Proving SIFA Protection of Masked Redundant Circuits [⋆]

Vedad Hadžić, Robert Primas, and Roderick Bloem

Graz University of Technology, Graz, Austria
`firstname.lastname@iaik.tugraz.at`
https://www.iaik.tugraz.at/

Abstract. Implementation attacks like side-channel and fault attacks pose a considerable threat to cryptographic devices that are physically accessible by an attacker. As a consequence, devices like smart cards implement corresponding countermeasures like redundant computation and masking. Recently, statistically ineffective fault attacks (SIFA) were shown to be able to circumvent these classical countermeasure techniques. We present a new approach for verifying the SIFA protection of arbitrary masked implementations in both hardware and software. The proposed method uses Boolean dependency analysis, factorization, and known properties of masked computations to show whether the fault detection mechanism of redundant masked circuits can leak information about the processed secret values. We implemented this new method in a tool called *Danira*, which can show the SIFA resistance of cryptographic implementations like AES S-Boxes within minutes.

1 Introduction

Cryptographic primitives are primarily designed to withstand mathematical attacks in a black-box setting. However, when these primitives are deployed in the real world, they find themselves in a grey-box setting in which an attacker may try to force faulty computations or observe additional physical side-channel information, such as instantaneous power consumption. This improved attacker capability simplifies the extraction of secrets like cryptographic keys.

Active implementation attacks, such as fault analysis [9,7], and passive side-channel attacks, like power or electromagnetic (EM) analysis [26,27], are among the most serious threats for implementations of cryptographic algorithms. A common algorithmic countermeasure strategy against these attacks is the combination of masking against power analysis with redundant computation against fault attacks. *Masking* is a secret-sharing technique where one splits a cryptographic computation into $d + 1$ random shares. This technique ensures that the observation of up to d intermediate values of that masked computation does not

[⋆] This work was supported by the *Austrian Research Promotion Agency (FFG)* via the **FERMION** project (grant number 867542, ICT of the Future), and the K-project **DeSSnet** (funded in the context of COMET). We refer to the online appendix [22] for the formal proofs of the presented lemmas and theorems.

reveal any information about native (unmasked) values [24,20,21,6,15]. *Redundant computation* tries to prevent the release of faulty cryptographic computations caused by environmental influences or malicious tampering such as voltage glitches, lasers, or rapid temperature variations. Without this countermeasure, an attacker with access to faulty computations can learn information about the used cryptographic key in many different ways [3,23,16].

Researchers long believed that the combination of redundancy and masking could adequately deal with active and passive implementation attacks. However, it was recently shown that when using *statistical ineffective fault attacks* (SIFA), even such protected cryptographic implementations are vulnerable to rather straightforward implementation attacks [13,12,14]. The key observation behind SIFA attacks is that a cryptographic key may correlate with the suppression of a faulted cryptographic computation. Thus, the attacker can obtain information about this key by observing whether the output of a faulted cryptographic computation is suppressed by a redundancy countermeasure or not.

For example, if a 1-bit signal carries a secret value, and the attacker can force this signal to zero, they can learn the secret value by observing whether or not this fault is detected. While this simplified example is obvious, SIFA is interesting because it works even if the fault injection targets just one share of a masked secret. In fact, SIFA is exploitable even if the attacker does not know the exact effect of a fault injection on the faulted value [12].

Most proposed mitigation techniques against SIFA so far use error correction, which is however costly when combined with masking [11,29]. Another recently proposed SIFA mitigation tries to solve this issue with a careful combination of redundancy, masking, and reversible computing [10], achieving protection against SIFA without significant overheads. The authors give detailed *circuit* descriptions of protected cipher components that can be mapped into concrete software or hardware implementations. However, even minor modifications of the circuit description due to human error, compilers, or synthesis tools, although preserving functional equivalence, may make the circuit vulnerable to SIFA. Consequently, there is a high demand for tooling that can support designers in building efficient cryptographic implementations resistant against power analysis and fault attacks, including SIFA.

1.1 Related Work

The empirical and formal verification of power analysis and fault attack countermeasures is an already well established topic in the cryptographic research community [1,4,8,19,17,18,25]. On a conceptual level, the verification of masking countermeasures — ensuring that individual computations are unrelated to any cryptographic secret — does perform statistical independence checks that could also be adapted for verifying SIFA protection, i.e., that cryptographic secrets do not correlate with the suppression of a faulted cryptographic computation. However, in the following we argue that such existing tools either cannot be easily adapted for SIFA verification, or would come with performance overheads that make them unattractive for practical use.

Tools like Rebecca [8] and its successor Coco [19] use correlation tracking to show statistical independence in (sequential) masked hardware circuits. Although their method ignores the *strength* and *sign* of correlations for performance reasons, the remaining information is still sufficient to show standard probing resistance of masked circuits. However, these approximations are not applicable for SIFA verification. Since Rebecca and Coco do not track the *sign* of correlations, there is no way to distinguish the correlation sets of a negated value from a non-negated value. Due to the nature of bit-flip faults, this method leads to falsely reported leaks due to the structure of the fault-detection mechanism. Similarly, tools like `maskVerif` [4] rely on security proofs for a gate's input signals to prove the gate's security. According to our investigation, since the fault-detection mechanism combines the shares in its sub-formulas, a leakage report is triggered even though the value cannot be observed.

Exact methods like SILVER [25] use some form of model counting to track exact probability distributions of values within masked circuits and check whether the correlation strength is zero for all secret values. These methods could be adopted for SIFA verification, e.g., by using a strategy as outlined in Figure 1 but will lead to verification runtimes significantly higher compared to the approach that we will present in this paper.

Besides masking verification tools, there also exists VerFI [2], a verification tool dedicated to fault attacks that, amongst others, does have the capability to verify SIFA protection of a given circuit in certain scenarios. More precisely, VerFI can detect SIFA vulnerability of a given circuits using an empirical and simulation-based approach that essentially checks if either (1) all fault injections are being corrected through error correction methods, or (2) all fault injections are being detected via redundancy methods. This empirical approach can be used for error-correction-based SIFA countermeasures, however, VerFI is not suited for the verification of, e.g., the more efficient SIFA countermeasure design by Daemen et al.[10] that does not need be able to correct any possible fault injection.

