualifies as a win.

Definition 3. *Let* $S = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be an encryption scheme as above. For* $G \in (\text{IND-CPA}, \text{IND-CCA1}, \text{IND-CCA2}, \text{IND-CCA3})$ *and the corresponding procedures given in Section A, for an adversary* A*, define* $\text{Exp}_{G,A}$ *as a probabilistic experiment where the* ***Initialize*** *procedure is executed once and then the adversary is executed while having oracle access to all other procedures. Calling the* ***Finalize*** *procedure finishes the experiment. Set* $\text{Exp}_{G,A} \Rightarrow 1$ *if "Win" was returned during the experiment (either from the Forge (IND-CCA3) or from the Finalize procedures), and* $\text{Exp}_{G,A} \Rightarrow 0$ *otherwise. The advantage of the adversary is defined as*

$$\mathbf{Adv}_S^G(A) = 2 \cdot \Pr[\text{Exp}_{G,A} \Rightarrow 1] - 1.$$

We say that S *achieves* G*-security if the advantage is negligible for all efficient adversaries.*

We remark that the described security notions are developed to formalize and quantify the ability of an adversary to extract confidential information from encryption/decryption oracles or to compromise integrity. The games themselves are formal constructs that do not directly correspond to attack scenarios.

Note that the CCA1 model itself does not prohibit any forgeries, although often a successful forgery leaks information about the scheme. Indeed, the decryption oracle is useless if forgeries are complex to create. It is well known that authenticated encryption (IND-CCA3) follows from IND-CPA security and security against forgeries (integrity). We will explore these ideas further in Section 5.

In the CCA2 model, it is sufficient to create a forgery related to one of the challenge plaintexts, allowing the two challenge ciphertexts to be distinguished. An instantiation of this idea is formalized below. An example would be a forgery that only changes a part of the corresponding plaintext (e.g., one or a few bits).

Proposition 1. *Let* $P \subseteq \mathbb{F}_2^n$ *be a set of messages,* $\boldsymbol{p}_0 \in P$ *and* $\boldsymbol{p}_1 \in \overline{P}$*. If, given an encryption* $\boldsymbol{c} = E_{\boldsymbol{k}}(\boldsymbol{p}_b), b \in (0,1)$*, an adversary can produce a valid ciphertext* $\boldsymbol{c}'$ *decrypting to* $D_{\boldsymbol{k}}(\boldsymbol{c}') = \boldsymbol{p}'_b$*, where* $\boldsymbol{p}_b$ *and* $\boldsymbol{p}'_b$ *are either both in* P *or both outside, then the target encryption scheme is not IND-CCA2-secure.*

In the CCA3 model, it is sufficient to create any forgery (called *existential forgery*) to win the game, by design. In most cases, this is the main attack direction against a scheme claiming IND-CCA3 security. Otherwise, as outlined above, the attacker must deal with the IND-CPA security scheme.

Definition 4. *Let $E : \mathbb{F}_2^n \to \mathbb{F}_2^m$, $D : \mathbb{F}_2^m \to \mathbb{F}_2^n \cup (\perp)$ be encryption/decryption functions (note that E implicitly takes randomness as an extra input). For each possible key $\boldsymbol{k}$, define the* randomness *of $E_{\boldsymbol{k}}$ by $\mathcal{N}_{rand.}(E_{\boldsymbol{k}})$, and the* redundancy *of D_k by $\mathcal{N}_{red.}(D_{\boldsymbol{k}})$, such that :*

$$\mathcal{N}_{rand.}(E_{\boldsymbol{k}}) = \log_2 |\operatorname{Im}(E_{\boldsymbol{k}}))| - n$$
$$\mathcal{N}_{red.}(D_{\boldsymbol{k}}) = m - \log_2 |(\boldsymbol{c} \in \mathbb{F}_2^m \mid D_{\boldsymbol{k}}(\boldsymbol{c}) \neq \perp)|.$$

For simplicity, we will assume that both measures are independent of the chosen key, that each message has exactly $2^{\mathcal{N}_{\text{rand.}}(E_{\boldsymbol{k}})}$ possible encryptions, and that each valid ciphertext can be produced by the encryption function with the same key. In this case, the general inequality $m \geq n + \mathcal{N}_{\text{rand.}}(E_{\boldsymbol{k}}) + \mathcal{N}_{\text{red.}}(D_{\boldsymbol{k}})$ becomes an equality $\mathcal{N}_{\text{rand.}}(E_{\boldsymbol{k}}) + \mathcal{N}_{\text{red.}}(D_{\boldsymbol{k}}) = m - n$.

The main goal of these measures is to show that ciphertext expansion is necessary and that reducing the ciphertext size is a valid attack.

Proposition 2. *An encryption scheme $E_{\boldsymbol{k}}$ can be attacked in the IND-CPA model in $O(\sqrt{2^{\mathcal{N}_{rand.}(E_{\boldsymbol{k}})}})$ queries and operations.*

Proof. The attacker creates a pool of $O(\sqrt{2^{\mathcal{N}_{\text{rand.}}(E_{\boldsymbol{k}})}})$ encryptions of $\boldsymbol{p}_0$ and an analogous pool for the challenge $\boldsymbol{p}_b$. If $b = 0$, with overwhelming probability, there will be a collision due to the birthday paradox, and no collisions if $b = 1$.

By definition of redundancy, the probability of a randomly sampled ciphertext being valid is $2^{-\mathcal{N}_{\text{red.}}(D_k)}$.

Proposition 3. *A decryption scheme $D_{\boldsymbol{k}}$ can be attacked in the IND-CCA3 model in $O(2^{\mathcal{N}_{red.}(D_{\boldsymbol{k}})})$ queries and operations.*

In particular, it follows that IND-CCA3 security of λ bits requires at least an expansion of $m - n \geq 3\lambda$ bits ($\mathcal{N}_{\text{rand.}} \geq 2\lambda$ and $\mathcal{N}_{\text{red.}} \geq \lambda$).

3 Local attacks and Haystack ciphers

In this section, we aim to establish a generic formal attack model encompassing a wide variety of attacks, based both on trace analysis and fault injections. The key feature of the covered attacks is *locality*: the attacks focus on a specific part of the white-box implementation, essentially treating the remaining parts as black boxes (oracles). Although our model hides many implementation details from an adversary, control over certain intermediate variables is extremely powerful, as evidenced by numerous attacks that can be mounted in this model. Therefore, securing implementations even in such a restricted model is still a very ambitious goal. The new generic model of local attacks motivates the modeling of concrete countermeasures as symmetric-key encryption schemes (the *haystack ciphers*), one of the key concepts in this paper.

3.1 Motivation

Consider the goal of formalizing the generic automated trace-based attacks from the literature (e.g., DCA [BHMT16], LDA [BU18,GPRW20], LPN [CU23], Filtering [CU24], etc.). These attacks analyze sliding windows over computational traces across multiple executions of the implementation (e.g., multiple plaintext encryptions). An obvious approach to model trace-based attacks is to give the adversary access to the computational traces without providing any structural/data-dependency information. Unfortunately, for typical fan-in-2 arithmetic circuits, this approach degenerates to the complete white-box setting. Indeed, an adversary may simply reconstruct the full circuit by a combinatorial attack of a small order. This can be done, for example, by guessing the positions of the two inputs and the output of a gate in the computational traces and testing the gate relation. The recovery can be further generalized and optimized by various techniques, but we leave this out of the scope of this work.

The discussion above leads us to the key idea of giving the adversary *unrestricted* access but only *to a small region* of the computational traces (the window in typical trace-based attacks). In order to stay in the restricted model, the region should not be too large as to include any essential data-dependency information. For example, observing both the inputs and the output of a gate reveals a potential link between these nodes, which can lead to improved attacks. At the same time, the region should contain sufficient information about its internal state so that the model can fairly evaluate countermeasures.

The general scenario can be formalized as follows. The region (or window) of computational traces observed by an adversary can be viewed as an *oracle* available to the adversary. The adversary may choose arbitrary inputs/plaintexts for the implementation and observe the intermediate computations falling in the region. Although many trace-based attacks are usually mounted in the known plaintext setting (e.g., DCA, LDA, etc.), there is no meaningful reason to restrict adversaries from choosing plaintexts arbitrarily and adaptively. In particular, some works showed how to exploit the CPA capability to improve the attacks [BU21,TGLZ23,CU24]; countermeasures such as linear/nonlinear masking or shuffling provide their intended security even in the CPA setup.

3.2 Local attack models

Formally, consider a white-box implementation of a function $C : \mathbb{F}_2^n \to \mathbb{F}_2^m : \boldsymbol{x} \mapsto \boldsymbol{y}$. We consider a vector of intermediate variables $\boldsymbol{s} \in \mathbb{F}_2^{n_s}$ which can be expressed as a function S of the input $\boldsymbol{x}$ (since in the white-box context all intermediate values are functions of the input). We will call S *the leakage function*. The vector $\boldsymbol{s} = S(\boldsymbol{x})$ can be arbitrarily augmented by additional intermediate values $\boldsymbol{r} = R(x) \in \mathbb{F}_2^{n_r}$ so that a pair $(\boldsymbol{r}, \boldsymbol{s})$ encompasses a full intermediate computational state of the implementation. In other words, there exist functions $R : \mathbb{F}_2^n \to \mathbb{F}_2^{n_r}$, $S : \mathbb{F}_2^n \to \mathbb{F}_2^{n_s}$, $Y : \mathbb{F}_2^{n_r} \times \mathbb{F}_2^{n_s} \to \mathbb{F}_2^m$ such that $C = Y \circ (R||S)$ (see Figure 1).

Here, R is only needed to complete the decomposition of the implementation formally and is not relevant for the attacks; the $(\boldsymbol{r}, \boldsymbol{s})$-decomposition of an

internal state also suggests that we are looking into intermediate variables of some implementation and not an arbitrary, unrelated leakage function. In this formalization of local trace-based attacks, the adversary is given oracle access to the function S and to the full function C (this allows us to consider selection functions built from the ciphertext). We emphasize that the underlying implementations of S/C are unavailable to the adversary. However, we do not restrict the adversary's actions when attacking the implementation using the given oracles.

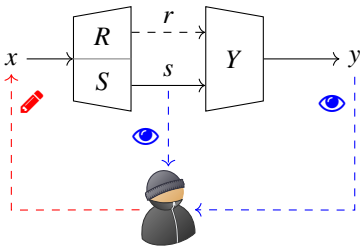

Fig. 1. Formal representation of a local trace-based attack *without* faults (Local Trace Model).

