IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 4, pp. 1–39. DOI:10.46586/tches.v2024.i4.1-39

SAT-based Formal Verification of Fault Injection Countermeasures for Cryptographic Circuits*

Huiyu Tan[1,2], Pengfei Gao[3], Fu Song[4,5] (✉), Taolue Chen[6] and Zhilin Wu[4]

[1] ShanghaiTech University, Shanghai 201210, China
[2] Wingsemi Technology Co., Ltd., Shanghai 201203, China
[3] Bytedance, Beijing 100098, China
[4] Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China, songfu@ios.ac.cn
[5] Nanjing Institute of Software Technology, Nanjing 211135, China
[6] Birkbeck, University of London, WC1E 7HX, UK

Abstract. Fault injection attacks represent a type of active, physical attack against cryptographic circuits. Various countermeasures have been proposed to thwart such attacks, however, the design and implementation of which are intricate, error-prone, and laborious. The current formal fault-resistance verification approaches are limited in efficiency and scalability. In this paper, we formalize the fault-resistance verification problem and show that it is coNP-complete. We then devise a novel approach for encoding the fault-resistance verification problem as the Boolean satisfiability (SAT) problem so that modern off-the-shelf SAT solvers can be utilized. The approach is implemented in an open-source tool FIRMER which is evaluated extensively on realistic cryptographic circuit benchmarks. The experimental results show that FIRMER is able to verify fault-resistance of almost all (72/76) benchmarks in 3 minutes (the other three are verified in 35 minutes and the hardest one is verified in 4 hours). In contrast, the prior approach fails on 31 fault-resistance verification tasks even after 24 hours (per task).

Keywords: Fault Injection · Cryptographic Circuits · SAT · Formal Verification

1 Introduction

Cryptographic circuits have been widely used in providing secure authentication, privacy, and integrity, due to rising security risks in sensor networks, healthcare, cyber-physical systems, and the Internet of Things [AIM10, TS21, NIS22]. However, cryptographic circuits are vulnerable to various effective physical attacks, which remains an open challenge even after two decades of research. This paper focuses on an infamously effective attack, i.e., fault injection attacks [BS97, BDF+14, BHL17, Bak22].

Fault injection attacks deliberately inject disturbances into a cryptographic circuit when it is running cryptographic computation, and analyze the information from the correct (non-faulty) and the incorrect (faulty) outputs, attempting to deduce information on the secret key. Fault injection attacks allow the adversary to bypass certain assumptions in classical cryptanalysis methods where the cipher is considered to be a black box and therefore cannot

*This work was funded by the Strategic Priority Research Program of CAS (XDA0320101), National Natural Science Foundation of China (62072309), CAS Project for Young Scientists in Basic Research (YSBR-040), ISCAS New Cultivation Project (ISCAS-PYFX-202201), ISCAS Fundamental Research Project (ISCAS-JCZD-202302), an overseas grant from the State Key Laboratory of Novel Software Technology, Nanjing University (KFKT2023A04).

Received: 2024-04-15 Accepted: 2024-06-15 Published: 2024-09-05

be tampered. The disturbances could be injected in various different ways, such as clock glitches [ADN+10, ESH+11, SHO19], underpowering [SGD08], voltage glitches [ZDCT13], electromagnetic pulses [DDRT12, DLM19, DLM21] and laser beams [SA03, RSDT13, CLFT14, SFG+16, DBC+18]. Secret information can be deduced by differential fault analysis [BS97], ineffective fault analysis [Cla07a], statistical fault analysis [FJLT13], and statistical ineffective fault analysis [DEG+18]. Therefore, fault injection attacks pose a severe security threat to embedded computing devices with cryptographic modules.

Both detection-based and correction-based countermeasures have been proposed to defend against fault injection attacks [MSY06, AMR+20, SRM20]. The former aims to detect fault injections and infect the output result with an error flag in the presence of faults so an attacker cannot exploit them; the latter aims to correct the faulty cryptographic computation in the presence of faults. An effective countermeasure must be *fault-resistance*, i.e., detecting or correcting faults in time once they occur. Designing and implementing secure, efficient, and low-cost cryptographic circuits is notably error-prone, hence it is crucial to rigorously verify fault-resistance, especially at the gate level (which is closer to the final circuit sent to the fab for the tape-out). Typically this is done at the last stage of the front-end design so the flaws introduced by front-end tools (e.g., optimization passes) can be detected.

There is more specialized work for assuring fault-resistance, (e.g., [SKK13, BGE+17, SMD18, AWMN20, KRH17, SSR+20, WLR+21, NOV+22]), but almost all of them focus on finding flaws or checking the effectiveness of user-specified instances (given by fault test vectors). In principle, to achieve completeness, all the possible test vectors (varying in fault types, injected gates, and clock cycles) must be checked under all valid input combinations, which is virtually infeasible in practice, as already recognized, e.g., by [RBSS+21]. To alleviate this issue, recently, a binary decision diagram (BDD) [Bry86] based approach, called FIVER [RBSS+21], was proposed, which does not need to explicitly enumerate all valid input combinations and is optimized to avoid some fault test vectors. However, it still has to repeatedly build BDD models for a huge number of fault test vectors, failing to verify relatively larger circuits in a reasonable amount of time. (For instance, it fails to prove fault-resistance of a single-round 2-bit protected AES in 24 hours.)