1.2 Contribution

The contribution of this paper is threefold and consists of a method and its implementation, its evaluation, and resulting SIFA-resistant circuit artifacts.

Method. We present a formal verification approach to determine whether a masked redundant cipher implementation is SIFA resistant within a well-defined attacker model. Our verification approach checks whether the output of the fault-detection mechanism correlates with secrets used in the computation. We present three properties and their respective checking methods that serve as sufficient conditions for SIFA protection. *(Incompleteness):* If a function δ does not functionally depend on all shares of a secret s, it cannot leak the secret. *(Hiding):* If a function δ can be written as $m \oplus \delta'$, where m is a uniformly distributed random variable and δ' is functionally independent of m, δ does not leak information about any secrets. *(Inferred independence):* For a function $\delta = \bigvee_i \delta_i$, if all linear combinations of its partial functions δ_i are statistically independent of a secret s, δ cannot leak the secret s. We present an algorithm that uses these sufficient but not necessary conditions to prove the security of circuits. Our tool *Danira* implements this algorithm and is, to our knowledge, the first tool for formal verification of SIFA resistance of masked redundant circuits.

Evaluation. We provide an experimental evaluation of our method. Because the sufficient conditions may not be able to prove SIFA resistance, we show in our experimental section that the approach gives precise results for a representative range of secure circuits. If *Danira* cannot prove resistance, it provides fault locations that might leak information about the secrets. We show that *Danira* can accurately prove security or find bugs in S-Boxes, the non-linear parts of cryptographic implementations, in minutes or even seconds. With respect to SIFA verification, masked linear layers do not need any further analysis as fault injections in these components are not exploitable with SIFA. Ultimately, we give practical examples illustrating that, even when a design is secure against SIFA on paper, vulnerabilities may arise as a result of simple compiler/synthesis optimizations, which can then however be identified with *Danira*.

Artifacts. As a direct result of this work, we present the first SIFA-resistant Verilog implementations of Daemen et al.[10] designs for a masked AES S-Box, the KECCAK χ_3 S-Box, and all classes of quadratic 4-bit S-Boxes.

2 Preliminaries

Masking is an algorithmic countermeasure that, while primarily intended to prevent power analysis attacks, also plays an essential role in SIFA attacks. In a masked cipher implementation, each input, output, and intermediate variable is split into $d+1$ *shares* so that their XOR is equal to the original *native variable* [24]. In Boolean masking, a native variable s is split random shares $s_0 \dots s_d$ that satisfy $s = s_0 \oplus \dots \oplus s_d$. As long as an attacker cannot observe a set of values statistically dependent on all $d+1$ shares of a native value, the computation is secure against classical power analysis techniques. Dealing with linear functions is trivial as they can be computed on each share individually. However, implementing masking for non-linear functions (S-Boxes) requires computations on all shares, which is more challenging to implement securely and correctly, and thus the main interest in the literature.

Redundant computation is an implementation-level fault attack countermeasure for cryptographic computations. The main idea is to perform the same computation several times and release a result only if the redundant computations match. This check prevents cases where an attacker forces faults in the computation, leading to incorrect results that correlate with native secrets [3]. Figure 1 shows the structure of a fault detection mechanism for redundant computations. If an attacker introduces a fault and the outputs do not match, output δ signals the faults and prevents the release of the result.

Statistical ineffective fault attacks (SIFA), first presented at CHES 2018 by Dobraunig et al., is a relatively new type of fault attack technique capable of circumventing common fault/power analysis countermeasures, while being

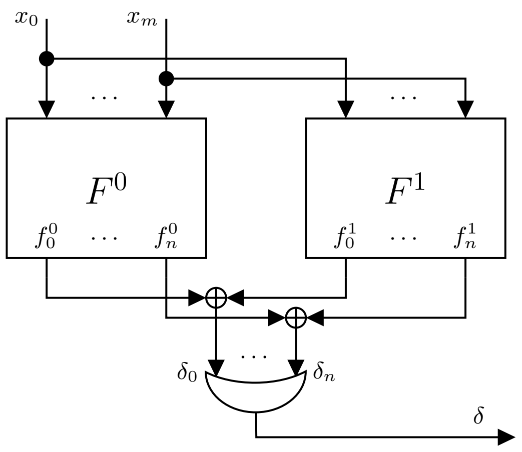

Fig. 1. A redundant computation with inputs $x_0, \ldots, x_m$, which are passed to both computation instances F^0 and F^1. The disjunction of differences $\delta_0, \ldots, \delta_n$ is used to determine whether there was a fault in one of the computation instances.

applicable to a wide variety of block ciphers or AEAD schemes [13,14,28,12]. When performing SIFA, an attacker calls a cryptographic operation (e.g. block cipher) with varying inputs, injects a fault during each of the computations, and only collects outputs in cases where the fault injection did not cause a faulty computation result (i.e. the output is not suppressed). This *filtered* set of outputs can then be used to perform a key recovery attack on a block cipher as follows.

A typical block cipher design of an iterated round function, consisting of a linear and non-linear layer, that mixes the current state with the cryptographic key such that in the end, each bit of the block cipher output is uniformly distributed. If we now consider, e.g., an AND computation that occurs in the non-linear layer of a (later) round function, one can observe that a fault-induced difference in one operand only propagates to the AND output if the other operand is '1'. Hence, if an attacker repeatedly calls a block cipher with varying inputs, while injecting the same difference in each computation, and only collecting outputs that are correct (not suppressed), a certain intermediate value should show a bias towards '0'. Given such a set of faulted but correct block cipher outputs, an attacker can now make a partial key guess of the last round key and calculate back to the faulted operation for each collected output (ciphertext). If the partial key guess was correct, the observed distribution of an intermediate value at that location should be biased. Otherwise, if the observed distribution is uniform, the key guess was wrong. For a more complete attack description targeting the AES-128 block cipher we refer to the description in [12].