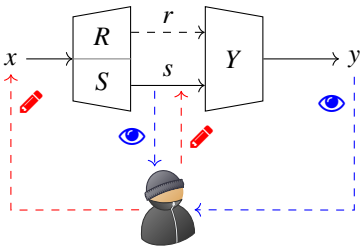

Fig. 2. Formal representation of a local trace-based attack *with* faults (Local Fault Model).

This viewpoint also allows us to incorporate many families of *fault injection* attacks in the model. For this purpose, we endow the adversary with the ability to *modify* the intermediate value vector $\boldsymbol{s}$ during computations and obtain the corresponding "faulty" output of the implementation. Equivalently, we provide the adversary with an *oracle* F mapping an input pair $(\boldsymbol{x}, \boldsymbol{s})$ to $Y(R(\boldsymbol{x}), \boldsymbol{s})$. This oracle allows one to inject any modification $\boldsymbol{s}'$ (a fault) into the intermediate vector $\boldsymbol{s}$, which is already available to the adversary, and obtain the corresponding faulty ciphertext $Y(R(\boldsymbol{x}), \boldsymbol{s}')$ (see Figure 2). As we will show later, this model encompasses various powerful fault attacks, including DFA, SIFA, and dummy computation elimination, among others.

Definition 5 (Local Trace/Fault Model). *Consider an implementation of a function* $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$. *Let* $R : \mathbb{F}_2^n \to \mathbb{F}_2^{n_r}$, $S : \mathbb{F}_2^n \to \times\mathbb{F}_2^{n_s}$, $Y : \mathbb{F}_2^{n_r} \times \mathbb{F}_2^{n_s} \to \mathbb{F}_2^m$ *be such that* $C = Y \circ (R||S)$.

In the local trace model*, the adversary aims to attack the implementation given only oracle (black-box) access to functions* S *and* C.

In the local fault model*, the adversary is given, in addition, access to the oracle* $F : \mathbb{F}_2^n \times \mathbb{F}_2^{n_s} \to \mathbb{F}_2^m : (\boldsymbol{x}, \boldsymbol{s}) \mapsto Y(R(\boldsymbol{x}), \boldsymbol{s})$.

On the choice of S The choice of the leakage function S is a *parameter* of the above definition. Some choices of S may lead to attacks, while others may not. For example, S can be set to include full implementation, and in this case, our model deteriorates to a pure white-box model (since it is no longer local).

Choices of smaller S but which include data dependencies may potentially leak relations between variables, which would then be too specific to implementation details.

Since our goal is to evaluate various countermeasures, we will primarily focus on leakage functions that contain protected (encoded) representations of a sensitive internal state. We will not consider internal wires of gadgets since most existing generic attacks apply directly to the encoded states. Similarly, we will not consider leakage of supporting components such as PRNG or fault detection; the primary focus is on secure intermediate computations.

3.3 Countermeasure model: Haystack cipher

In the following, we consider a countermeasure that represents an internal sensitive state in some protected manner. This protection may include protection against trace-based attacks, as well as fault injections, in the form of redundancy.

We start by defining countermeasures, which mainly need to specify their interface. It is the same as for the encryption schemes; the only difference would be in the security definitions.

Definition 6 (Countermeasure). *A* countermeasure *is a pair of algorithms* $(\mathrm{Encode}, \mathrm{Decode})$ *such that, for any key* $\boldsymbol{k}$*,*

1. Encode *is randomized and may depend on* $\boldsymbol{k}$*, and maps an* n*-bit input to an* s*-bit output;*
2. Decode *is deterministic and may depend on* $\boldsymbol{k}$*, and maps an* s*-bit output to an* n*-bit output or a symbol* $\perp$*;*
3. *for all* $\boldsymbol{x} \in \mathbb{F}_2^n$*,* $\Pr\left[\mathrm{Decode}(\mathrm{Encode}(\boldsymbol{x})) = \boldsymbol{x}\right] = 1$*.*

For an integer $b \geq 1$*, by* $(\mathrm{Encode}^{\otimes b}, \mathrm{Decode}^{\otimes b})$ *we denote the new countermeasure consisting of parallel* b *applications of the initial countermeasure (*Encode*,* Decode*). We refer to such countermeasures as* parallel*.*

Example 1. Common masking schemes (ISW, SEL, etc.) define parallel countermeasures since every bit of the sensitive values is encoded/decoded independently. Code-based masking, however, encodes multiple bits at once; yet, it can still be applied to multiple blocks in parallel.

Usual attacks on countermeasures rely on the following assumptions, which we further strengthen to *abstract away from details of a white-box implementation and the underlying cipher*. The strengthening gives the adversary more power (than may be available in a concrete cipher's case).

Full encoding leakage (shuffled) From the designer's viewpoint, it is desirable to maintain security even if full encoded information is present in the analyzed window (which corresponds to the leakage vector $\boldsymbol{s}$). For example, for masking schemes, it is common to assume that all the shares of the correct selection function are contained in the window. Let P denote the selection function being tested. Then, it is assumed that $S(\boldsymbol{x})$ contains in some positions (unknown

to the adversary) a full countermeasure-encoding of $P(\boldsymbol{x})$, that is, $\mathrm{Encode}(P(\boldsymbol{x}))$. This assumption is necessary, since for many countermeasures, missing a linear share can lead to an information-theoretically unrecoverable selection function. The usual source of cryptographic hardness here is the precise *positions* of shares inside the window, which are placed among other, not directly related intermediate variables. These variables can be interpreted as noise-providing or amplifying IND-CPA security.

Control over the selection function values The adversary aims to distinguish a correct selection function (e.g., corresponding to the correct guess of a part of a cipher's key) from an incorrect one. In most cases, the selection function consists of one bit (or, e.g., the Hamming weight of one byte), so they can effectively control the value of the (guessed) selection function by sampling random plaintexts and filtering out those having a wrong value. We simplify this assumption by allowing the adversary to directly control the selection function values being encoded, both in the correct and incorrect cases. Essentially, this means that we set the function $P(\boldsymbol{x})$ from the above discussion to the identity: $P(\boldsymbol{x}) = \boldsymbol{x}$. This simplification essentially leads us to the IND-CPA security notion, where an adversary has to distinguish encryptions of two chosen messages.

Fault injections as decryption queries Similarly to the previous idea, we will also assume that the output of the white-box implementation (as well as the fault oracle $(\boldsymbol{x}, \boldsymbol{s}) \mapsto (Y(R(\boldsymbol{x}), \boldsymbol{s}))$ from the local fault model) contains the unencoded internal values, i.e., the selection function. At first glance, this ruins the IND-CPA security: the adversary may trivially distinguish the two encryptions $S(\boldsymbol{m}_1)$, $S(\boldsymbol{m}_2)$ by observing the decrypted value $\boldsymbol{m}_b$ in the output of the implementation. However, we adhere to the idea of the IND-CCA oracle, which does not permit requesting decryption of the challenge ciphertexts. We emphasize that this restriction is simply a design element of the IND-CCA game; it does not constitute any additional assumption on the adversary's capabilities. This is similar to the challenge oracle in the IND-CPA game, which is just a part of the game and is not available to the adversary during an actual white-box attack. In other words, these standard security games are designed to test an adversary's ability to extract information from the encryption scheme, in the presence of encryption and decryption oracles.

Randomization of encodings Most known countermeasures rely on (pseudo)randomness during the encoding phase and the computations. It is modeled as true randomness in security proofs (for example, of masking schemes), whereas in the white-box setting, it must be derived pseudorandomly from the input. The secure implementation of such pseudorandomness generation is a challenging open problem that falls outside the scope of this work. Here, we will allow randomized queries for two reasons. First, this capability aligns well with typical attacks, where selection functions depend only on a portion of the input (for example, a single AES plaintext byte). The adversary may thus vary the remaining part of the input to obtain randomized encodings of the same values as the target part. Second, this capability aligns well with standard security no-

tions, both from the physical cryptanalysis world / masking schemes, as well as from symmetric-key encryption.

These discussions motivate the key concept of the paper - the *haystack ciphers*.

Definition 7 (Haystack cipher). *Let* (Encode, Decode) *be a countermeasure. Define the corresponding* Haystack cipher *as the encryption scheme* $(E_{\boldsymbol{k},r}, D_{\boldsymbol{k},r})$ *parametrized by a nonnegative integer* r *as follows:*

$$E_{\boldsymbol{k},r}(\boldsymbol{p}) = \text{Shuffle}_{\boldsymbol{k}}(\text{Encode}(\boldsymbol{p})||\$^r); \qquad D_{\boldsymbol{k},r}(\boldsymbol{c}) = \begin{bmatrix} (\boldsymbol{e}||\_) \leftarrow \text{Shuffle}_{\boldsymbol{k}}^{-1}(\boldsymbol{c}), \\ \textit{return } \text{Decode}(\boldsymbol{e}) \end{bmatrix}.$$

The key concept here is the *Hay randomness* $\$^r$ being statically shuffled with the output bits of the base encoding.

We recall that $\text{Shuffle}_{\boldsymbol{k}}$ denotes a fixed permutation of its input bits, which is chosen uniformly at random per each secret key $\boldsymbol{k}$. We emphasize that this shuffling is *static*, i.e., the permutation is independent of the input and is fixed once the secret key is fixed. This is different from dynamic shuffling (e.g., dummy shuffling), where a new permutation is generated *per each input*.

Remark 1. As we will show, the parameter r only makes sense in the CPA game. In the CCA games (when the decryption oracle is available to the adversary), the Hay randomness can be removed, thus r can be set to 0 without loss of security. This precisely corresponds to the pseudorandomness removal technique from Proposition 8 of Section 5.

Correspondence between white-box implementations and haystack ciphers Following the indistinguishability games of Subsection 2.4 and Section A, the symmetric cryptography concepts of a Haystack cipher have corresponding representations in white-box cryptography, as depicted in Table 1.

Table 1. Correspondence between symmetric and white-box cryptography terms in the haystack model.