Contributions. In this work, inspired by the consolidated fault model for precisely defining fault injection adversaries [RBSG23], we define a fault-resistance model as $\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$, where $\mathtt{n}_e$ specifies the maximum number of fault events per clock cycle, $\mathtt{n}_c$ specifies the maximum number of clock cycles in which fault events can occur, T specifies the set of allowed fault types including bit-set, bit-reset and bit-flip; and ℓ specifies the types of gates that can be faulted including logic gates in combinational circuits and memory gates.

Note that, as in [RBSG23, RBSS+21], we focus on *transient* fault events that are of a dynamic nature and become inactive after certain periods or changes in the circuit. More precisely, each fault event is accompanied with the clock cycle of the fault injection. There are *persistent* and *permanent* fault events which are of a static nature and will remain active for several or even the entire clock cycles. As remarked by [RBSS+21], the latter two can be modeled as repetitive transient fault events, thus are not *explicitly* considered in the fault-resistance model. Moreover, following [RBSG23, RBSS+21], we assume that the adversary can precisely control fault injections, consequently, fault-resistance against random faults can be achieved as they can be encoded by fault vectors and our verification approach covers all the feasible fault vectors. We formalize the fault-resistance verification problem using the fault-resistance model which is shown to be coNP-complete.[1] This lays a solid foundation for the subsequent verification.

We propose novel SAT-based approaches for verifying fault-resistance. Technically, with

[1] In the computational complexity theory, coNP is the class of problems the complement of which are in NP, where NP is the class of problems which can be solved in polynomial-time by a nondeterministic Turing machine.

a countermeasure and a fault-resistance model, we generate a new conditionally-controlled faulty circuit, which is in turn reduced to the SAT problem. Intuitively we replace each vulnerable gate with a designated gadget (i.e., sub-circuit) with (1) a control input for controlling if a fault is injected on the gate, and (2) selection inputs for choosing which fault type is injected. This approach avoids explicit enumeration of all the possible fault test vectors and can fully utilize the conflict-driven clause learning (CDCL) feature of modern SAT solvers. Furthermore, we introduce a reduction technique to safely reduce the number of vulnerable gates when verifying fault-resistance, which significantly improves the verification efficiency.

We implement our approach in an open-source tool FIRMER (**F**ault **I**njection counte**R** **M**easure verifi**ER**), based on Verilog gate-level netlists. We evaluate FIRMER on 33 realistic cryptographic circuits (i.e., rounds of AES, CRAFT, LED, GIFT, PRESENT and SIMON) with both detection- and correction-based countermeasures, where the number of gates ranges from 608 to 68,703. The results show that our approach is effective and efficient in verifying the fault-resistance against various fault-resistant models. Almost all the benchmarks (72 out of 76) can be verified in less than 3 minutes (except for three which take 35 minutes and one which takes 4 hours). In comparison, FIVER runs out of time (with timeout 24 hours) on 31 fault-resistance verification tasks in the same setting.

To summarize, we make the following major contributions:

- We formalize the fault-resistance verification problem and identify its coNP-complete computational complexity for the first time.
- We propose a novel SAT-based approach for formally verifying fault-resistance with an accelerating technique.
- We implement an open-source tool for verifying fault-resistance in Verilog gate-level netlists.
- We extensively evaluate our tool on realistic cryptographic circuits, demonstrating its effectiveness and efficiency.

Outline. Section 2 briefly recaps circuits, fault injection attacks and their countermeasures. Section 3 formulates the fault-resistance verification problem, studies its computational complexity, and introduces an illustrating example. Section 4 presents our SAT-based verification approach. Section 5 reports experimental results. We discuss related work in Section 6 and finally conclude this work in Section 7.

To foster further research, benchmarks, experimental data and the source code of our tool are released at

https://github.com/S3L-official/FIRMER.

2 Preliminary

2.1 Notations

We denote by $\mathbb{B}$ the Boolean domain $\{\mathtt{0}, \mathtt{1}\}$ and by $[n]$ the set of integers $\{1, \cdots, n\}$ for an integer $n \geq 1$. To describe standard circuits, we consider the logic gates: `and` ($\wedge$), `or` ($\vee$), `nand` ($\overline{\wedge}$), `nor` ($\overline{\vee}$), `xor` ($\oplus$), `xnor` ($\overline{\oplus}$), and `not` ($\neg$), all of which are binary gates except for `not`. Note that $\overline{\bullet}(x_1, x_2) = \neg \bullet (x_1, x_2)$, so $\overline{\overline{\bullet}}$ may be used to denote $\bullet$ for $\bullet \in \{\wedge, \vee, \oplus\}$. In addition, we introduce three auxiliary logic gates to describe faulty circuits: `nnot` ($\overline{\neg}$), `set` ($\sqcap$) and `reset` ($\sqcup$), where $\overline{\neg} x = x$, $\sqcap$ and $\sqcup$ are two constant logic gates whose outputs are `1` and `0`, respectively.

2.2 Synchronous Circuits

We first introduce combinational circuits based on which we define synchronous circuits.

Definition 1 (Combinational circuit)**.** A *combinational circuit* C is a tuple

$$(V, I, O, E, \mathsf{g}),$$

where

- V is a finite set of vertices, $I \subset V$ is a set of inputs, and $O \subset V$ is a set of outputs;
- $E \subseteq (V \setminus O) \times (V \setminus I)$ is a set of edges each of which represents a wire connecting two vertices and carrying a digital signal from the domain $\mathbb{B}$;
- (V, E) forms a directed acyclic graph (DAG);
- each internal vertex $v \in V \setminus (I \cup O)$ is a logic gate associated with its function, given by $\mathsf{g}(v)$, whose fan-in size is equal to the in-degree of the vertex v.