If we now additionally consider masked implementations where each intermediate value is split into multiple random shares, filtering outputs based on the operand of one AND gate is not sufficient anymore. In fact, for SIFA to work in masked scenarios, the attacker needs to work with fault inductions that cause a difference that propagates into multiple AND gates that use the shares of one

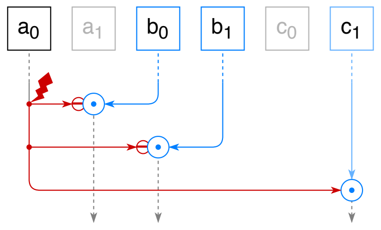

Fig. 2. Simplified example of SIFA against masked χ_3 using two shares. The induced difference cancels out, and the attacker learns $b_0 \vee b_1 \vee c_1$.

native value as other operands. We show this with a small example inspired by Daemen et al.[10].

Example 1. Consider a masked S-Box implementation that operates on shared inputs and outputs. For simplicity, assume that we repeatedly call this S-Box with uniformly distributed inputs and observe the corresponding outputs. Since an S-Box is a bijective function, uniformly distributed inputs should give uniformly distributed outputs. Figure 2 shows a reduced depiction of a masked χ_3 S-Box, the smaller version of χ_5, which is used in KECCAK (SHA-3). The S-Box takes a 3-bit input, represented by bits a, b and c. Therefore a first-order masked version of χ_3 takes the bits $a_0, a_1, b_0, b_1, c_0, c_1$ as input, with $a = a_0 \oplus a_1$, etc.. If we assume a fault targeting a_0 at the specified location, the induced bit-difference propagates into three AND-gates that take the bits b_0, b_1, and c_1 as the other inputs. In this case, the bit-difference cancels out and produces a value $\delta = b_0 \vee b_1 \vee c_1$. When a fault is not detected, an attacker knows that b_0, b_1, and c_1 are all zero, and therefore, b is zero as well. In this concrete case, the attacker uses a fault injection to filter out computations where the distribution of b is biased, and uses them to recover the key

Efficient SIFA countermeasures were presented at CHES 2020 [10]. Their SIFA mitigation strategy has almost no overhead and builds upon a careful combination of masking, redundant computation, and reversible computing. They show that, by building non-linear operations from incomplete and invertible building blocks, they achieve implementations where a single fault in the computation is either (1) not exploitable by SIFA, or (2) detectable via redundant computations. This approach is comparably easy to implement for small S-Boxes and can also be extended to larger S-Boxes such as the AES S-Box.

Boolean formulas are a symbolic composition of Boolean variables using logic operators. For a propositional boolean formula f, we write $\mathrm{Var}(f)$ to refer to the variables that occur in f. When clear from context, we write V to denote a superset of all used variables, *i.e.*, $\mathrm{Var}(f) \subseteq V$. The *partial evaluation* of f, where a variable q is assigned a value $p \in \mathbb{B}$ is written as $f\,[q \leftarrow p]$. Given a set of variables Q and an assignment $\alpha : Q \to \mathbb{B}$, we write $f\,[\alpha]$ to denote the partial evaluation of f where each variable in Q is assigned according to α.

We say that a formula f is *functionally dependent* on a variable x if and only if the concrete value of $x \in \mathbb{B}$ has an influence on the value of $f \in \mathbb{B}$. Henceforth, for a given formula f, we write $\mathcal{D}(f) \subseteq \mathrm{Var}(f)$ to denote the set of variables that f functionally depends on. That is, $x \in \mathcal{D}(f)$ if and only if there exists $\alpha : \mathrm{Var}(f) \setminus \{x\} \to \mathbb{B}$, such that $f[\alpha]\,[x \leftarrow \bot] \oplus f[\alpha]\,[x \leftarrow \top] = \top$. The above property can be checked by a SAT solver.

To discuss information leakage caused by a fault, we first define what it means for a formula f to contain information about another formula g. We define the weight of a Boolean function as $\#_V(f) = |\{\alpha : V \to \mathbb{B} \mid f[\alpha] = \top\}|$. Formulas f and g are *statistically dependent* if and only if $\#_V(f \wedge g) \cdot \#_V(\neg f) \neq \#_V(\neg f \wedge g) \cdot \#_V(f)$. That is, regardless of the observed value of f, the proportion of assignments α for which $g\,[\alpha] = \top$ is constant.

Example 2. Let $V = \{a, b, c\}$ be a set of variables. Let $f = a \wedge b$, $g = \neg a \vee c$, and $h = b \oplus c$ be Boolean formulas. Formulas f and g are statistically dependent because $\#_V(f \wedge g) \cdot \#_V(\neg f) = 6$ and $\#_V(\neg f \wedge g) \cdot \#_V(f) = 10$. Indeed, if $f\,[\alpha] = \top$, then probably $g\,[\alpha] = \bot$, whereas if $f\,[\alpha] = \bot$, then $g\,[\alpha] = \top$ is just as likely as $g\,[\alpha] = \bot$. The formulas f and h are statistically independent because $\#_V(f \wedge h) \cdot \#_V(\neg f) = 6$ and $\#_V(\neg f \wedge h) \cdot \#_V(f) = 6$.

We say that a Boolean formula f is *balanced* if and only if $\#_V(f) = \#_V(\neg f) = 2^{|V|-1}$. A Boolean variable x, interpreted as a formula, is inherently balanced for any variable set $x \in V$ as there are $2^{|V|-1}$ assignments $\alpha : V \to \mathbb{B}$ with $\alpha(x) = \top$. Lemma 1 states that this can be extended to functions of the form $f = x \oplus g$.

Lemma 1. *Let $f = x \oplus g$ be a Boolean formula with $x \notin \mathrm{Var}(g)$. We have that f is balanced.*

We measure the *Boolean distance* of two formulas f and g as the number of assignments where their values are different. This is equivalent to the weight of their difference $\#_V(f \oplus g)$. Lemma 2 states the connection between statistical independence and Boolean distance.

Lemma 2. *Let f and g be Boolean formulas and let f be balanced. Formulas f and g are statistically independent if and only if their difference is balanced.*

3 Verification Method

In this section, we introduce a method for verifying resistance against SIFA. That is, we show how to verify whether the fault-detection mechanism could give away information about native secrets processed by a software computation or hardware circuit. Our method focuses on proving the statistical independence of the fault-detection value δ and any of the secrets $s \in S$. We do not show this directly and instead try to prove the statistical independence using the *incompleteness*, *hiding*, and *inferred statistical independence* properties we introduce in this section. However, we first define the exact attack model considered in this verification approach.