Security	Symmetric-key cryptography	White-box cryptography
CPA	Plaintext Ciphertext	Selection function Trace window
CCA1	Decryption query Decryption failure ($\perp$)	Fault injection Fault detection ($\perp$)
CCA2	Relative forgery (Malleability)	Targeted fault injection
CCA3	Existential forgery attack	Undetected fault

A haystack cipher corresponds to the combination of the countermeasure encoding of the selection function and the difficulty of locating the encoding shares

among other computational nodes. Thus, the resulting haystack ciphertext represents a local observation in the traces. Similarly, access to the implementation's plaintext or ciphertext represents access to the selection function.

When attacking the haystack cipher in the CPA model, an attacker can query encryptions of chosen plaintexts, which represent the ability of an attacker to generate multiple traces in white-box cryptography.

In the CCA models, the attacker can ask for decryption queries, corresponding to retrieving the encoded value information from a fault injection. Assuming that an attacker can retrieve the underlying value being encoded from a fault injection is a strong assumption; thus, a scheme that resists a CCA attacker ensures security even if its encoded values are leaked. This assumption models the ability of a white-box adversary to infer information about the sensitive variables from faulty ciphertexts.

In the CCA2 model, the concept of a *relative forgery* represents the ability of an attacker to transform a ciphertext from the encryption oracle into a new one that is valid and whose corresponding plaintext is related to the original message predictably (this security requirement is also called *non-malleability*). In the white-box, this would correspond to performing faults on the ciphertext to end up with two different selection function values that have some desired relations. For instance, the DFA attack [BS97] requires faulting a precise part of the implementation and is inefficient otherwise: the goal is to introduce a fault that modifies only one particular sensitive bit.

Lastly, in the CCA3 model, it is possible to win the game if an attacker can perform an existential forgery attack, that is, create a new ciphertext that the decryption method will consider valid. In the white-box model, this exactly corresponds to the ability of an attacker to perform a fault that will be undetected by the cipher. Clearly, this does not guarantee that any useful information can be recovered from such a fault. Thus, CCA3 security of the haystack cipher is not strictly necessary. However, such forgeries can often leak information about the ciphertext's structure (equally, the countermeasure's encoding positions), and a countermeasure designer may wish to achieve this notion. In addition, classically, CCA3 is conceptually more straightforward to reach by combining an IND-CPA-secure scheme with an integrity protection (such as MACs).

Limitations This model is insufficient to prove the full security of an implementation. By design, it encompasses only local attacks and does not consider circuit analysis attacks or obfuscation countermeasures. Furthermore, we only consider the decoding function for the reasons mentioned above, while gadgets are essential for security.

Respectively, in [ISW03,BU21], the authors introduced Non-Interference and Algebraic Security, claiming that if a gadget respects these notions, it will be secure against DPA (and, therefore, DCA) and LDA attacks. In our model, a gadget could be CPA secure while not respecting these notions, as only the decoding function is studied. Similarly, including the whole gadget as a Haystack cipher would not be enough to assess its security. Indeed, while one can prove that a gadget is Non-Interferent and establish its CCA security, it does not nec-

essarily result in a Non-Interferent and secure CCA implementation [CPRR14], as opposed to the Strong Non-Interference notion [BBD+16].

4 Haystack ciphers in CPA model

In this section, we study basic properties of general Haystack ciphers, as well as define and analyze Haystack ciphers for white-box countermeasures from the literature.

4.1 On the plaintext size

We start with a proposition relating security of Haystack ciphers with different block sizes. Our main goal here is to motivate the analysis of 1-bit block schemes.

Proposition 4. *Let* $(\mathrm{Encode}, \mathrm{Decode})$ *be a countermeasure with* Encode *independent of the key. For any* $r \in \mathbb{Z}_{\geq 0}$, $b \in \mathbb{Z}_{>0}$, *let* $(E_{\boldsymbol{k},r,b}, D_{\boldsymbol{k},r,b})$ *be the Haystack cipher corresponding to the countermeasure* $(\mathrm{Encode}^{\otimes b}, \mathrm{Decode}^{\otimes b})$ *and* r *random bits (as in Definition 7). Then, for any* $b > 1$, *any IND-CPA adversary* A_b *against* $E_{\boldsymbol{k},r,b}$ *can be converted to an IND-CPA adversary* A_1 *against* $E_{\boldsymbol{k},r,1}$ *with the same IND-CPA advantage.*

Proof. We must construct the adversary for the 1-block Haystack cipher $E_{\boldsymbol{k},r,1}$. We will simulate the adversary A_b, and respond to its b-block queries by querying the target 1-block oracle with the first message block, extending the ciphertexts by concatenating encodings of the remaining message blocks, and reshuffling the full extended ciphertexts in a randomly chosen fixed order. Such simulated encryption is indistinguishable from a b-block encryption. More precisely, it is easy to see that the following two functions have equal distributions:

$$\mathrm{Shuffle}_{\boldsymbol{k}'}\Big(\mathrm{Shuffle}_{\boldsymbol{k}}(\mathrm{Encode}(\boldsymbol{m}_1)||\$^r) \,||\, \mathrm{Encode}(\boldsymbol{m}_2) \,||\, \ldots \,||\, \mathrm{Encode}(\boldsymbol{m}_n)\Big)$$
$$\mathrm{Shuffle}_{\boldsymbol{k}''}(\mathrm{Encode}(\boldsymbol{m}_1)||\mathrm{Encode}(\boldsymbol{m}_2)||\ldots||\mathrm{Encode}(\boldsymbol{m}_n)||\$^r),$$

where the three involved permutations are chosen uniformly at random. It follows that the IND-CPA game with $b = 1$ blocks can be solved with the same advantage as the given adversary for $b > 1$ blocks.

One could expect a reduction in the other direction. The idea would be to pad a 1-block query into a b-block query (arbitrarily). The irrelevant ciphertext blocks could be treated as noise. The problem is that this noise may have a different distribution than the bits of $\$^r$ expected by the 1-block adversary, potentially preventing them from performing their attack. In practice, non-uniformly distributed randomness would not significantly affect any attacks and may even help to classify and group ciphertext bits, reducing randomness and improving the attacks. Due to this argument, we will mainly focus on 1-block countermeasures (with block size $n = 1$ bit in most cases). In addition, we can also formally prove the converse for encodings whose output distribution is uniform.

Proposition 5. *If* $\mathrm{Encode}(\boldsymbol{x})$ *has a uniform distribution in* $\mathbb{F}_2^s$ *for uniformly distributed* $\boldsymbol{x} \in \mathbb{F}_2^n$*, then any adversary against* $E_{\boldsymbol{k},r,1}$*,* $r \geq b \cdot s$ *can be converted into an adversary against* $E_{\boldsymbol{k},r-b\cdot s,b}$*.*

Proof. Given a 1-block query of the adversary, we will pad it with blocks chosen uniformly at random, and query the b-block oracle, passing the result to the adversary. It is easy to see that the following two distributions are equal (due to the uniformity assumption):

$$\mathrm{Shuffle}_{\boldsymbol{k}}(\mathrm{Encode}(\boldsymbol{m})||\$^r)$$
$$\mathrm{Shuffle}_{\boldsymbol{k}}(\mathrm{Encode}(\boldsymbol{m}_1)||\mathrm{Encode}(\boldsymbol{t}_2)||\ldots||\mathrm{Encode}(\boldsymbol{t}_b)||\$^{r-b\cdot s}),$$

where each block $\boldsymbol{t}_i \in \mathbb{F}_2^n$ is chosen uniformly at random.

4.2 Haystack-ISW

In this section, we illustrate Haystack ciphers by using linear countermeasures for instantiation. All such ciphers are insecure due to being susceptible to a linear algebraic attack, essentially a direct reinterpretation of the trace-based LDA attack in the IND-CPA Haystack model. These examples serve the purpose of showing how Haystack ciphers work and how closely connected the attacks on them and the corresponding white-box countermeasures are.

Definition 8 (Haystack-ISW). *For a positive integer* $\ell > 1$*, define*

$$\mathrm{Encode}_\ell^{ISW} : \mathbb{F}_2 \to \mathbb{F}_2^\ell : p \mapsto (p \oplus t_2 \oplus \ldots \oplus \ldots t_\ell,\ t_2, t_3, \ldots, t_\ell),$$

where all t_i *are sampled independently and uniformly at random from* $\mathbb{F}_2$*. Furthermore, define the corresponding Haystack cipher with* r *random bits and* $m = \ell + r$ *ciphertext bits by Haystack-ISW*$_{\ell,r} = (E_{\boldsymbol{k},\ell,r}, D_{\boldsymbol{k},\ell,r})$*.*

An unpacked definition of the Haystack-ISW encryption is given by:

$$E_{\boldsymbol{k},\ell,r}(p) = \mathrm{Shuffle}_{\boldsymbol{k}}(p \oplus t_2 \oplus \ldots \oplus t_\ell,\ t_2, t_3, \ldots, t_\ell,\ \$_1, \ldots, \$_r).$$

It is essential to distinguish the encoding randomness $(t_2, \ldots, t_\ell)$ from the Hay randomness $\$^r = (\$_1, \ldots, \$_r)$: the former carries the information about the message p. At the same time, the latter is just noise aiming to hide the positions of shares after shuffling.

We will now illustrate the reinterpretation of the white-box LDA attack as an IND-CPA attack on the symmetric-key cipher Haystack-ISW. As mentioned in Subsection 2.4, it is sufficient to recover (an equivalent of) a decryption function; therefore, winning the CPA game is then trivial. The proof follows the idea of the LDA attack in the white-box model.

Proposition 6. *There exists an IND-CPA adversary against Haystack-ISW cipher* $(E_{\boldsymbol{k},\ell,r})$ *with time and query complexity* $O(m^\omega)$ *and negligible failure probability (note* $m = \ell + r$*).*

Proof. Let $\epsilon \in \mathbb{Z}_{>0}$ be a parameter and set $T = \ell + r + \epsilon$. Perform T queries with random inputs, obtaining ciphertexts $\boldsymbol{c}_i = E_{\boldsymbol{k},\ell,r}(p_i) \in \mathbb{F}_2^{\ell+r}$, $1 \leq i \leq T$. Let C be an $(\ell + r + \epsilon) \times (\ell + r)$ matrix with rows $(\boldsymbol{c}_i)_i$. Observe that the matrix C is uniformly random over $\mathbb{F}_2$ (over the choices of p_i and the encryption randomness). Thus, it has a nontrivial right kernel with negligible probability (exponentially small in ϵ). It follows that C has full rank $\ell+r$ with overwhelming probability. It follows that the weight-ℓ vector $\boldsymbol{z}$, which describes the position of the ℓ shares after shuffling, is the unique solution to the equation $C \times \boldsymbol{z} = \boldsymbol{p}$, which can be recovered using any linear algebra method. This vector enables efficient decryption and a straightforward victory in the IND-CPA game.