Intuitively, a combinational circuit represents a Boolean function. The behavior of a combinational circuit is memoryless, namely, the outputs depend solely on the inputs and are independent of the circuit's past history. The semantics of the combinational circuit C is described by the associated Boolean function $[\![C]\!] : \mathbb{B}^{|I|} \to \mathbb{B}^{|O|}$ such that for any signals $\vec{x} \in \mathbb{B}^{|I|}$ of the inputs I, $[\![C]\!](\vec{x}) = \vec{y}$ iff under the input signals $\vec{x}$ the output signals O of the circuit C are $\vec{y}$.

A (synchronous) sequential circuit is a combinational circuit with feedback synchronized by a global clock. It has primary inputs, primary outputs, a combinational circuit and memory in the form of registers (or flip-flops). The output signals of registers at a clock cycle represent an internal state. At each clock cycle, the combinational circuit produces its output using the current internal state and the primary inputs as its inputs. The output comprises two parts: one is used as primary output while the other is stored in the registers, which will be the internal state for the next clock cycle and can be seen as the feedback of the combinational circuit to the next clock cycle.

We focus on round-based circuit implementations of cryptographic primitives so that the synchronous circuits always have bounded clock cycles and can be automatically unrolled by clock cycles. However, in theory, our methodology is generic and our approach may be adapted to handle other architectures with bounded clock cycles. The details are left as future work.

Definition 2 (Synchronous circuit)**.** A *k-clock cycle synchronous (sequential) circuit $\mathcal{S}$* for $k \geq 1$ is a tuple

$$(\mathcal{I}, \mathcal{O}, \mathcal{R}, \vec{s}_0, \mathcal{C}),$$

where

- $\mathcal{I}$ (resp. $\mathcal{O}$) is a finite set of primary inputs (resp. primary outputs);
- $\mathcal{R} = R_0 \uplus \cdots \uplus R_k$ is a set of registers, called *memory gates*;
- $\vec{s}_0 \in \mathbb{B}^{|R_0|}$ gives initial signals to the memory gates in R_0;
- $\mathcal{C} = \{C_1, \cdots, C_k\}$ where $C_i = (V_i, I_i, O_i, E_i, \mathsf{g}_i)$ for each $i \in [k]$ is a combinational circuit. Moreover, the inputs I_i are only connected from the primary inputs and memory gates R_{i-1}, the outputs O_i are only connected to the primary outputs and memory gates R_i, and $V_i \cap V_j = \emptyset$ for any $j \neq i$.

Since memory gates are used for synchronization only and are essentially the same as the identity function, for the sake of presentation, we extend the function g_i such that for every memory gate $r \in R_{i-1}$, $\mathsf{g}_i(r) = \neg\!\!\neg$. However, we emphasize that it may be changed if fault injections are considered.

A *state* $\vec{s}$ of the circuit $\mathcal{S}$ comprises the output signals of the memory gates. At any clock cycle $i \in [k-1]$, given a state $\vec{s}_{i-1}$ and signals $\vec{x}_i$ of the primary inputs $\mathcal{I}$, the next state $\vec{s}_i$ is $[\![C_i]\!](\vec{s}_{i-1}, \vec{x})$ projected onto the registers R_i and $[\![C_i]\!](\vec{s}_{i-1}, \vec{x})$ projected onto $\mathcal{O}$ gives the primary outputs $\vec{y}_i$. In general, we write $\vec{s}_{i-1} \xrightarrow{\vec{x}_i | \vec{y}_i} \vec{s}_i$ for the state transition at the i-th clock cycle.

A *run* π under a given sequence of primary inputs $(\vec{x}_1, \cdots, \vec{x}_k)$ is a sequence

$$\vec{s}_0 \xrightarrow{\vec{x}_1|\vec{y}_1} \vec{s}_1 \xrightarrow{\vec{x}_2|\vec{y}_2} \vec{s}_2 \xrightarrow{\vec{x}_3|\vec{y}_3} \vec{s}_3 \cdots \vec{s}_{k-1} \xrightarrow{\vec{x}_k|\vec{y}_k} \vec{s}_k.$$

The *semantics* of the circuit $\mathcal{S}$ is described by its associated Boolean function

$$[\![\mathcal{S}]\!] : (\mathbb{B}^{|\mathcal{I}|})^k \to (\mathbb{B}^{|\mathcal{O}|})^k$$

such that for any sequence of input signals $\vec{x}_1, \cdots, \vec{x}_k \in (\mathbb{B}^{|\mathcal{I}|})^k$, the following condition holds:

$$[\![\mathcal{S}]\!](\vec{x}_1, \cdots, \vec{x}_k) = (\vec{y}_1, \cdots, \vec{y}_k) \text{ iff } \vec{s}_0 \xrightarrow{\vec{x}_1|\vec{y}_1} \vec{s}_1 \xrightarrow{\vec{x}_2|\vec{y}_2} \vec{s}_2 \xrightarrow{\vec{x}_3|\vec{y}_3} \vec{s}_3 \cdots \vec{s}_{k-1} \xrightarrow{\vec{x}_k|\vec{y}_k} \vec{s}_k.$$