3.1 Attack Model

Formally proving resistance against SIFA requires a definition of the attacker's capabilities and the exact information they observe. We use an attack model that is very similar to the one introduced by Daemen et al.[10]. We consider redundant masked implementations of S-Boxes that the attacker can query. Figure 1 shows a diagram of such an implementation, where the outputs of the two computation units are used to compute the fault-detection value δ. With SIFA, the value of δ is the only information the attacker receives from the computation. The goal of an attacker is to learn information about the native secret values processed by the computation. The inputs of the computation are categorized as masks and secret shares. In the rest of the section, we say that M is the set of mask variables, and S is the set of formulas representing the secrets. We, therefore, have the set of input variables $V = M \cup \bigcup_{s \in S} \text{Var}(s)$.

As SIFA is a fault attack, the attacker has the technical capabilities to introduce the fault that changes the value of an intermediate computation. If we represent δ as a computational circuit, a fault modifies the output of precisely one logic gate used during the computation. In our attack model, we consider faults that can negate the value of the gate by causing a bit-flip, which also captures many other fault models such as stuck-at faults for masked circuits [10]. The attacker's goal is to find a fault location that would cause a statistical dependency between δ and one of the formulas $s \in S$. Our verification does not currently take into account the possible effects of "glitchy" fault injections, i.e., faults with specific timing behavior that causes the output of gates to change (glitch) several times before reaching a stable logic state While it has been shown that such effects need to be taken into account for implementing masking correctly in hardware, it is currently not clear if, or to what extend, they are relevant for SIFA attacks in realistic attacker settings.

Proposition 1. *A computation with a fault-detection value δ is SIFA resistant against a fault-inducing attacker if δ is statistically independent of all native secrets $s \in S$.*

3.2 Incompleteness

First, we prove that a fault-detection formula δ that does not functionally depend on all shares of a secret s, cannot be statistically dependent on s. A syntactic version of this property is known as *non-interference* in the literature [5,4]. Intuitively, if one of the shares is absent from the formula δ, then an attacker cannot infer anything about s without this missing piece of information. Definition 1 formally states this intuition of *incomplete* secrets. Lemma 3 states that *incompleteness* is sufficient for statistical independence.

Definition 1. *Let f be a formula, and s be a secret represented by the formula $s_0 \oplus \ldots \oplus s_d$, where the shares s_i are variables. We say that a secret s is* incomplete *in formula f whenever $\mathcal{D}(s) \not\subseteq \mathcal{D}(f)$.*

Lemma 3. *Let secret $s = s_0 \oplus \ldots \oplus s_d$ be incomplete in the fault-detection formula δ. Then δ and s are statistically independent.*

3.3 Hiding

Assume that the formula δ is functionally dependent on all shares of a secret $s = s_0 \oplus \ldots \oplus s_d$, *i.e.*, $\mathcal{D}(s) \subseteq \mathcal{D}(\delta)$. Incompleteness, as defined in Definition 1, is thus not fulfilled. However, δ and s could still be statistically independent. Intuitively, if δ is balanced and masked by some uniformly random value, it cannot statistically correlate with any secret $s \in S$.

Definition 2. *A uniformly random variable x* hides *a secret $s \in S$ in the error-detection formula δ whenever $\delta = x \oplus f$, with $x \notin \mathcal{D}(s) \cup \mathcal{D}(f)$.*

Not all variables can hide secrets. Masks hide secrets because they are uniformly random by definition. Although individual shares s_i of a secret $s \in S$ are guaranteed to be uniformly random, their corresponding native secrets are not. Consequently, when investigating the hiding property from Definition 2, we only consider masks and shares of incomplete secrets in δ, as stated in Lemma 4.

Lemma 4. *Let δ be a formula, S' be the set of secrets that are incomplete in δ, i.e., $S' = \{s \in S \mid \mathcal{D}(s) \cap \mathcal{D}(\delta) \neq \emptyset\}$, M be the uniformly random mask variables, and X be the union $X = M \cup \bigcup_{s \in S'} \mathcal{D}(s)$. If there exists an $x \in X$ that hides a secret $s \in S$, then δ and s are statistically independent.*

Lemma 5 presents a method that tests whether the factorization needed for the *hiding* property is possible. The method uses a SAT solver and is similar to the method that checks functional dependencies.

Lemma 5. *Let f be a Boolean formula and $x \in \mathrm{Var}(f)$ be a variable. Then $f = x \oplus f\,[x \leftarrow \bot]$ if and only if $f\,[x \leftarrow \bot] \oplus f\,[x \leftarrow \top] = \top$.*

It is enough to find one uniformly random variable x to show that δ is statistically independent of all secrets $s \in S$. As discussed earlier, not all variables in $\mathrm{Var}(\delta)$ are eligible for the hiding property. Thus, our verification method only checks the hiding property after determining incomplete secrets first.

3.4 Inferred Statistical Independence

Although incompleteness and hiding are enough in most cases, the structure of δ can make them inapplicable. Therefore, it is possible that δ functionally depends on some secret s, and no uniformly random value hides s in δ. Example 3 illustrates this situation.

Example 3. Let δ be the fault-detection formula with $\delta = \delta_0 \vee \delta_1$, $\delta_0 = x \oplus s_0$ and $\delta_1 = y \oplus s_1$ be its sub-formulas, $M = \{x, y\}$ be the masks, and $s = s_0 \oplus s_1$ be a secret. Formula δ is functionally dependent on both s_0 and s_1, since there are no assignments $\alpha : \mathrm{Var}(\delta) \setminus \{s_i\} \to \mathbb{B}$ such that $\delta[\alpha]\,[s_i \leftarrow \bot] \oplus \delta[\alpha]\,[s_i \leftarrow \top] = \top$. Similarly, δ cannot be factorized into either $\delta = x \oplus \delta\,[x \leftarrow \bot]$ or $\delta = y \oplus \delta\,[y \leftarrow \bot]$, so neither x nor y hide s. However δ is indeed statistically independent of s because $\#_{\mathrm{Var}(\delta)}(\delta \wedge s) \cdot \#_{\mathrm{Var}(\delta)}(\neg\delta) = \#_{\mathrm{Var}(\delta)}(\neg\delta \wedge s) \cdot \#_{\mathrm{Var}(\delta)}(\delta) = 24$.