The definition and the attack extend straightforwardly to any linear encoding function over any finite field, including inner-product masking, polynomial masking, and code-based masking [WMCS20], which covers all previous examples. We show in Section B that a CPA attack can be mounted on Code-Based countermeasures similarly to the attack on ISW.

The SEL masking scheme [SEL21] leads to another example of a Haystack cipher. Due to several attacks [CU24] its practical relevance is significantly diminished. For completeness, we provide a detailed study of the attacks and their interpretation within the Haystack framework in Section C, as well as variations of the SEL masking scheme combined with ISW sharing.

4.3 Haystack-DS (Dummy Shuffling)

We now move on to white-box countermeasures based on dummy shuffling [BU21] and proceed to define and analyze the corresponding haystack ciphers.

Haystack ciphers from pure Dummy Shuffling Although shuffling is already at the heart of the haystack ciphers, it is only used *statically*: the resulting permutation of the elements only depends on the secret key k, and is fixed during the security games. Dummy shuffling, on the other hand, is *dynamic*: the shuffling permutation is randomly generated *for each execution* (this applies to the algebraic security and probing models, as well as to the haystack ciphers; in the real white-box setting, the order has to be derived pseudorandomly from the input). Its purpose is to introduce nonlinearity and prevent (higher-degree) algebraic attacks (HDDA in the white-box terminology [GPRW20]).

Definition 9 (Haystack-DS). *For integers $d \in \mathbb{Z}_{>0}$, define*

$$\text{Encode}_d^{DS} : \quad \mathbb{F}_2^n \to \mathbb{F}_2^{nd+\epsilon} : \quad \boldsymbol{p} \mapsto f \;||\; \text{Shuffle}_{\boldsymbol{f}}(\boldsymbol{p}, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_d),$$

where each $\boldsymbol{t}_i \in \mathbb{F}_2^n$ is sampled uniformly at random, $\boldsymbol{f} \in \mathbb{F}_2^\epsilon$ is the shuffling seed sampled uniformly at random. Furthermore, define Haystack-DS$_{d,r}$ $= (E_{\boldsymbol{k},d,r}, D_{\boldsymbol{k},d,r})$ as the corresponding Haystack cipher with r random bits.

Note that $\text{Shuffle}_{\boldsymbol{f}}$ shuffles whole n-bit blocks with a permutation of d elements, while $\text{Shuffle}_{\boldsymbol{k}}$ of the haystack cipher ciphers elements bitwise yet statically. This means that, for each dummy shuffling key $\boldsymbol{f}$, there exist d fixed lists of positions of ciphertext bits, each of length n, such that, for each ciphertext, one of the lists describes positions of the bits of $\boldsymbol{p}$.

Of course, the pure dummy shuffling countermeasure and the corresponding haystack cipher are insecure against correlation attacks. Each encoding bit has a correlation of $\frac{1}{2d}$ with the input, and this property is preserved after Haystack's static shuffling. The adversary has to test each ciphertext bit for the correlation. Therefore, it is suggested to use dummy shuffling together with the ISW countermeasure.

Shuffling information We include $\boldsymbol{f}$ in the output so that the encoding is in fact invertible. Since $\boldsymbol{f}$ contains sensitive information about the countermeasure, it has to be protected in some way, for example, encrypted (note that it does not take part in the computations, so that it only needs to be accessed at the encoding and decoding stages). However, this interesting question is outside the scope of our work, and we will ignore the presence of $\boldsymbol{f}$ in the ciphertext from now on.

5 Haystack ciphers in CCA model

While trace-based attacks are well-studied in the white-box setting, fault attacks, despite being very powerful, are less analyzed in the literature. In this section, building on the Haystack cipher CCA model presented in Subsection 3.3, we revisit existing fault attacks and propose new ones.

5.1 Haystack randomness removal

In [GPRW20], it was shown that pseudorandomness could be easily removed by using fault attacks. Indeed, faulting a node corresponding to the pseudorandomness of the cipher does not impact its output (unless it is protected, which should be reflected in the definition of the Haystack cipher). With enough traces, one can identify and then remove all random nodes from the cipher, in turn significantly weakening CPA countermeasures. We will now demonstrate that a CCA attacker can similarly remove the Hay randomness, and the complexity of doing so depends on the countermeasure employed.

Example 2. Let us consider the $\text{SEL}_{\ell,d}$ masking scheme with $d = 5$ non-linear shares and $\ell = 2$ linear shares, which will replace every bit variable of the cipher by $x = x_1 \oplus x_2 \oplus x_3 \cdot x_4 \cdot x_5 \cdot x_6 \cdot x_7$. Now, faulting the bit variable x_3 may not modify the original value x: it will modify it only if x_3 was equal to 0 before the fault, and if $x_4 = x_5 = x_6 = x_7 = 1$, which happens with probability $p = 1/2^5$. So, on average, 2^5 couple of traces and faulty traces are needed to distinguish a pseudorandom bit of the cipher from a nonlinear share of $\text{SEL}_{\ell,5}$.

We therefore introduce the burning value β of a Haystack cipher, which allows an attacker to burn the hay of a haystack to recover the needle. However, it does not guarantee the overall success of the attack.

Definition 10. *In the CCA models, we denote the average number of decryption queries required to distinguish any bit of a Haystack cipher H (in the worst-case) from Hay randomness by β, the burning value of H.*

Proposition 7. *Consider a countermeasure given by $Enc : \mathbb{F}_2^n \to \mathbb{F}_2^s$ and $Dec : \mathbb{F}_2^s \to \mathbb{F}_2^n \cup \{\perp\}$. Let $H = (E_{\boldsymbol{k},r}, D_{\boldsymbol{k},r})$ be the corresponding Haystack cipher.*

For a ciphertext $\boldsymbol{c}$, let us denote by $\boldsymbol{c}^{*i}$ its copy with the i^{th} bit flipped, and the function $\delta : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2, (\boldsymbol{a}, \boldsymbol{b}) \mapsto 1$ if $\boldsymbol{a}, \boldsymbol{b}, \perp$ are all distinct, and 0 otherwise. The burning value β of the above-defined haystack cipher H is upper bounded by:

$$\beta \leq \frac{2^s}{\min_{i \in \{1,\cdots,s\}} \left(\sum_{c \in \mathbb{F}_2^s} \delta(Dec(\boldsymbol{c}), Dec(\boldsymbol{c}^{*i})) \right)}$$

Remark 2. If there exist shares for which faulting has no impact on the Haystack decryption, then $\sum_{\boldsymbol{c} \in \mathbb{F}_2^m} \delta(\mathrm{Dec}(\boldsymbol{c}), \mathrm{Dec}(\boldsymbol{c}^{*i})) = 0$ and the formula is not defined. These shares need to be considered as Hay randomness.

Proof. For any index i representing the Hay randomness and for all ciphertexts $c \in \mathbb{F}_2^m$, we have $\delta(D_{\boldsymbol{k},r}(\boldsymbol{c}), D_{\boldsymbol{k},r}(\boldsymbol{c}^{*i})) = 0$, as faulting them does not impact the underlying decoding process Dec. For the shares that are not the Hay randomness, we can compute $p = 2^{-s}(\sum_{\boldsymbol{c} \in \mathbb{F}_2^s} \delta(\mathrm{Dec}(\boldsymbol{c}), \mathrm{Dec}(\boldsymbol{c}^{*i})))$ for a given share of index i, which gives us the probability of observing a difference in the plaintext by faulting this share once. The expected number of queries to observe a difference is therefore $1/p$. Now, since we are searching for the highest of these numbers amongst the shares, we can search for the minimum probability p over the shares of H, giving us the formula.

The burning parameter of ISW_ℓ [ISW03] is 1, since flipping one of its shares flips the output, while the burning parameter of BU [BU18] is $2^2 = 4$, since it has two non-linear shares. Similarly, the burning parameter of $\mathrm{SEL}_{\ell,d}$ [SEL21] is 2^d, with d indicating its degree, as explained in the previous Example 2. Additionally, the burning parameter of Dummy Shuffling[BU21] is d, where d represents the number of slots, as each slot has a $1/d$ chance of being the main one. Finally, the combination of dummy shuffling and ISW has the same burning parameter as Dummy Shuffling, and the combination of ISW and SEL has the same burning parameter as SEL, independently of the order of combination.

It is interesting to note that some attacks can still be effective even when some of the shares are removed. For instance, WBLPN [CU23] only needs to be performed on linear shares; thus, by using fault attacks, we could also remove the non-linear shares of SEL by faulting over a single query to perform the attack thereafter.

We will now prove that we can remove the bits of Hay randomness from a Haystack cipher, even against a CCA-secure scheme.

Proposition 8. *For a positive integer $s \in \mathbb{Z}_{>0}$, define $\mathit{Haystack}^{CCA}_{s,\boldsymbol{k}} : \mathbb{F}_2^n \to \mathbb{F}_2^s : \boldsymbol{p} \mapsto \mathit{Shuffle}_{\boldsymbol{k}}(\mathit{Encode}^{CCA}_s(\boldsymbol{p}))$, such that $\mathit{Haystack}^{CCA}_{s,\boldsymbol{k}}$ constitutes a CCA-secure scheme with no Hay randomness, with burning parameter β.*

Define $\mathit{Haystack}^{CCA+\$}_{s,r,\boldsymbol{k}} : \mathbb{F}_2^n \to \mathbb{F}_2^{s+r} : \boldsymbol{p} \mapsto \mathit{Shuffle}_{\boldsymbol{k}}(\mathit{Encode}^{CCA}_s(\boldsymbol{p}), \$_1, \cdots \$_r)$. In the CCA1 model, an attacker can remove the r bits of Hay randomness and perform forgery attacks in the CCA2 and CCA3 models with $O(\beta(s+r))$ queries and time complexity.