Given an output $o \in \mathcal{O}$, we define $[\![\mathcal{S}]\!]_{\downarrow o}$ as a function such that for any sequence of input signals $\vec{x}_1, \cdots, \vec{x}_k \in (\mathbb{B}^{|\mathcal{I}|})^k$, $[\![\mathcal{S}]\!]_{\downarrow o}(\vec{x}_1, \cdots, \vec{x}_k)$ is the signal of the output o at the k-th clock cycle. Given a clock cycle $i \in [k]$, we denote by $\mathcal{S}_i$ the sub-circuit of the circuit $\mathcal{S}$ in which all the combinational circuits C_j for $j > i$ are removed. Thus, for any sequence of input signals $\vec{x}_1, \cdots, \vec{x}_k \in (\mathbb{B}^{|\mathcal{I}|})^k$ such that $[\![\mathcal{S}]\!](\vec{x}_1, \cdots, \vec{x}_k) = (\vec{y}_1, \cdots, \vec{y}_k)$, we have: $[\![\mathcal{S}_i]\!](\vec{x}_1, \cdots, \vec{x}_i) = (\vec{y}_1, \cdots, \vec{y}_i)$. Furthermore, $[\![\mathcal{S}_i]\!]_{\downarrow o}(\vec{x}_1, \cdots, \vec{x}_i)$ is the signal of the output o of the circuits $\mathcal{S}_i$ and $\mathcal{S}$ at the i-th clock cycle.

We remark that in practice, the combinational circuits C_i's in round-based hardware implementations of a cryptographic primitive are often similar (many of them are actually the same up to renaming of the vertices), because the internal rounds of a cryptographic primitive often perform similar computations. Furthermore, only partial signals of primary inputs $\mathcal{I}$ may be used in one clock cycle and only the signals of primary outputs $\mathcal{O}$ produced in the last clock cycle may be useful for the circuit functionality (in which case the signals of primary outputs $\mathcal{O}$ in the other clock cycles are useless for the circuit functionality). Our formalization is designed to be general.

2.3 Fault Injection Attacks

Fault injection attacks are a type of physical attacks that actively inject faults on some logic and/or memory gates during the execution of a cryptographic circuit and then statistically analyze the faulty primary outputs to deduce sensitive data such as the cryptographic key [Bak22]. Over the last two decades, various fault injection mechanisms have been proposed such as clock glitches [ADN+10, ESH+11, SHO19], underpowering [SGD08], voltage glitches [ZDCT13], electromagnetic pulses [DDRT12, DLM19, DLM21], and laser beams [SA03, RSDT13, CLFT14, SFG+16, DBC+18].

Clock glitch causes transient faults in circuits by tampering with a clock signal with glitches. Under the normal clock, the clock cycle is larger than the maximum path delay in combinational circuits, allowing full propagation of the signals so that the input signals to memory gates are stable before the next clock signal triggers the sampling process of the memory gates. In contrast, under a clock with glitches, some clock periods are shorter than the maximum path delay so the input signals to memory gates become unstable (i.e., only parts of input signals have reached). As a result, the memory gates may sample faulty results.

Underpowering and voltage glitches are similar to clock glitches except that underpowering lowers the supply voltage of the device throughout an entire execution while voltage glitches only lower the supply voltage for a limited period of time during an execution. In contrast to clock glitches that decrease clock periods, lowering supply voltage increases the maximum path delay in combinatorial circuits which also induces memory gates to sample faulty results.

Electromagnetic pulses induce currents in wire loops that are power and ground networks in integrated circuits. The induced current in a wire loop leads to a (negative or positive) voltage swing between the power and ground grid. A negative (resp. positive) voltage swings decreases (resp. increases) the clock and input signals to memory gates, often leading to reset (resp. set) of the corresponding memory gates, thus injecting faults on memory gates. A laser beam on a transistor produces a dense distribution of electron-hole pairs along the laser path, leading to a reduced voltage and eventually a temporary drift current. The temporary drift current can be used to alter the output signal of a (logic or memory) gate.

Clock glitches, underpowering and voltage glitches are non-invasive, as they do not require a modification of the targeted device, thus are considered as rather inexpensive. In contrast, electromagnetic pulses and laser beams are semi-invasive, allowing the adversary to inject localized faults, thus have higher precision than non-invasive attacks, but still at reasonable equipment and expertise requirement.

2.4 Countermeasures

Various countermeasures have been proposed to defend against fault injection attacks. For clock glitches, underpowering and voltage glitches, an alternative implementation of the circuit can be developed where signal path delays in combinatorial circuits are made independent of the sensitive data. For instance, delay components can be added to certain signal paths [GAS14, ELH+15], or combinational circuits can be reorganized [EWW16], so that the arrival time of all output signals of logic gates are independent of the sensitive data. However, such countermeasures fail to defend against electromagnetic pulses and laser beams.

Redundancy-based countermeasures are proposed to detect the presence of a fault. For instance, spatial redundancy recomputes the output multiple times in parallel [MSY06]; temporal redundancy recomputes the output multiple times consecutively [MSY06], and information redundancy leverages linear error code from coding theory [AMR+20]. Once a fault is detected, the output is omitted or the sensitive data is destroyed, with an error flag signal. However, such countermeasures are still vulnerable against advanced fault injection attacks such as Ineffective Fault Attack (IFA) [Cla07b] and Statistical Ineffective Fault Analysis (SIFA) [DEK+18]. The linear error-code based approach proposed in [AMR+20] was extended in [SRM20], which can correct faults to protect against IFA and SIFA.

In this work, we are interested in verifying redundancy based countermeasures including detection- and correction-based ones [MSY06, AMR+20, SRM20]. We do not consider countermeasures that make the arrival time of all output signals of logic gates independent of the sensitive data [GAS14, ELH+15], as they fail to defend against more advanced fault injection attacks.

3 The Fault-Resistance Verification Problem

In this section, inspired by the consolidated fault model [RBSG23], we first formalize the fault-resistance verification problem, and then present an illustrating example.