Therefore, because of the structure of the fault-detection formula δ, there is a real possibility that the incompleteness and hiding checks are not sufficient to show that δ does not statistically depend on any secrets. However, this can be mitigated by inferring whether δ is statistically independent of s by looking at its sub-formulas δ_i instead. Lemma 6 introduces a method for inferring the statistical independence of two Boolean formulas f and g, where one has the topmost operation Or, just like δ, and the other is a balanced function, just like a secret. This property is inspired by correlation propagation used in Rebecca [8].

Lemma 6. *Let $f = a \vee b$ and g be Boolean formulas with the variable sets $\mathrm{Var}(f) \subseteq V$ and $\mathrm{Var}(g) \subseteq V$. If $\bot$, a, b, and $a \oplus b$ are statistically independent of g, then f is also statistically independent of g.*

Therefore, at least in the case where $\delta = \delta_0 \vee \delta_1$, we can infer that δ is statistically independent of a secret s, as long as δ_0, δ_1, and s fulfill the conditions of Lemma 6. Example 4 illustrates this.

Example 4. Let δ, δ_0, δ_1 and s be as in Example 3. By Lemma 1, s is balanced. The hiding property applies for δ_0, δ_1 and $\delta_0 \oplus \delta_1$, where x, y, and $x \oplus y$ can be factorized out respectively. According to Lemma 2, all of the prerequisites for Lemma 6 are met, so we are able to show that δ is indeed statistically independent of s, without testing the statistical independence definition explicitly.

However, in general, δ will be a formula of the form $\delta = \bigvee_{i=1}^{n} \delta_i$. Although it is possible to apply Lemma 6 recursively, it is not ideal because we run into the same problem we demonstrated in Example 3, just one recursive application later. Luckily, Lemma 6 can be generalized to Or operations with multiple arguments, as shown in Theorem 1.

Theorem 1. *Let $\Phi = \{\phi_1, \dots, \phi_n\}$ be a set of Boolean formulas, $f = \bigvee_{i=1}^{n} \phi_i$ be their disjunction, g be another Boolean formula, and $\mathrm{Var}(f) \subseteq V$ and $\mathrm{Var}(g) \subseteq V$ be their variables. If for all $\Psi \in \mathcal{P}(\Phi)$, where $\mathcal{P}(\cdot)$ is the power-set operation, $f' = \bigoplus_{\psi \in \Psi} \psi$ is statistically independent of g, then so is f.*

Theorem 1 suggests that if we prove that all linear combinations of the error lines δ_i are statistically independent of a secret s, then we have indirectly shown that their disjunction δ is also statistically independent of s. Additionally, the condition of Theorem 1 can be further simplified because some of the linear combinations produced by $X \in \mathcal{P}(\Phi)$ could be equivalent. Instead of considering Φ, we could instead consider the maximal linearly independent subset of Φ.

Lemma 7. *Let Φ and g be as in Theorem 1. Let $\Phi' \subseteq \Phi$ be a linearly independent subset of Φ, i.e., $\forall \phi \in \Phi'.\ \forall \Psi \subseteq \Phi' \setminus \{\phi\}.\ \phi \neq \bigoplus_{\psi \in \Psi} \psi$, and let Φ' be maximal, i.e., $\forall \phi \in \Phi \setminus \Phi'.\ \exists \Psi \subseteq \Phi'.\ \phi = \bigoplus_{\psi \in \Psi} \psi$. If for all $\Psi \subseteq \Phi'$, $\bigoplus_{\psi \in \Psi} \psi$ is statistically independent of g, then the same holds for all $\Psi \subseteq \Phi$.*

As stated in Lemma 7, instead of considering all linear combinations in Φ, it is sufficient to consider only linear combinations of its maximally linearly independent subset Φ' when applying Theorem 1. In many cases, this substantially reduces the number of checks our verification method performed.

3.5 Approximating Statistical Independence

Theorem 1, together with the optimized condition from Lemma 7, is powerful enough to show that, given the mentioned conditions for δ_i, δ is statistically independent of a secret s. The statistical independence of the linear combinations of δ_i can be shown using the *incompleteness* and *hiding* properties discussed in Sections 3.2 and 3.3. However, issuing exponentially many satisfiability queries required by Theorem 1 is still undesirable. Therefore, we introduce an over-approximation which only calls the SAT solver to perform factorization and functional dependency tests for each relevant δ_i with all variables in $\mathrm{Var}(\delta_i)$. We then use the gathered data to over-approximate the incompleteness and hiding properties for all linear combinations of δ_i.

In general a Boolean formula f can be rewritten as an equivalent formula $f = g \oplus h$. Here $g = \bigoplus_{x \in X} x$ is the linear sub-formula where $X \subseteq \mathrm{Var}(f)$ is a set of variable symbols for which Lemma 5 applies, *i.e.*, $f\left[x \leftarrow \bot\right] \oplus f\left[x \leftarrow \top\right] = \top$. Consequently, h is the remaining sub-formula of f, *i.e.*, $h = f\left[\alpha\right]$ where $\alpha : X \mapsto \bot$ assigns $\bot$ to all variables in X. Henceforth, we write $\mathcal{C}(f)$ to denote the maximal set of variables that can be factorized out of f via Lemma 5, *i.e.*, $\mathcal{C}(f) = \{x \mid x \in \mathrm{Var}(f), f\left[x \leftarrow \bot\right] \oplus f\left[x \leftarrow \top\right] = \top\}$. Furthermore, call $f = f^{\mathrm{lin}} \oplus f^{\mathrm{nl}}$ the maximal factorization, where $f^{\mathrm{lin}} = \bigoplus_{x \in \mathcal{C}(f)} x$, $f^{\mathrm{nl}} = f\left[\alpha\right]$ and $\alpha : \mathcal{C}(f) \mapsto \bot$. Knowing both $\mathcal{C}(f)$ and $\mathcal{D}(f)$ allows us to perform easy hiding and incompleteness checks for f against some linear formula f'. Additionally, $\mathcal{C}(\cdot)$ and $\mathcal{D}(\cdot)$ allow us to approximate the maximal factorization for linear combinations $f = \bigoplus_{i=1}^{n} \phi_i$, where ϕ_i themselves are also formulas.