Proof. First, one can observe that $\mathrm{Haystack}^{\mathrm{CCA}}_{s,\boldsymbol{k}}$ and $\mathrm{Haystack}^{\mathrm{CCA+\$}}_{s,r,\boldsymbol{k}}$ share the same burning parameter β, as adding more Hay randomness does not change the computation of β. Let $\epsilon \in \mathbb{Z}_{>0}$ be a parameter. Perform $T = \beta + \epsilon$ queries with random inputs, obtaining T plaintext/ciphertext pairs. For each of the $\mathrm{Haystack}^{\mathrm{CCA+\$}}_{s,r,\boldsymbol{k}}$ output bits, perform a bit-flip fault and generate T queries for the same plaintexts, and observe whether the ciphertexts match. If so, the faulted bit is one of the Hay randomness, which results in a forgery attack on the CCA2 and CCA3 games, and can be ignored. After going through all the $s+r$ output bits of $\mathrm{Haystack}^{\mathrm{CCA+\$}}_{s,r,\boldsymbol{k}}$, all the r Hay random bits are removed, leaving only the s output bits of $\mathrm{Haystack}^{\mathrm{CCA}}_{s,\boldsymbol{k}}$.

Remark 3. Proposition 13 in Section D completes this proof by showing that corrective schemes do not prevent the removal of Hay randomness.

While removing the Hay randomness of any scheme can be done in $O(\beta(s+r))$, an attacker can adapt it to the countermeasure. Suppose we want to remove the Hay randomness of $\mathrm{SEL}_{2,5}$ from Example 2. Instead of trying to find faults in $\beta = 2^d$ traces, one can run the attack on one trace to remove all the linear shares first, and then run it over β traces.

If β is high, another trick would be to fault multiple nodes simultaneously over fewer than β traces, as each faulted node has a probability $1/\beta$ to create an error individually. Then, a divide-and-conquer approach can be used to find the shares if a difference is observed in the output.

Therefore, in the CCA model, since we can remove Hay randomness in $O(\beta(s+r))$ query and time complexity, and since a countermeasure needs to resist other attacks, we will not further study the burning value of Haystack ciphers. We will consider that the Hay randomness removal has been applied and will study Haystack without the presence of Hay randomness.

5.2 Multi-persistent fault attack against corrective schemes

Most fault attacks in the physical cryptanalysis literature consist of faulting a part of the state of the encryption algorithm to recover information about the secret key. It is the case of the differential fault attack (DFA) [BS97], where the author proposed to fault bits of the DES to observe its output to uncover key information, but also on more recent families of attacks [LSG+10,LRD+12,FJLT13].

For instance, to perform a DFA, an attacker needs to introduce a fault at a precise moment in the ciphering process, specifically before the last round in the

case of the AES [PQ03]. The attacker then observes the difference between its faulty and non-faulty output, giving them an oracle as to whether or not they succeeded in performing the fault: if the fault occurs too early in the ciphering process, the fault will impact the difference too much; and if too late, not enough.

These attacks are even easier to mount in the white-box model as the attacker has direct access to the software implementation and can choose with perfect precision where to apply a fault, and is not limited to the number of simultaneous faults. The white-box cryptanalysis tool JeanGrey[5] performs clusters of simultaneous faults in the implementation, discarding the whole cluster if the output is changed not enough, or performing sub-clusters of faults recursively if the output difference is too high[6]. This tool was alone sufficient to break some of the countermeasures proposed in the 2017 WhiBox contest [AT20].

Following Definition 4, an encoding function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^s$ has $s-n$ bits of redundancy, which ensures a pool of 2^{s-n} different encoded values on average for each input. Therefore, there exist 2^n undetected encoded values amongst the 2^s available ones, so an attacker trying to find a valid one by brute force needs to exhaust 2^{s-n-1} encoded values on average.

Having and preserving high redundancy against fault attacks is crucial to prevent an attacker from successfully finding another faulty encoded value that can't be detected by the scheme, thereby allowing effective fault attacks. Indeed, if an attacker removes completely the redundancy (makes s equal to n), the decoding function becomes a bijection, and all the encoded values are considered correct, removing entirely the detecting property of the scheme, weakening its security, and exposing it to free forgery attacks.

Once a fault is detected, the countermeasure outputs $\perp$. This symbol can either represent the *corrective* or the *infective* approach to prevent the fault attacker from recovering information from the faulty output. In the *infective* approach, the goal is to propagate the fault everywhere to create a seemingly random output. In contrast, in the *corrective* approach, the goal is to correct the fault so that the output appears unchanged. We will now show in Proposition 9 that an attacker in the CCA model can remove the bits of redundancy if a corrective approach is used.

Definition 11 (Haystack-CS$_f$). *Let n, $s > n$, and $f \leq s-n$ be positive integers, and let* $\mathrm{Encode}_{s,f} : \mathbb{F}_2^n \to \mathbb{F}_2^s$, $\boldsymbol{p} \mapsto \boldsymbol{x}$, *and* $\mathrm{Decode}_{s,f} : \mathbb{F}_2^s \to \mathbb{F}_2^n \cup \{\perp\}$, $\boldsymbol{x} \mapsto \boldsymbol{p}$ *be encoding and decoding functions of a detecting scheme, such that for any combination of f faults or less represented by a vector $\boldsymbol{e} \in \mathbb{F}_2^s$ of Hamming Weight at most f, for all $\boldsymbol{p} \in \mathbb{F}_n^2$, we have:*

$$\mathrm{Decode}_{s,f}(\mathrm{Encode}_{s,f}(\boldsymbol{p})) = \mathrm{Decode}_{s,f}(\mathrm{Encode}_{s,f}(\boldsymbol{p}) + \boldsymbol{e}) = \boldsymbol{p}.$$

Define Haystack-CS$_f$ $= (E_f, D_f)$ the corresponding Haystack cipher of this Correcting Scheme with no Hay randomness, with burning parameter β.

[5] https://github.com/SideChannelMarvels/JeanGrey

[6] https://blog.quarkslab.com/differential-fault-analysis-on-white-box-aes-implementations.html

Proposition 9 (Multi Persistent Fault Analysis (MPFA)). *In the CCA-1 model, it is possible to remove f bits of redundancy of Haystack-CS$_f$ in query and time complexity $O(\beta s)$.*

Proof. Let $\epsilon \in \mathbb{Z}_{>0}$ be a parameter. Perform $T = \beta + \epsilon$ queries with random inputs, obtaining T plaintext/ciphertext pairs $(\boldsymbol{p}_i, \boldsymbol{c}_i), 1 \leq i \leq T$. Let $j = 1$. For every ciphertext $\boldsymbol{c}_i, 1 \leq i \leq T$ perform a bit-flip fault on every index lesser than or equal to j, name the result $\boldsymbol{c}'_i$, and verify that $D_f(\boldsymbol{c}'_i) = \boldsymbol{p}_i$ for each i. If so, increment j by one and redo the previous step; if not, stop the algorithm. We can now choose randomly the t first ciphertext bits without impacting its decoded value, so we reduced the redundancy by t bits. Since we sampled T different random ciphertexts, the probability of finding T valid ones such that their first $t' > f$ bits are flipped is negligible (exponentially small in T), as there exist 2^{s-f} different valid ciphertexts that are corrected to the same plaintext, out of 2^s. Thus, we have $t = f$, and reduced the redundancy by f bits.

Remark 4. In practice, an attacker can perform persistent faults throughout the implementation nodes, realizing $\beta + \epsilon$ verifications per node to ensure if the fault succeeded or not. If not, perform a persistent fault on the next node; if so, undo the previous persistent fault and continue.

While MPFA alleviates the redundancy of correcting schemes, it does not address infective schemes. Indeed, while it is possible to distinguish a corrected output for the circuit from a faulted one, it is impossible to distinguish an infected output $\perp$ from a faulted one, which prevents determining whether a persistent must be performed or not. However, it is interesting to note that a corrective scheme could be employed if an infective approach is performed on top of it.

5.3 Ineffective Fault Attacks: Haystack forgery attack

The Ineffective Fault Attacks (IFA), firstly introduced in [Cla07] exploit information on the faults that do not impact the output. For instance, giving two bit variables a and b such that $a \cdot b$ is computed, faulting a to stuck it at 0 will generate a fault only when b is equal to 1 and when a is supposed to be equal to 1, which gives information. SIFA attack [DEK+18] expanded the attack by assessing the probability of success of a fault.

In white-box cryptography, the model guarantees fault success with bit precision, which allows for broader attacks. For instance, in [EGMC25], the authors demonstrated that it is possible to recover the locations of the ISW masking scheme shares by performing groups of two simultaneous faults: if the double fault does not impact the output of the implementation, then either the two bits are pseudorandom data or the two elements are shares of the same encoded value. Similarly, in [GPRW20], the authors showed that faults allow the retrieval of the main slot of a dummy shuffling countermeasure: if the fault of a share impacts the output, then, for this trace, the share is one of the main slots; if not, it is one of the dummy slots. Similarly, faulting a share of Haystack-SEL$_{\ell,d}$ from Definition 14 without Hay randomness yields two different outcomes: either the

fault is effective all the time and the share is linear, or the fault is ineffective with a probability of $1 - 1/2^d$ and the share is non-linear.

These ineffective faults create another state value, leading to a valid ciphertext. In the Haystack CCA model, this corresponds to a forgery attack: creating a new Haystack ciphertext that the decryption function will consider valid. Therefore, a CCA countermeasure must resist forgery attacks, which include SIFA.

Linear Fault Recoding (LFR): We show that there exists an attack on countermeasures based on linear codes. This attack, *Linear Fault Recoding* (LFR), as for LDA previously explained in the proof of Proposition 6, exploits the fact that an attacker can observe linear relations between the output elements of a haystack cipher. An attacker is not limited to bit-flip faults, and here we show that faulting the elements of the ciphertext following the found linear equations allows constructing valid codewords of the linear code countermeasure, removing its redundancy, even if it is combined with other schemes.

Definition 12 (Haystack-CCA-CB). *Let Haystack-CCA(E_{CCA}, D_{CCA}), with $E_{CCA} : \mathbb{F}_q^n \to \mathbb{F}_q^{m'}$, and $D_{CCA} : \mathbb{F}_q^{m'} \to \mathbb{F}_q^n \cup \{\perp\}$ be a secure Haystack cipher against any CCA attacker, such that sampling $m + \epsilon$ ciphertexts spans the ciphertext space.*

Furthermore, let Haystack-CB$_{\ell,G} = (E_{\boldsymbol{k},\ell,G}, D_{\boldsymbol{k},\ell,G})$ with $E_{\boldsymbol{k},\ell,G} : \mathbb{F}_q^{m'} \to \mathbb{F}_q^m$, and $D_{\boldsymbol{k},\ell,G} : \mathbb{F}_q^m \to \mathbb{F}_q^{m'} \cup \{\perp\}$ be a Code-Based haystack cipher following Definition 13, with no random bits and encoding function for a given matrix $G \in \mathbb{F}^{(m'+\ell)\times m}$, and with the corresponding decoding function $D_{\boldsymbol{k},\ell,G}$ returning the vector $\boldsymbol{p} \in \mathbb{F}_q^{m'}$ if there exists a vector $\boldsymbol{v} \in \mathbb{F}_q^{\ell}$ such that $\boldsymbol{x} = (\boldsymbol{p}||\boldsymbol{v})G$, $\perp$ otherwise.