3.1 Problem Formulation

Fix a k-clock cycle circuit $\mathcal{S} = (\mathcal{I}, \mathcal{O}, \mathcal{R}, \vec{s}_0, \mathcal{C})$, where $\mathcal{C} = \{C_1, \cdots, C_k\}$ and $C_i = (V_i, I_i, O_i, E_i, \mathsf{g}_i)$ for each $i \in [k]$. We assume that $\mathcal{S}$ is a cryptographic circuit without deploying any countermeasures. Let $\mathcal{S}' = (\mathcal{I}, \mathcal{O}', \mathcal{R}', \vec{s}'_0, \mathcal{C}')$ be the protected counterpart of $\mathcal{S}$ using a detection-based or correction-based countermeasure [MSY06, AMR+20, SRM20], where $\mathcal{C}' = \{C'_1, \cdots, C'_k\}$ and $C'_i = (V'_i, I'_i, O'_i, E'_i, \mathsf{g}'_i)$ for each $i \in [k]$. We assume that $\mathcal{O}' = \mathcal{O} \cup \{o_{\texttt{flag}}\}$, where $o_{\texttt{flag}}$ is an error flag output indicating whether a fault was detected when the circuit $\mathcal{S}'$ adopts a detection-based countermeasure. If $\mathcal{S}'$ adopts a correction-based countermeasure, i.e., no error flag output is involved, for clarity, we assume that the error flag output $o_{\texttt{flag}}$ is added but is always $\texttt{0}$.

To formalize the fault-resistance verification problem, we first introduce some notations. We denote by $\mathbf{B}$ the blacklist of gates that are protected against fault injection attacks. Note that the blacklist $\mathbf{B}$ is configurable which may be empty as in [RBSS$^+$21]. It usually contains the gates used in the sub-circuits implementing a detection or correction mechanism, otherwise, the adversary can directly inject faults into them. It can be seen as a set of minimal vulnerable gates that should be protected. Note that the effects of faults injected on the other gates can be propagated into the gates in $\mathbf{B}$.

To model the effects of different fault injections, we introduce the following three fault types:

- bit-set fault τ_s: when injected on a gate, its output becomes `1`;
- bit-reset fault τ_r: when injected on a gate, its output becomes `0`;
- bit-flip fault τ_{bf}: when injected on a gate, its output is flipped, i.e., either from `1` to `0` or from `0` to `1`.

These fault types are able to capture all the effects of faults induced by both non-invasive fault injections (i.e., clock glitches, underpowering and voltage glitches) and semi-invasive fault injections (i.e., electromagnetic pulses and laser beams). We refer readers to [RBSG23] for the detailed discussion. We denote by $\mathcal{T} = \{\tau_s, \tau_r, \tau_{bf}\}$ the set of fault types.

A fault injection with fault type $\tau \in \mathcal{T}$ on a gate can be exactly characterized by replacing its associated function $\bullet$ with $\tau(\bullet)$, where

$$\tau(\bullet) := \begin{cases} \sqcap, & \text{if } \tau = \tau_s; \\ \sqcup, & \text{if } \tau = \tau_r; \\ \bar{\bullet}, & \text{if } \tau = \tau_{bf}. \end{cases}$$

To specify when, where and how a fault is injected, we introduce fault events.

Definition 3 (Fault event)**.** A *fault event* is given by $\mathsf{e}(\sigma, \beta, \tau)$, where

- $\sigma \in [k]$ specifies the clock cycle of the fault injection, namely, the fault injection occurs at the σ-th clock cycle;
- $\beta \in R'_{\sigma-1} \cup V'_\sigma \setminus (I'_\sigma \cup O'_\sigma)$ specifies the gate on which the fault is injected; we require that $\beta \notin \mathbf{B}$;
- $\tau \in \mathcal{T}$ specifies the fault type of the fault injection.

A fault event $\mathsf{e}(\sigma, \beta, \tau)$ yields the faulty circuit $\mathcal{S}'[\mathsf{e}(\sigma, \beta, \tau)] = (\mathcal{I}, \mathcal{O}', \mathcal{R}', \vec{s}'_0, \mathcal{C}'')$, where $\mathcal{C}'' = \{C''_1, \cdots, C''_k\}$, for each $i \in [k]$ and every $\beta' \in R'_{\sigma-1} \cup V'_\sigma \setminus (I'_\sigma \cup O'_\sigma)$,

- $C''_i := \begin{cases} (V'_i, I'_i, O'_i, E''_i, \mathsf{g}''_i), & \text{if } i = \sigma; \\ C'_i, & \text{if } i \neq \sigma; \end{cases}$
- $\mathsf{g}''_\sigma(\beta') := \begin{cases} \tau(\mathsf{g}'_\sigma(\beta)), & \text{if } \beta' = \beta; \\ \mathsf{g}'_\sigma(\beta), & \text{if } \beta' \neq \beta, \end{cases}$
- E''_i is obtained from E'_i by removing the incoming edges of β if $\tau \in \{\tau_s, \tau_r\}$.

Intuitively, the faulty circuit $\mathcal{S}'[\mathsf{e}(\sigma, \beta, \tau)]$ is the same as the circuit $\mathcal{S}'$ except that the function $\mathsf{g}'_\sigma(\beta)$ of the gate β is transiently replaced by $\tau(\mathsf{g}'_\sigma(\beta))$ in the σ-th clock cycle, while all the other gates at all the clock cycles remain the same. We denote by $\tau(\beta)$ the faulty counterpart of the gate β with fault type τ.