Lemma 8. *Let $f = \bigoplus_{i=1}^{n} \phi_i$ be a formula with sub-formulas ϕ_i. The variable set $\widehat{\mathcal{C}}(f) = \triangle_{i=1}^{n} \mathcal{C}(\phi_i) \setminus \bigcup_{i=1}^{n} \mathcal{D}({\phi_i}^{\mathrm{nl}})$ is an under-approximation of $\mathcal{C}(f)$. Similarly, the set $\widehat{\mathcal{D}}(f) = \triangle_{i=1}^{n} \mathcal{C}(\phi_i) \cup \bigcup_{i=1}^{n} \mathcal{D}({\phi_i}^{\mathrm{nl}})$ is an over-approximation of $\mathcal{D}(f)$.* [1]

These two approximations are much easier to compute than the real variable sets $\mathcal{C}(\delta)$ and $\mathcal{D}(\delta)$. Ideally, we first compute $\mathcal{D}(\delta_i)$ and $\mathcal{C}(\delta_i)$ for each of the fault-detection values δ_i using a SAT solver. Afterward, when checking all their linear combinations, we only use fast set computation operations from Lemma 8. Since $\widehat{\mathcal{D}}(\cdot)$ is an over-approximation, it must contain all functional dependencies and possibly some spurious ones. If we show the incompleteness of a secret s with $\widehat{\mathcal{D}}(\cdot)$, we would have gotten the same result with $\mathcal{D}(\cdot)$. Similarly, $\widehat{\mathcal{C}}(\cdot)$ contains a subset of the variables that can be factorized out of the formula. It is still a factorization, although it is not guaranteed to be maximal like $\mathcal{C}(\cdot)$. Therefore, if we show that a secret is hidden by some uniformly random variable using $\widehat{\mathcal{C}}(\cdot)$, it is guaranteed to be hidden.

3.6 Verification Algorithm

In this section, we summarize how the verification algorithm works. In particular, we focus on the order of checks performed by the algorithm and show how

[1] Operator $\triangle$ signifies symmetric difference: $A \triangle B = (A \cup B) \setminus (A \cap B)$

Algorithm 1: *Danira* algorithm for verifying SIFA resistance

```
Input : fault detection formulas $\{\delta_1, \ldots, \delta_n\}$, $\delta := \bigvee_{i=1}^{n} \delta_i$
        masks $M$, secrets $S = \{s^1, \ldots, s^d\}$
Output: secure or unknown
1  $R := M$ ;                                                        // variables that hide
2  $K := \emptyset$ ;                                                // complete secrets
3  for $s \in S$ do
4      if $\mathcal{D}(s) \subseteq \mathcal{D}(\delta)$ then $K := K \cup \{s\}$;          // mark as complete
5      if $\mathcal{D}(s) \not\subseteq \mathcal{D}(\delta)$ then $R := R \cup (\mathcal{D}(s) \cap \mathcal{D}(\delta))$;  // shares can hide
6  if $K = \emptyset$ or $R \cap \mathcal{C}(\delta) \neq \emptyset$ then return secure;   // incomplete or hidden
7  $G := \emptyset$ ;                                                // basis of $\delta_i$ formulas
8  for $i \in \{1, \ldots, n\}$ do
9      if $\forall G' \subseteq G.\ \delta_i \neq \bigoplus_{g \in G'} g$ then $G := G \cup \{\delta_i\}$;  // include $\delta_i$ in basis $G$
10 for $G' \subseteq G$ do
11     $\phi = \bigoplus_{g \in G'} g$ ;                                 // comb. of sub-formulas
12     if $\forall s \in K. \mathcal{D}(s) \not\subseteq \widehat{\mathcal{D}}(\phi)$ then continue;       // no secrets complete
13     if $R \cap \widehat{\mathcal{C}}(\phi) \neq \emptyset$ then continue;                  // secrets are hidden
14     return unknown ;                                              // $\phi$ maybe dependent
15 return secure ;                                                   // all $\phi$ independent
```

they correspond to the previous exposition. As described in Section 3.1, the attacker can introduce a fault in any sub-formula ϕ of δ. The verification method summarized in Algorithm 1 is given the faulted δ and its sub-formulas δ_i, the set of masks M, and the set of formulas S representing each secret as a linear combination of its shares. The show algorithm considers only one fault at a time, and our tool *Danira* runs it separately for each possible fault location.

First, the algorithm computes the set K of complete secrets, *i.e.*, secrets for which δ functionally depends on all its shares. Simultaneously, the algorithm computes the set R of uniformly random values that contains all masks M and shares of incomplete secrets $s \notin K$. In the rest of the algorithm, only values in R can hide secrets. If there are no complete secrets in K or a uniformly random variable from R can be factorized out of δ and hides all secrets in K, we know that δ is statistically independent of the secrets S.

Next, the algorithm computes a maximal linearly independent subset G of fault-detection values δ_i. As discussed previously in Lemma 6, it is sufficient to apply Theorem 1 to this subset when proving statistical independence. The algorithm computes the approximations $\widehat{\mathcal{D}}(\phi)$ and $\widehat{\mathcal{C}}(\phi)$ for all possible linear combinations ϕ from G. It uses the approximations to check whether any of the secrets in K are complete in $\widehat{\mathcal{D}}(\phi)$, and if they are, whether any of the random values from R appear in $\widehat{\mathcal{C}}(\phi)$ and hide them. If we were able to show statistical independence of secrets for all ϕ, Algorithm 1 declares the computation secure for the given fault.