Let Haystack-CCA-CB$_{\boldsymbol{k},\ell,G} = (E_{\boldsymbol{k},\ell,G}(E_{CCA}), D_{\boldsymbol{k},\ell,G}(D_{CCA}))$ be the composed Haystack cipher with:

$$\mathbb{F}_q^n \xrightarrow{E_{CCA}} \mathbb{F}_q^{m'} \xrightarrow{E_{\boldsymbol{k},\ell,G}} \mathbb{F}_q^m \quad \text{and} \quad \mathbb{F}_q^n \cup (\perp) \xleftarrow{D_{CCA}} \mathbb{F}_q^{m'} \cup (\perp) \xleftarrow{D_{\boldsymbol{k},\ell,G}} \mathbb{F}_q^m$$

Proposition 10. *In the CCA1 model, it is possible to remove the $m - m'$ bits of redundancy of Haystack-CB$_{\ell,G}$ from Haystack-CCA-CB$_{\ell,G}$ in query complexity $O(m)$ and time complexity $O(m^{\omega+1})$.*

Proof. Let $\epsilon \in \mathbb{Z}_{>0}$ be a parameter and set $T = m + \epsilon$. As for the proof of Proposition 6, perform T queries with random inputs, obtaining ciphertexts $\boldsymbol{c}_i = E_{\boldsymbol{k},\ell,G}(E_{\mathrm{CCA}}(\boldsymbol{c}_i)) \in \mathbb{F}_q^m$, $1 \le i \le T$. Let C be the $(m+\epsilon) \times (T)$ matrix with rows $(\boldsymbol{c}_i)_i$. Observe that the matrix C is uniformly random over $\mathbb{F}_q$. Thus, it has a non-trivial right kernel with negligible probability (exponentially small in ϵ). It follows that C has full rank m with overwhelming probability. Since the rows are made of vectors constructed with the linear expression $(E_{\mathrm{CCA}}(\boldsymbol{p}_i)||\$^{\ell})G$, given an index $i \in (1, \cdots, m)$, if $\boldsymbol{C}_i$, the i^{th} column of C is not linearly independent from the others, we can find the set of indices $S, i \notin S$ of columns $\boldsymbol{C}_j, j \in S$, such that $\boldsymbol{C}_i = \sum_{j \in S} \boldsymbol{C}_j$. Let $\boldsymbol{v}$ be a vector in $\mathbb{F}_q^s$ and let it satisfy $\boldsymbol{v} = \sum_{j \in S} \boldsymbol{v}_j$. The vector $\boldsymbol{v}$ follows the sets of conditions to be a valid codeword of Haystack-CB$_{\ell,G}$. Therefore, once we have determined the linear relations of the code-based countermeasure, to fault a share of index i, we replace it with the sum of

the shares of indexes in S, creating a valid codeword and an undetected fault. Similarly, we can find the family of m' vectors that spans the matrix C, which allows us to reduce the space of possible valid ciphertexts of Haystack-CCA-CB from q^m to $q^{m'}$.

Remark 5. Suppose only a linear code-based countermeasure has been employed. In that case, it is possible to mount a CCA1 forgery attack by taking one of the ciphertexts and faulting it until finding a ciphertext not previously queried. This attack is implemented in SageMath [SD25] for exposition [CU25].

While we effectively reduced the redundancy of the combined countermeasure by removing the redundancy brought by the Haystack-CB cipher, we did not break it totally, as the retrieved valid elements in $\mathbb{F}_q^{m'}$ are still mixed linearly. One can observe that the LFR attack does not require any knowledge of the underlying matrix G of the linear code-based countermeasure, so hiding it from an attacker does not prevent the attack. The assumption that the Haystack-CCA cipher spans the ciphertext space after $m + \epsilon$ queries ensures that the set of ciphertexts in $\mathbb{F}_q^{m'}$ creates a matrix that achieves full rank; otherwise, the LFR attack could make incorrect linear assumptions, as Haystack-CCA could induce some linear equations. We emphasize that the LFR attack is not limited to code-based countermeasures. For instance, it can break the time redundancy with the comparison proposed in [BCN+06], even if more than two duplicates of the scheme are made, since the decoding function is linear.

6 Conclusion

While the white-box community has thoroughly studied trace-based attacks and countermeasures (represented by the CPA attacks in our Haystack model), the formal study of fault attacks seems to lag behind (represented by CCA attacks), leaving no viable countermeasure in the white-box setting that would prevent an attacker from performing attacks that require no human effort. We therefore raise the need for further studies in fault attacks, specifically in fault countermeasure designs in the white-box model.

Acknowledgments This research was funded in part by the Luxembourg National Research Fund (FNR), by the project CryptoFin C22/IS/17415825, it is in the scope of grant reference NCER22/IS/16570468/NCER-FT, and in part by the project PQseal C24/IS/18978392.

References

AT20. Estuardo Alpirez Bock and Alexander Treff. Security assessment of white-box design submissions of the CHES 2017 CTF challenge. In Guido Marco Bertoni and Francesco Regazzoni, editors, *COSADE 2020*, volume 12244 of *LNCS*, pages 123–146. Springer, Cham, April 2020. doi:10.1007/978-3-030-68773-1_7. 8, 23

BBB+22. Anubhab Baksi, Shivam Bhasin, Jakub Breier, Dirmanto Jap, and Dhiman Saha. A survey on fault attacks on symmetric key cryptosystems. *ACM Comput. Surv.*, 55(4), November 2022. doi:10.1145/3530054. 7

BBD+16. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM Press, October 2016. doi:10.1145/2976749.2978427. 3, 17

BCN+06. Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proc. IEEE*, 94(2):370–382, 2006. doi:10.1109/JPROC.2005.862424. 8, 26

BDJR97. Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, October 1997. doi:10.1109/SFCS.1997.646128. 8

BGEC04. Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box AES implementation. In Helena Handschuh and Anwar Hasan, editors, *SAC 2004*, volume 3357 of *LNCS*, pages 227–240. Springer, Berlin, Heidelberg, August 2004. doi:10.1007/978-3-540-30564-4_16. 2

BHMT16. Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 215–236. Springer, Berlin, Heidelberg, August 2016. doi:10.1007/978-3-662-53140-2_11. 2, 4, 5, 11

BN00. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, Berlin, Heidelberg, December 2000. doi:10.1007/3-540-44448-3_41. 8

BRVW19. Andrey Bogdanov, Matthieu Rivain, Philip S. Vejre, and Junwei Wang. Higher-order DCA against standard side-channel countermeasures. In Ilia Polian and Marc Stöttinger, editors, *COSADE 2019*, volume 11421 of *LNCS*, pages 118–141. Springer, Cham, April 2019. doi:10.1007/978-3-030-16350-1_8. 6

BS97. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski, Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 513–525. Springer, Berlin, Heidelberg, August 1997. doi:10.1007/BFb0052259. 2, 7, 16, 22

BU18. Alex Biryukov and Aleksei Udovenko. Attacks and countermeasures for white-box designs. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 373–402. Springer, Cham, December 2018. doi:10.1007/978-3-030-03329-3_13. 2, 6, 7, 11, 21, 32

BU21. Alex Biryukov and Aleksei Udovenko. Dummy shuffling against algebraic attacks in white-box implementations. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 219–248. Springer, Cham, October 2021. doi:10.1007/978-3-030-77886-6_8. 2, 3, 7, 11, 16, 19, 21

CEJv02. Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In *Digital Rights Management Workshop*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002. doi:10.1007/978-3-540-44993-5_1. 2

CEJvO03. Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 250–270. Springer, Berlin, Heidelberg, August 2003. doi:10.1007/3-540-36492-7_17. 2

Cla07. Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 181–194. Springer, Berlin, Heidelberg, September 2007. doi:10.1007/978-3-540-74735-2_13. 7, 24

CPRR14. Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 410–424. Springer, Berlin, Heidelberg, March 2014. doi:10.1007/978-3-662-43933-3_21. 17

CU23. Alex Charlès and Aleksei Udovenko. LPN-based attacks in the white-box setting. *IACR TCHES*, 2023(4):318–343, 2023. doi:10.46586/tches.v2023.i4.318-343. 7, 11, 21, 33, 34

CU24. Alex Charlès and Aleksei Udovenko. White-box filtering attacks breaking SEL masking: from exponential to polynomial time. *IACR TCHES*, 2024(3):1–24, 2024. doi:10.46586/tches.v2024.i3.1-24. 5, 7, 11, 19, 33

CU25. Alex Charlès and Aleksei Udovenko. HaystackCiphers - supporting code, September 2025. doi:10.5281/zenodo.17078398. 3, 26

DEK+18. Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: Exploiting ineffective fault inductions on symmetric cryptography. *IACR TCHES*, 2018(3):547–572, 2018. URL: https://tches.iacr.org/index.php/TCHES/article/view/7286, doi:10.13154/tches.v2018.i3.547-572. 3, 7, 24

DES77. Data encryption standard. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce, January 1977. 2

DR98. Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. AES submission. See also http://csrc.nist.gov/archive/aes/rijndael/, 1998. 2, 4

DR02. Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Information Security and Cryptography. Springer-Verlag Berlin Heidelberg, 2002. doi:10.1007/978-3-662-04722-4. 4

DRP13. Yoni De Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the Xiao-Lai white-box AES implementation. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012*, volume 7707 of *LNCS*, pages 34–49. Springer, Berlin, Heidelberg, August 2013. doi:10.1007/978-3-642-35999-6_3. 2

EGMC25. Prouff Emmanuel, Renault Guenael, Rivain Matthieu, and O'Flynn Colin. *Embedded Cryptography*, chapter 3. Wiley, 2025. URL: https://www.wiley.com/en-us/Embedded+Cryptography+3-p-9781394351923. 8, 24