In practice, multiple fault events can occur simultaneously during the same clock cycle and/or consecutively in different clock cycles, allowing the adversary to conduct sophisticated fault injection attacks. To formalize this, we introduce fault vectors, as a generalization of fault events.

Definition 4 (Fault vector)**.** A *fault vector* $\mathsf{V}(\mathcal{S}', \mathbf{B}, T)$ is given by a (non-empty) set of fault events

$$\mathsf{V}(\mathcal{S}',\mathbf{B},T) := \left\{ \begin{array}{c|c} \mathsf{e}(\alpha_1,\beta_1,\tau_1),\cdots,\mathsf{e}(\alpha_m,\beta_m,\tau_m) & \begin{array}{c} \forall i,j\in[m].\alpha_i\in[k]\wedge\tau_i\in T\wedge \\ (i\neq j\wedge\alpha_i=\alpha_j \implies \beta_i\neq\beta_j) \end{array} \end{array} \right\}.$$

A fault can be injected to a gate *at most once* in the circuit $\mathcal{S}'$, but multiple faults can be injected to different gates, in the same or different clock cycles. Note that $\mathcal{S}'$ is unrolled with clock cycles where each physical gate in the original circuit is renamed in different clock cycles. As a result, different gates in a fault vector may correspond to the same physical gate in the original circuit, allowing us to capture persistent and permanent faults (called multi-cycle faults hereafter) using fault vectors.

For instance, consider a physical gate $\beta\notin\mathbf{B}$ in the original circuit. The unrolled counterpart $\mathcal{S}'$ consists of k versions $\{\beta_1,\cdots,\beta_k\}$ of the physical gate β, where the gate β_k denotes the physical gate β in the k-th clock cycle of the original circuit. A d-cycle fault on the physical gate β with fault type τ is captured by the set of fault vectors $\{\mathsf{V}_\alpha(\mathcal{S}',\mathbf{B},\{\tau\})\mid 0\leq\alpha<k\}$, where the fault vector $\mathsf{V}_\alpha(\mathcal{S}',\mathbf{B},\{\tau\})$ is defined as

$$\mathsf{V}_\alpha(\mathcal{S}',\mathbf{B},\{\tau\}) = \{\mathsf{e}(\alpha+1,\beta_{\alpha+1},\tau),\cdots,\mathsf{e}(\alpha+d',\beta_{\alpha+d'},\tau)\mid d'=\min(d,k-\alpha)\}.$$

A fault vector $\mathsf{V}(\mathcal{S}',\mathbf{B},T)$ on the circuit $\mathcal{S}'$ yields the faulty circuit $\mathcal{S}'[\mathsf{V}(\mathcal{S}',\mathbf{B},T)]$, which is obtained by iteratively applying fault events in $\mathsf{V}(\mathcal{S}',\mathbf{B},T)$, i.e.,

$$\mathcal{S}'[\mathsf{V}(\mathcal{S}',\mathbf{B},T)] := \mathcal{S}'[\mathsf{e}(\alpha_1,\beta_1,\tau_1)]\cdots[\mathsf{e}(\alpha_m,\beta_m,\tau_m)].$$

Definition 5 (Effectiveness of fault vectors)**.** A fault vector $\mathsf{V}(\mathcal{S}',\mathbf{B},T)$ is *effective* if there exists a sequence of primary inputs $(\vec{x}_1,\cdots,\vec{x}_k)$ such that the sequences of primary outputs $[\![\mathcal{S}']\!](\vec{x}_1,\cdots,\vec{x}_k)$ and $[\![\mathcal{S}'[\mathsf{V}(\mathcal{S}',\mathbf{B},T)]]\!](\vec{x}_1,\cdots,\vec{x}_k)$ differ at some clock cycle which is before the clock cycle when the error flag output $o_{\mathtt{flag}}$ differs.

Intuitively, an effective fault vector breaks the functional equivalence between $\mathcal{S}$ and $\mathcal{S}'$ and the fault is *not* successfully detected (i.e., setting the error flag output $o_{\mathtt{flag}}$). Note that there are two possible cases for an ineffective fault vector: either $[\![\mathcal{S}']\!](\vec{x}_1,\cdots,\vec{x}_k)$ and $[\![\mathcal{S}'[\mathsf{V}(\mathcal{S}',\mathbf{B},T)]]\!](\vec{x}_1,\cdots,\vec{x}_k)$ are the same for each sequence of primary inputs $(\vec{x}_1,\cdots,\vec{x}_k)$ or the fault is successfully detected in time.

Hereafter, $\sharp\mathtt{Clk}(\mathsf{V}(\mathcal{S}',\mathbf{B},T))$ denotes the cardinality of the set $\{\alpha_1,\cdots,\alpha_m\}$, i.e., the number of clock cycles when fault events can occur, and by $\mathtt{MaxFEpClk}(\mathsf{V}(\mathcal{S}',\mathbf{B},T))$ the maximum number of fault events per clock cycle, i.e., $\max_{\alpha\in[k]}|\{e(\alpha,\beta,\tau)\in\mathsf{V}(\mathcal{S}',\mathbf{B},T)\}|$. Inspired by the consolidated fault model [RBSG23], we introduce the security model of fault-resistance which characterizes the capabilities of the adversary.

Definition 6 (Fault-resistance model)**.** A *fault-resistance model* is given by $\zeta(\mathtt{n}_e,\mathtt{n}_c,T,\ell)$, where

- $\mathtt{n}_e$ is the maximum number of fault events per clock cycle;
- $\mathtt{n}_c$ is the maximum number of clock cycles in which fault events can occur;
- $T\subseteq\mathcal{T}$ specifies the allowed fault types; and
- $\ell\in\{\mathtt{l},\mathtt{m},\mathtt{lm}\}$ defines vulnerable gates: $\mathtt{l}$ for logic gates in combinational circuits, $\mathtt{m}$ for memory gates and $\mathtt{lm}$ for both logic and memory gates.