Theorem 2. *Algorithm 1 is sound: if it returns* **secure***, the analyzed fault in the attack model from Section 3.1 is not exploitable via SIFA.* □

Procedure Chi3: Implementation of a masked Keccak χ_3 S-Box [10]

```
Input  : ${a_0, a_1}, {b_0, b_1}, {c_0, c_1}, M = {m_r, m_t}$
Output: ${r_0, r_1}, {s_0, s_1}, {t_0, t_1}$
1  $m_s := m_r \oplus m_t$;
2  $x_0 := \neg b_0 \wedge c_1$;
3  $x_2 := a_1 \wedge b_1$;
4  $x_1 := \neg b_0 \wedge c_0$;
5  $x_3 := a_1 \wedge b_0$;
6  $r_0 := x_0 \oplus m_r$;
7  $t_1 := x_2 \oplus m_t$;
8  $r_0 := r_0 \oplus x_1$;
9  $t_1 := t_1 \oplus x_3$;
10 $x_0 := \neg c_0 \wedge a_1$;
11 $x_2 := b_1 \wedge c_1$;
12 $x_1 := \neg c_0 \wedge a_0$;
13 $x_3 := b_1 \wedge c_0$;
14 $s_0 := x_0 \oplus m_s$;
15 $r_1 := x_2 \oplus m_r$;
16 $s_0 := s_0 \oplus x_1$;
17 $r_1 := r_1 \oplus x_3$;
18 $x_0 := \neg a_0 \wedge b_1$;
19 $x_2 := c_1 \wedge a_1$;
20 $x_1 := \neg a_0 \wedge b_0$;
21 $x_3 := c_1 \wedge a_0$;
22 $t_0 := x_0 \oplus m_t$;
23 $s_1 := x_2 \oplus m_s$;
24 $t_0 := t_0 \oplus x_1$;
25 $s_1 := s_1 \oplus x_3$;
26 $r_0 := r_0 \oplus a_0$;
27 $t_1 := t_1 \oplus c_1$;
28 $s_0 := s_0 \oplus b_0$;
29 $r_1 := r_1 \oplus a_1$;
30 $t_0 := t_0 \oplus c_0$;
31 $s_1 := s_1 \oplus b_1$;
```

4 Case Studies

This section evaluates our new verification approach against the secured implementations presented by Daemen et al.[10]. *Danira* [2] uses the netlist of a combinatorial circuit as the input. It interprets the inputs as variables and the intermediate computations as Boolean formulas. From a theoretical standpoint, it does not matter whether the analyzed circuit has a state or not because we only consider the outputs after the computation finishes.

In the rest of this section, we consider the SIFA-resistant masked implementations of Keccak χ_3, all classes of quadratic 4-bit S-Boxes, and an AES S-Box [10]. We argue that without a sophisticated verification method, it is extremely easy to introduce bugs that produce correct computations but break the theoretical SIFA-resistance guarantees.

Finally, we summarize the performance of *Danira* on several versions of the same designs.

4.1 Masked Keccak χ_3

The Keccak permutation χ_3 is a simple circuit with three inputs and three outputs used in many lightweight ciphers. Implementing a masked version is straightforward because of its low polynomial degree. Chi3 shows the masked computation of χ_3 proposed by Daemen et al.[10]. The secrets processed by the circuit are $a = a_0 \oplus a_1$, $b = b_0 \oplus b_1$ and $c = c_0 \oplus c_1$, whereas m_r and m_t are used as uniformly random masks. The results of the computation r, s, and t are also split into two shares, respectively. The circuit was designed in such a way that the outputs are used for fault detection. Given two redundant computations of Chi3 with outputs $\{r_0, r_1, s_0, s_1, t_0, t_1\}$ and $\{r'_0, r'_1, s'_0, s'_1, t'_0, t'_1\}$, the fault-detection values are defined as $\delta_1 = r_0 \oplus r'_0, \ldots, \delta_6 = t_1 \oplus t'_1$.

Each line of Chi3 is a possible fault location according to our attack model in Section 3.1. Introducing a bit-flip fault means negating the result of one such line in one of the redundant computations. Our verification method goes through

[2] *Danira*'s code is available at https://extgit.iaik.tugraz.at/scos/danira

each of the fault locations, negates the result at that point in the computation, and generates the fault-detection formulas $\delta_1, \ldots, \delta_6$. We specify $S = \{a, b, c\}$ and $M = \{m_r, m_t\}$, and run Algorithm 1 to see if the considered fault could leak information about the secrets.

We implemented the netlist for Chi3 manually, and *Danira* was able to verify that the design proposed in [10] was indeed SIFA resistant. However, when we synthesized an equivalent RTL design with Yosys, *Danira* reported that it could not prove SIFA resistance. In the synthesized netlist, Yosys introduced a temporary gate $v_0 = \neg b_0$ which it used to simplify Line 2 to $x_0 := v_0 \wedge c_1$ and Line 4 to $x_1 := v_0 \wedge c_0$. Although this makes sense from an optimization perspective because it effectively reduces the size of the circuit by one gate, it breaks the SIFA resistance. A fault at this new gate v_0 in the synthesized design is the same as two faults at Lines 2 and 4. As a result, δ becomes statistically dependent on c, which the attacker can exploit. Unfortunately, this demonstrates that *(1)* an analysis on the gate level is unavoidable and *(2)* they must be implemented manually, as synthesis tools or compilers break SIFA resistance while maintaining functional correctness.

4.2 Masked AES S-Box

Compared to χ_3, the AES S-Box is a significantly more complex circuit of high polynomial degree. The authors of the CHES paper [10] propose a high-level sketch of a SIFA-resistant masked AES S-Box. There are many ways to implement this high-level description and achieving SIFA resistance is not trivial. After several failed attempts, we managed to implement a protected version of the proposed AES S-Box with the help of our new verification tool. We are convinced that correctly protecting a circuit as large as an AES S-Box is infeasible without the help of an automated verification method such as *Danira*.

4.3 Performance Evaluation

This section gives a breakdown of *Danira*'s performance on correctly (and incorrectly) protected implementations. We performed all experiments on a notebook with an eight-core *Intel i7-8550U 1.8GHz* CPU and *16* GiB of memory.

As shown in Table 1, *Danira* instantly verified (or falsified) all tested Keccak χ_3 and quadratic 4-bit S-Box designs. We also demonstrate that for Keccak χ_3 and the AES S-Box, even one re-used gate leads to vulnerabilities. *Danira* verifies the SIFA resistance of our implementation in about three minutes. For the AES S-Boxes, *Danira* performs significantly better than a version SILVER [25] which we extended to verify SIFA resistance. However, although this shows *Danira*'s potential, our extension of SILVER with construct as shown in Figure 1 is not perfect and could be further improved by its authors.

In summary, the results of our experiments in Table 1 indicate that: (1) the over-approximation we introduce in this paper is strong enough to prove SIFA resistance for secure designs, and (2) our verification method applied by *Danira* is fast enough for complex masked implementations.

Table 1. Performance of *Danira* (D) and a modified version of SILVER [25] (S) for different masked designs. Correct (incorrect) designs are denoted by ✔ (✖). In all cases, the reused gate was the exploitable fault location.