FJLT13. Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In Wieland Fischer and Jörn-Marc Schmidt, editors, *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 108–118. IEEE Computer Society, 2013. doi:10.1109/FDTC.2013.18. 7, 22

GPRW20. Louis Goubin, Pascal Paillier, Matthieu Rivain, and Junwei Wang. How to reveal the secrets of an obscure white-box implementation. *Journal of Cryptographic Engineering*, 10(1):49–66, April 2020. doi:10.1007/s13389-019-00207-5. 2, 3, 5, 6, 7, 8, 11, 19, 20, 24

GRW20. Louis Goubin, Matthieu Rivain, and Junwei Wang. Defeating state-of-the-art white-box countermeasures. *IACR TCHES*, 2020(3):454–482, 2020. URL: https://tches.iacr.org/index.php/TCHES/article/view/8597, doi:10.13154/tches.v2020.i3.454-482. 8, 34

ISW03. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Berlin, Heidelberg, August 2003. doi:10.1007/978-3-540-45146-4_27. 3, 6, 16, 21

KJJ99. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Berlin, Heidelberg, August 1999. doi:10.1007/3-540-48405-1_25. 1, 2, 6

Koc96. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Berlin, Heidelberg, August 1996. doi:10.1007/3-540-68697-5_9. 1

LRD+12. Ronan Lashermes, Guillaume Reymond, Jean-Max Dutertre, Jacques J. A. Fournier, Bruno Robisson, and Assia Tria. A DFA on AES based on the entropy of error distributions. In Guido Bertoni and Benedikt Gierlichs, editors, *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012*, pages 34–43. IEEE Computer Society, 2012. doi:10.1109/FDTC.2012.18. 7, 22

LRD+14. Tancrède Lepoint, Matthieu Rivain, Yoni De Mulder, Peter Roelse, and Bart Preneel. Two attacks on a white-box AES implementation. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 265–285. Springer, Berlin, Heidelberg, August 2014. doi:10.1007/978-3-662-43414-7_14. 2

LSG+10. Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta. Fault sensitivity analysis. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 320–334. Springer, Berlin, Heidelberg, August 2010. doi:10.1007/978-3-642-15031-9_22. 7, 22

PQ03. Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 77–88. Springer, Berlin, Heidelberg, September 2003. doi:10.1007/978-3-540-45238-6_7. 7, 23

RW19. Matthieu Rivain and Junwei Wang. Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. *IACR TCHES*, 2019(2):225–255, 2019. URL: https://tches.iacr.org/index.php/TCHES/article/view/7391, doi:10.13154/tches.v2019.i2.225-255. 6

SD25. The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 10.6)*, 2025. URL: https://www.sagemath.org. 3, 26

SEL21. Okan Seker, Thomas Eisenbarth, and Maciej Liskiewicz. A white-box masking scheme resisting computational and algebraic attacks. *IACR TCHES*,

2021(2):61–105, 2021. URL: https://tches.iacr.org/index.php/TCHES/article/view/8788, doi:10.46586/tches.v2021.i2.61-105. 2, 7, 19, 21, 32

Shr04. Tom Shrimpton. A characterization of authenticated-encryption as a form of chosen-ciphertext security. Cryptology ePrint Archive, Report 2004/272, 2004. URL: https://eprint.iacr.org/2004/272. 8

SMdH15. Eloi Sanfelix, Cristofaro Mune, and Job de Haas. Unboxing the white-box. Practical attacks against obfuscated ciphers. Black Hat Europe 2015, 2015. 2, 4, 8

Str69. Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, Aug 1969. doi:10.1007/BF02165411. 3

Teu25. Philippe Teuwen. Tools for white-box cryptanalysis. *Embedded Cryptography 3*, page 53, 2025. 4

TGCX23. Yufeng Tang, Zheng Gong, Jinhai Chen, and Nanjiang Xie. Higher-order DCA attacks on white-box implementations with masking and shuffling countermeasures. *IACR TCHES*, 2023(1):369–400, 2023. doi:10.46586/tches.v2023.i1.369-400. 2, 5

TGLZ23. Yufeng Tang, Zheng Gong, Bin Li, and Liangju Zhao. Revisiting the computation analysis against internal encodings in white-box implementations. *IACR TCHES*, 2023(4):493–522, 2023. doi:10.46586/tches.v2023.i4.493-522. 11

WMCS20. Weijia Wang, Pierrick Méaux, Gaëtan Cassiers, and François-Xavier Standaert. Efficient and private computations with code-based masking. *IACR TCHES*, 2020(2):128–171, 2020. URL: https://tches.iacr.org/index.php/TCHES/article/view/8547, doi:10.13154/tches.v2020.i2.128-171. 3, 8, 19

A Security games for symmetric-key encryption

▷ (IND-CPA)
proc Initialize()
$k \xleftarrow{\$} K();\ b \xleftarrow{\$} (0,1)$

proc Enc(p)
$c \xleftarrow{\$} E_k(p)$; **return** c

proc LR(p_0, p_1)
$c \xleftarrow{\$} E_k(p_b)$; **return** c

proc Finalize(b')
If $(b' = b)$ **return** Win
else **return** Loose

▷ (IND-CCA1 / non-adaptive)
proc Initialize()
$k \xleftarrow{\$} K();\ b \xleftarrow{\$} (0,1);\ f \leftarrow 0$

proc Enc(p)
$c \xleftarrow{\$} E_k(p)$; **return** c

proc Dec(c)
If $f = 0$ then **return** $D_k(c)$
else **return** $\perp$

proc LR(p_0, p_1)
$c \xleftarrow{\$} E_k(p_b);\ f \leftarrow 1$; **return** c

proc Finalize(b)
If $(b' = b)$ **return** Win
else **return** Loose

▷ (IND-CCA2 / adaptive)
proc Initialize()
$k \xleftarrow{\$} K();\ b \xleftarrow{\$} (0,1);\ S \leftarrow \emptyset$

proc Enc(p)
$c \xleftarrow{\$} E_k(p)$; **return** c

proc LR(p_0, p_1)
$c \xleftarrow{\$} E_k(p_b);\ S \leftarrow S \cup (c)$; **return** c

proc Dec(c)
If $c \notin S$ then **return** $D_k(c)$
else **return** $\perp$

proc Finalize(b)
If $(b' = b)$ **return** Win
else **return** Loose

▷ (IND-CCA3 / authenticated)
proc Initialize()
$k \xleftarrow{\$} K();\ b \xleftarrow{\$} (0,1);\ S \leftarrow \emptyset$

proc Enc(p)
$c \xleftarrow{\$} E_k(p);\ S \leftarrow S \cup (c)$; **return** c

proc LR(p_0, p_1)
$c \xleftarrow{\$} E_k(p_b);\ S \leftarrow S \cup (c)$; **return** c

proc Forge(c)
If $c \notin S$ and $D_k(c) \neq \perp$ **return** Win

proc Finalize(b)
If $(b' = b)$ **return** Win
else **return** Loose

B CPA attack on Code-Based countermeasure

Following Subsection 4.3, we demonstrate here that a CPA attacker can break a code-based Haystack cipher using an attack derived from LDA.

Definition 13 (Haystack-CB). *Let $\mathbb{F}$ be a finite field, $\ell \in \mathbb{Z}_{>0}$ and $G \in \mathbb{F}^{(n+\ell)\times s}$ be a right-invertible matrix (in particular, $s \geq n+\ell$). Define an encoding function*

$$\mathrm{Encode}^{CB}_{\ell,G} : \mathbb{F}^n \to \mathbb{F}^s : \boldsymbol{p} \mapsto (\boldsymbol{p}||\$^\ell) \times G,$$

where $\$^\ell$ is sampled uniformly at random from $\mathbb{F}^\ell$.

Furthermore, define Haystack-$CB_{\ell,G} = (E_{k,\ell,G,r}, D_{k,\ell,G,r})$ as the corresponding Haystack cipher with r random bits.

Remark 6. In principle, the Hay randomness $\$^r$ can be incorporated into the definition of G by increasing ℓ and passing the added random bits to the output without mixing them into the generator matrix G.

Proposition 11. *There exists an IND-CPA adversary against Haystack-CB ($E_{k,\ell,G,r}$) with time and query complexity $O(m^\omega)$ and a negligible failure probability (note that $m = s + r$).*

Proof. The proof is similar to the Proposition 6. Although the corresponding matrix C is not entirely random and not full rank due to possible redundancies introduced by G, it is easy to show that any solution to the equation $C \times \boldsymbol{z} = \boldsymbol{p}^{(j)}$ yields a decryption function with overwhelming probability. Here, $\boldsymbol{p}^{(j)}$ is a vector corresponding to the j-th coordinate of the messages.

C CPA attack on SEL masking scheme

We give here a thorough study of the SEL Haytsack cipher and its combinations with ISW, as discussed in **??**.

Haystack ciphers from pure SEL scheme We now proceed to the case of nonlinear masking, focusing on the scheme presented in [SEL21], which includes [BU18] as a special case. Its decoding function consists of one nonlinear monomial of degree d summed with ℓ linear shares. We note that [SEL21] only proposed gadgets for degrees 2 and 3, indicating that the instantiation of the scheme for higher degrees remains an open problem. Nonetheless, we analyze the scheme in full generality since our current study does not involve gadgets.

Definition 14 (Haystack-SEL). *For integers $\ell, d \in \mathbb{Z}_{>0}$, define*

$$\begin{aligned}\mathrm{Encode}^{SEL}_{\ell,d} : \mathbb{F}_2 \to \mathbb{F}_2^{\ell+d} :\\ p \mapsto (p \oplus \$_1\$_2 \ldots \$_d \oplus \$'_2 \oplus \ldots \oplus \$'_\ell,\quad \$'_2, \ldots, \$'_\ell,\quad \$_1, \$_2, \ldots, \$_d),\end{aligned} \tag{1}$$

where all $\$_i, \$'_j$ are sampled independently and uniformly at random from $\mathbb{F}_2$. Furthermore, define Haystack-$SEL_{\ell,d,r} = (E_{k,\ell,d,r}, D_{k,\ell,d,r})$ as the corresponding Haystack cipher with r random bits.