For instance, the fault-resistance model $\zeta(\mathtt{n}_e,k,\mathcal{T},\mathtt{lm})$ gives the strongest capability to the adversary for a large $\mathtt{n}_e$ allowing the adversary to inject faults to all the gates simultaneously at any clock cycle (except for those protected in the blacklist $\mathbf{B}$). The fault-resistance model $\zeta(1,1,\{\tau_{bf}\},\mathtt{l})$ only allows the adversary to choose one logic gate to inject a bit-flip fault in one chosen clock cycle. Formally, $\zeta(\mathtt{n}_e,\mathtt{n}_c,T,\ell)$ defines the set $[\![\zeta(\mathtt{n}_e,\mathtt{n}_c,T,\ell)]\!]$ of possible fault vectors that can be conducted by the adversary, i.e., $[\![\zeta(\mathtt{n}_e,\mathtt{n}_c,T,\ell)]\!]$ is

$$\{\mathsf{V}(\mathcal{S}',\mathbf{B}_\ell,T)\mid\mathtt{MaxFEpClk}(\mathsf{V}(\mathcal{S}',\mathbf{B}_\ell,T))\leq n_e\wedge\sharp\mathtt{Clk}(\mathsf{V}(\mathcal{S}',\mathbf{B}_\ell,T))\leq\mathtt{n}_c,\}$$

where $\mathbf{B}_\ell := \begin{cases} \mathbf{B}, & \text{if } \ell = \mathtt{lm}; \\ \mathbf{B} \cup \mathcal{R}, & \text{if } \ell = \mathtt{l}; \\ \mathbf{B} \cup \bigcup_{i\in[k]} V'_i \setminus (I'_i \cup O'_i), & \text{if } \ell = \mathtt{m}. \end{cases}$

The circuit $\mathcal{S}'$ is *fault-resistant* w.r.t. a blacklist $\mathbf{B}$ and a fault-resistance model $\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$, denoted by $\langle \mathcal{S}', \mathbf{B}\rangle \models \zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$, if all the fault vectors $\mathsf{V}(\mathcal{S}', \mathbf{B}, T) \in [\![\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)]\!]$ are ineffective on the circuit $\mathcal{S}'$.

Definition 7 (Fault-resistance verification problem)**.** The *fault-resistance verification problem* is to determine if $\langle \mathcal{S}', \mathbf{B}\rangle \models \zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$, and in particular, if $\langle \mathcal{S}', \mathbf{B}\rangle \models \zeta(\mathtt{n}_e, \mathtt{n}_c, \mathcal{T}, \mathtt{lm})$.

The definition of our fault-resistance covers all the feasible fault vectors $\mathsf{V}(\mathcal{S}', \mathbf{B}, T) \in [\![\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)]\!]$, allowing us to verify fault-resistance against both multi-cycle faults and random faults by choosing proper values for the parameters $\mathtt{n}_e$ and $\mathtt{n}_c$. For instance, it suffices to set $\mathtt{n}_e = n \cdot \min(d, m)$ and $\mathtt{n}_c = \max(m \cdot d, k)$ for n number of d-cycle (random) faults per clock cycle in at most m clock cycles for $m \leq k$ (i.e., up to $n \cdot m$ number d-cycle faults in total).

Proposition 1. *If $\langle \mathcal{S}', \mathbf{B}\rangle \models \zeta(\mathtt{n}_e, \mathtt{n}_c, \mathcal{T}, \mathtt{lm})$, then $\langle \mathcal{S}', \mathbf{B}\rangle \models \zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$ for any $T \subseteq \mathcal{T}$ and any $\ell \in \{\mathtt{l}, \mathtt{m}, \mathtt{lm}\}$.*

Theorem 1. *The problem of determining whether a circuit $\mathcal{S}'$ is fault-resistant is coNP-complete.*

Proof. We show that the problem of determining whether a circuit $\mathcal{S}'$ is *not* fault-resistant is NP-complete.

To show that this problem is in NP, for a given fault-resistance model $\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$, we first non-deterministically guess a sequence of primary inputs $(\vec{x}_1, \cdots, \vec{x}_k)$ and a fault vector $\mathsf{V}(\mathcal{S}', \mathbf{B}, T) \in [\![\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)]\!]$, then construct the faulty circuit $\mathcal{S}'[\mathsf{V}(\mathcal{S}', \mathbf{B}, T)]$ in polynomial time by traversing and manipulating gates in the circuit $\mathcal{S}'$, and finally check if the sequences of primary outputs $[\![\mathcal{S}']\!](\vec{x}_1, \cdots, \vec{x}_k)$ and $[\![\mathcal{S}'[\mathsf{V}(\mathcal{S}', \mathbf{B}, T)]]\!](\vec{x}_1, \cdots, \vec{x}_k)$ differ at some clock cycle before the error flag output $o_{\mathtt{flag}}$ differs in polynomial time by explicitly computing the sequences of primary outputs using the sequence of primary inputs $(\vec{x}_1, \cdots, \vec{x}_k)$. If yes, then $\langle \mathcal{S}', \mathbf{B}\rangle \not\models \zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$.