Design	Gates	($\wedge$)	($\oplus$)	Result	D (s)	S (s)
Keccak χ_3, full Chi3	37	12	25	✔	0.06	0.24
Keccak χ_3, reuse $\neg b_0$	36	12	24	✖	0.05	0.07
Keccak χ_3, reuse $\neg c_0$	36	12	24	✖	0.06	0.12
Keccak χ_3, reuse $\neg a_0$	36	12	24	✖	0.06	0.18
4-bit perm. $\mathcal{Q}^4_4$ [10]	10	4	6	✔	0.03	0.10
4-bit perm. $\mathcal{Q}^4_{12}$ [10]	20	8	12	✔	0.05	0.16
4-bit perm. $\mathcal{Q}^4_{293}$ [10]	30	12	18	✔	0.05	0.23
4-bit perm. $\mathcal{Q}^4_{294}$ [10]	30	12	18	✔	0.04	0.21
4-bit perm. $\mathcal{Q}^4_{299}$ [10]	50	20	30	✔	0.07	0.41
4-bit perm. $\mathcal{Q}^4_{300}$ [10]	36	12	24	✔	0.06	0.26
AES S-Box, reuse g_{104}	631	144	487	✖	14.67	551.1
AES S-Box, reuse g_{240}	631	144	487	✖	83.28	1336.7
AES S-Box, reuse g_{360}	631	144	487	✖	135.04	1941.7
AES S-Box, full [10]	634	144	490	✔	184.39	3297.4

5 Conclusion

Protecting masked implementations against SIFA is not straightforward. Designers can make mistakes when implementing a specification that is supposed to be secure. Additionally, compilers and synthesis tools can introduce simplifications that break the SIFA-resistance guarantees. *Danira* solves these problems using simple yet effective properties of redundant masked implementations to show whether they are SIFA resistant. As demonstrated by our case studies, *Danira* is able to verify designs that may be used in actual embedded systems. In cases where *Danira* cannot prove the security of a design, it gives a developer detailed debugging information about a problematic fault location.

References

1. V. Arribas, S. Nikova, and V. Rijmen. VerMI: Verification tool for masked implementations. In *ICECS*, 2018.
2. V. Arribas, F. Wegener, A. Moradi, and S. Nikova. Cryptographic fault diagnosis using VerFI. *IACR Cryptol. ePrint Arch.*, 2019.
3. H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2), 2006.
4. G. Barthe, S. Belaïd, G. Cassiers, P. Alain Fouque, B. Grégoire, and F.-X. Standaert. maskVerif: Automated verification of higher-order masking in presence of physical defaults. In *ESORICS*, 2019.
5. G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, and P.-Y. Strub. Verified proofs of higher-order masking. In *EUROCRYPT*, 2015.
6. G. Barthe, F. Dupressoir, S. Faust, B. Grégoire, F.-X. Standaert, and P.-Y. Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In *EUROCRYPT*, 2017.

7. E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, 1997.
8. R. Bloem, H. Groß, R. Iusupov, B. Könighofer, S. Mangard, and J. Winter. Formal verification of masked hardware implementations in the presence of glitches. In *EUROCRYPT*, 2018.
9. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In *EUROCRYPT*, 1997.
10. J. Daemen, C. Dobraunig, M. Eichlseder, H. Groß, F. Mendel, and R. Primas. Protecting against statistical ineffective fault attacks. *TCHES*, 2020.
11. S. Dhooghe and S. Nikova. My gadget just cares for me – how NINA can prove security against combined attacks. In *CT-RSA*, 2020.
12. C. Dobraunig, M. Eichlseder, H. Groß, S. Mangard, F. Mendel, and R. Primas. Statistical Ineffective Fault Attacks on masked AES with fault countermeasures. In *ASIACRYPT*, 2018.
13. C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas. SIFA: exploiting ineffective fault inductions on symmetric cryptography. *TCHES*, 2018.
14. C. Dobraunig, S. Mangard, F. Mendel, and R. Primas. Fault attacks on nonce-based authenticated encryption: Application to Keyak and Ketje. In *SAC*, 2018.
15. S. Faust, V. Grosso, S. Merino Del Pozo, C. Paglialonga, and F.-X. Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *TCHES*, 2018.
16. T. Fuhr, É. Jaulmes, V. Lomné, and A. Thillard. Fault attacks on AES with faulty ciphertexts only. In *FDTC*, 2013.
17. P. Gao, H. Xie, J. Zhang, F. Song, and T. Chen. Quantitative verification of masked arithmetic programs against side-channel attacks. In *TACAS*, 2019.
18. P. Gao, J. Zhang, F. Song, and C. Wang. Verifying and quantifying side-channel resistance of masked software implementations. *TOSEM*, 28(3), 2019.
19. B. Gigerl, V. Hadzic, R. Primas, S. Mangard, and R. Bloem. Coco: Co-design and co-verification of masked software implementations on CPUs. In *USENIX*, 2021.
20. H. Groß, R. Iusupov, and R. Bloem. Generic low-latency masking in hardware. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018.
21. H. Groß and S. Mangard. Reconciling d+1 masking in hardware and software. In *CHES*, 2017.
22. V. Hadzic, R. Primas, and R. Bloem. Proving SIFA protection of masked redundant circuits. *CoRR*, abs/2107, 2021.
23. M. Hutter and J.-M. Schmidt. The temperature side channel and heating fault attacks. In *CARDIS*, 2013.
24. Y. Ishai, A. Sahai, and D. A. Wagner. Private Circuits: Securing hardware against probing attacks. In *CRYPTO*, 2003.
25. D. Knichel, P. Sasdrich, and A. Moradi. SILVER - statistical independence and leakage verification. In *ASIACRYPT*, 2020.
26. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, 1999.
27. J.-J. Quisquater and D. Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *E-smart*, 2001.
28. K. Ramezanpour, P. Ampadu, and W. Diehl. A statistical fault analysis methodology for the Ascon authenticated cipher. In *HOST*, 2019.
29. S. Saha, D. Jap, D. B. Roy, A. Chakraborty, S. Bhasin, and D. Mukhopadhyay. A framework to counter statistical ineffective fault analysis of block ciphers using domain transformation and error correction. *TIFS*, 2020.