Proposition 12. *There exists an IND-CPA adversary against Haystack-SEL ($E_{k,\ell,d,r}$) with time and query complexity $O((\ell+d+r)^{\omega+1})$ and negligible failure probability.*

Proof. We will reapply the filtered LDA attack [CU24] in the Haystack cipher model. Let $\epsilon \in \mathbb{Z}_{>0}$ be a parameter and set $N = 3(\ell + d + r + \epsilon)$. Perform N queries with random inputs, obtaining ciphertexts $\boldsymbol{c}_i = E_{k,\ell,r}(m_i) \in \mathbb{F}_2^{\ell+d+r}$, $1 \le i \le N$. For each output bit position j, $1 \le j \le \ell + d + r$, perform the following attack. Let $\hat{S}$ denote the subset of ciphertexts for which the j-th bit equals 0. Let $\hat{C}$ be an $(\ell+r+\epsilon)\times(\ell+r)$ matrix with rows being ciphertexts from $\hat{S}$, and let $\hat{m}$ be the vector of plaintexts corresponding to the chosen ciphertexts. Then, apply the linear algebraic attack from Proposition 6 and Proposition 11, by solving the equation $\hat{C} \times \boldsymbol{z} = \hat{m}$. In the case that c_j corresponds to one of the nonlinear shares $\$_1, \ldots, \$_d$, the nonlinear monomial will be equal to 0, and we will essentially have an ISW-sharing of the message among unrelated random bits. The chances of having less than $\ell + d + r + \epsilon$ selected ciphertexts for each position j are exponentially small in ϵ; the same can be shown for the chances of having a non-unique solution. Therefore, with overwhelming probability, the attack succeeds and recovers the positions of linear shares and the positions of nonlinear shares. This allows the decryption of ciphertexts in 1/2 of the cases (when $\boldsymbol{c}_j = 0$), which is sufficient for breaking CPA security.

Haystack ciphers from SEL-then-ISW combination First, we study the composed countermeasure where a SEL-encoded input is further encoded with ISW. For simplicity, we set the number of linear shares in SEL to 1, since it is already expanded to more linear shares by ISW, and there is no clear benefit of having a larger number of them. This means that we have no $\$'$ shares in Definition 14.

Definition 15 (Haystack-SEL-ISW). *For positive integers $\ell, d \in \mathbb{Z}_{>0}$, define*

$$\mathrm{Encode}_{\ell,d}^{SEL\text{-}ISW} : \mathbb{F}_2 \to \mathbb{F}_2^{(d+1)\ell'} : p \mapsto [\mathrm{Encode}_{\ell}^{ISW}]^{\otimes(d+1)}(\mathrm{Encode}_{1,d}^{SEL}(p)),$$

Furthermore, define Haystack-SEL-ISW$_{\ell,d,r}$ $= (E_{k,\ell,d,r}, D_{k,\ell,d,r})$ as the corresponding Haystack cipher with r random bits.

The decoding function of SEL-ISW has the shape of the SEL decoding function, but with each variable independently replaced by ℓ' linear shares:

$$p = \mathrm{Decode}_{\ell,d}^{\text{SEL-ISW}}(x) = x'_1 + \ldots + x'_\ell + \\ (x_{1,1} + \ldots + x_{1,\ell}) \cdot (x_{2,1} + \ldots + x_{2,\ell}) \cdot \ldots \cdot (x_{d,1} + \ldots + x_{d,\ell}). \quad (2)$$

We now survey the main known attacks against this countermeasure from [CU23,CU24]. We recall that, in the corresponding Haystack cipher, the positions of the shares are unknown to the adversary. Two basic attacks against this cipher are based on guessing the locations of some shares. First, by guessing ℓ

shares $(x_{i,1}, \ldots, x_{i,\ell})$ of one of the nonlinear factors $(x_{i,1} + \ldots + x_{i,\ell})$, one can run the filtering attack. This is essentially a variant of the "higher-order filtering attack". A similar option is to guess the positions of the ℓ linear shares $(x'_1, \ldots, x'_\ell)$, compute their sum, and correlate with the input. This is the "higher-order differential computation analysis" attack [GRW20]. Both attacks have complexity dominated by guessing the locations of shares and a small factor of extra work. This requires time $\Omega(\binom{m}{\ell}) = \Omega(m^\ell)$ for $m \gg \ell$. Another general attack is the "higher-degree decoding attack", a generalization of LDA, which is based on linearization and runs in time $O(m^{\omega d})$.

A more powerful attack is the LPN method, which exploits the key algebraic weakness of the scheme: the monomial has a strong tendency to be equal to zero (probability $1 - 2^{-d}$), so that the scheme degenerates to a linear masking on most inputs. The tendency becomes stronger for higher d. However, the attack is exponential in the size of the ciphertext m. Using complexity estimate $O\left(\frac{m^{\omega-1}}{(1-2^{-d})^m}\right)$ from [CU23] with $\omega = 2.8$, one needs $m \geq 600$ to achieve 128-bit security for $d = 3$, and the number roughly doubles with each increment of d.

Due to the absence of known gadgets and strong attacks on the scheme/cipher, we do not explore the possibilities of trade-offs in parameters and focus on more relevant schemes.

Haystack ciphers from ISW-then-SEL combination Now, we will first apply the ISW encoding and then split each linear share into a SEL-sharing.

Definition 16 (Haystack-ISW-SEL). *For positive integers $\ell, d, \ell' \in \mathbb{Z}_{>0}$, define*

$$\mathrm{Encode}^{\mathit{ISW\text{-}SEL}}_{\ell,d,\ell'} : \mathbb{F}_2 \to \mathbb{F}_2^{(d+\ell)\ell'} : p \mapsto [\mathrm{Encode}^{\mathit{SEL}}_{\ell,d}]^{\otimes \ell'}(\mathrm{Encode}^{\mathit{ISW}}_{\ell'}(p)),$$

Furthermore, define Haystack-ISW-SEL$_{\ell,d,\ell',r}$ $= (E_{k,\ell,d,\ell',r}, D_{k,\ell,d,\ell',r})$ as the corresponding Haystack cipher with r random bits.

The decoding function of ISW-SEL has the shape of a sum of ℓ' instances of the SEL decoding function:

$$\begin{aligned} p = \mathrm{Decode}^{\text{ISW-SEL}}_{\ell,d,\ell'}(x) = x_{1,1} \cdot \ldots \cdot x_{1,d} \ + \ x'_{1,1} + \ldots + x'_{1,\ell} \\ \vdots \\ + \ x_{\ell',1} \cdot \ldots \cdot x_{\ell',d} \ + \ x'_{\ell',1} + \ldots + x'_{\ell',\ell}. \end{aligned}$$

This cipher seems to be much more resistant to various attacks. First, one needs to find all $\ell'\ell$ linear shares to perform a correlation attack (HODCA). Filtered LDA requires filtering at least one variable for each nonlinear monomial, that is, positions of ℓ' variables need to be guessed (with some slack given by the fact that we can use any of the d variables for each monomial). HDDA still takes the same amount of time: $O(m^{\omega d})$. The LPN noise of the construction increases linearly in ℓ' from 2^{-d} to $\frac{1}{2}(1-(1-2^{1-d})^{\ell'}) = \ell' 2^{-d} + O((\ell')^2)$ ([CU23, Prop.4]), making LPN less performant. In particular, setting $\ell' = 2^d$ maintains

the noise nearly constant (lower bounded by 0.43), although it is costly for large values of d. In this setup, for λ-bit security against HDDA, one could set $d = \sqrt{\lambda}, \ell' = 2^{\sqrt{\lambda}}, m = 2^{\frac{\sqrt{\lambda}}{\omega}}$ so that $O(m^{\omega d}) = O(2^{\lambda})$ and the LPN noise is constant. Again, the ciphertext size m is superpolynomial in λ, rendering the scheme not asymptotically efficient, and we do not even consider the complexity of the gadgets.

We conclude that SEL and its combinations with ISW do not yield efficient masking schemes or haystack ciphers.

D Haystack randomness removal in the presence of a correcting scheme

In Proposition 8 of Section 5, we showed that an attacker could remove the Haystack random bits of a Haystack cipher, as he knows if a fault he injected is successful by observing a non-null difference in the faulty / non-faulty Haystack ciphertexts. However, if a countermeasure detects and corrects the fault, the attacker cannot observe such a difference. In the following, we will show that an attacker can still succeed in removing the Hay randomness by performing an attack similar to the Multi-Persistent Fault Attack presented as a proof of Proposition 9.

Definition 17 (Haystack-CS$^{\$}$). *Consider the error correcting Haystack-CS$_f$ cipher defined in Definition 11 which can correct up to f faults. Define $E^{\$}_{f,k,r}$ and $D^{\$}_{f,k,r}$ as the previous decoding and encoding functions in the presence of Haystack random bits, such that:*

$$E^{\$}_{f,k,r} = \mathit{Shuffle}_k(E_f, \$_1, \cdots, \$_r) \text{ and } D^{\$}_{f,k,r} = \mathit{Unshuffle}_k(D_f),$$

and define Haystack-CS$^{\$}_{f,k,r} = (E^{\$}_{f,k,r}, D^{\$}_{f,k,r})$.

Proposition 13. *In the CCA-1 model, an attacker can remove the Hay random bits even against a correcting scheme in $O(\beta(s+r))$ queries and time complexity.*

Proof. Let $\epsilon \in \mathbb{Z}_{>0}$ be a parameter. Perform $T = \beta + \epsilon$ queries with random inputs, obtaining ϵ plaintext/ciphertext pairs $(\boldsymbol{p}_i, \boldsymbol{c}_i), 1 \leq i \leq T$. As for the previous proof, using MPFA, perform persistent faults and ask for queries using the same plaintexts $\boldsymbol{p}_i$ until the ciphertext does not match. Let us denote the number of persistent faults necessary to fault the ciphertext consistently by n. Since there are r Hay random bits, the persistent fault might also be on Hay bits, so $n \geq f$. Keep $n-1$ of the previous persistent faults, and, using the proof of Proposition 8, for each of the remaining $s+r-n$ output bits of the Haystack cipher, we fault it and generate $\beta + \epsilon$ queries using the same plaintexts p_i to observe if the ciphertexts are matching $\boldsymbol{c}_i$. If the ciphertexts match, the faulted bit is a Hay randomness and can be ignored; otherwise, it is one of the shares of Haystack-CS$_f$. We can now perform persistent faults on these $m \leq n$ newly found shares, and remove a set of m persistent faults of those of the first step,

to fault them individually to uncover new shares or remove more Hay random bits. Repeat until removing all the Hay randomness of Haystack-CS$^{\$}_{f,k,r}$.