The NP-hardness is shown by reducing from the SAT problem[2]. Let C_φ be a combinational circuit representing a Boolean formula φ, where the inputs of C_φ are the Boolean variables of φ (say $x_1, \cdots, x_m$), and the output indicates the result of φ. We create a circuit $\mathcal{S}' = (\mathcal{I}, \mathcal{O}, \mathcal{R}, \vec{s}_0, \mathcal{C})$ as shown in Fig. 1, where

- $\mathcal{I} = \{x_1, \cdots, x_m\}$ is the set of inputs of the circuit C_φ;
- $\mathcal{O}$ is the set $\{o_i, o_{\mathtt{flag}} \mid 1 \leq i \leq 2\mathtt{n}_e + 1\}$;
- $\mathcal{R} = R_1 \cup R_2$, with $R_1 = \{r_i \mid 1 \leq i \leq 2\mathtt{n}_e + 1\}$ and $R_2 = \{r'_i \mid 1 \leq i \leq 2\mathtt{n}_e + 1\}$;
- $\vec{s}_0$ is a vector consisting of $\mathtt{0}$;
- $\mathcal{C} = \{C_1, C_2, C_3\}$, such that
  - C_1 comprises $2\mathtt{n}_e + 1$ copies of the circuit C_φ: all the copies share the same inputs $\mathcal{I}$, the output of the i-th copy is connected to r_i, and the output $o_{\mathtt{flag}}$ is always $\mathtt{0}$;
  - C_2 outputs signals of the memory gates R_1 and stores them into the memory gates R_2 again, checks if $1 \leq \sum_{i=1}^{2\mathtt{n}_e+1} r_i \leq \mathtt{n}_e$, and the output $o_{\mathtt{flag}}$ is $\mathtt{1}$ iff $1 \leq \sum_{i=1}^{2\mathtt{n}_e+1} r_i \leq \mathtt{n}_e$;
  - C_3 checks whether $1 \leq \sum_{i=1}^{2\mathtt{n}_e+1} r'_i \leq 2\mathtt{n}_e$, and the output $o_{\mathtt{flag}}$ is $\mathtt{1}$ iff $1 \leq \sum_{i=1}^{2\mathtt{n}_e+1} r'_i \leq 2\mathtt{n}_e$.

[2] A Boolean formula is satisfiable iff there is an assignment of the variables under which the formula evaluates to true. The SAT problem is to determine whether a Boolean formula is satisfiable or not.

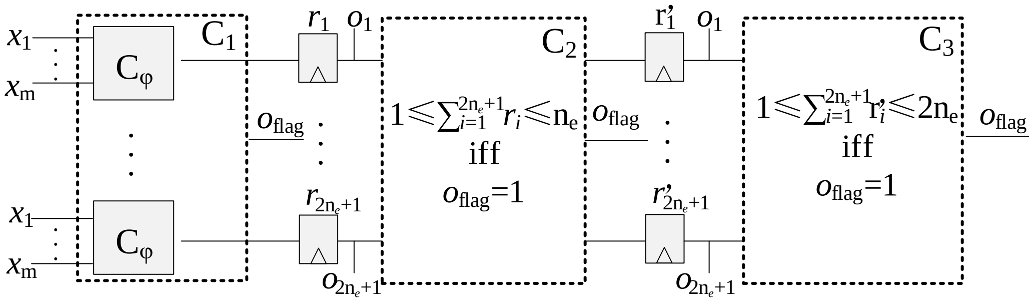


Figure 1: The circuit $\mathcal{S}'$ for NP-hardness.

Table 1: Truth table of the S-box in the block cipher RECTANGLE.

$\vec{x}$	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
$S(\vec{x})$	0110	0101	1100	1010	0001	1110	0111	1001	1011	0000	0011	1101	1000	1111	0100	0010
$S[\mathtt{s7},\tau_s](\vec{x})$	**1**110	**1**101	**0**100	**0**010	0001	**0**110	**1**111	1001	1011	**1**000	0011	**0**101	**0**000	**0**111	**1**100	**1**010
$S[\mathtt{s7},\tau_r](\vec{x})$	0110	0101	1100	1010	**1**001	1110	0111	**0**001	**0**011	0000	**1**011	1101	1000	1111	0100	0010
$S[\mathtt{s7},\tau_{bf}](\vec{x})$	**1**110	**1**101	**0**100	**0**010	**1**001	**0**110	**1**111	**0**001	**0**011	**1**000	**0**011	**0**101	**0**000	**0**111	**1**100	**1**010
$S[\mathtt{s9},\tau_s](\vec{x})$	0**0**1**1**	0101	110**1**	101**1**	0001	111**1**	0111	1001	1011	0**1**0**1**	0011	1101	100**1**	1111	0**0**01	0**1**1**1**
$S[\mathtt{s9},\tau_r](\vec{x})$	0110	0**0**0**0**	1100	1010	000**0**	1110	0**0**1**0**	1**1**0**0**	1**1**1**0**	0000	001**0**	110**0**	1000	111**0**	0100	0010
$S[\mathtt{s9},\tau_{bf}](\vec{x})$	0**0**1**1**	0**0**0**0**	110**1**	101**1**	000**0**	111**1**	0**0**1**0**	1**1**0**0**	1**1**1**0**	0**1**0**1**	001**0**	110**0**	100**1**	111**0**	0**0**01	0**1**1**1**

Claim. *The circuit $\mathcal{S}'$ is not fault-resistant w.r.t. the blacklist $\mathbf{B} = \emptyset$ and the fault-resistance model $\zeta(\mathtt{n}_e, 1, \{\tau_{bf}\}, \mathtt{m})$ iff the Boolean formula φ is satisfiable.*

($\Leftarrow$) Suppose φ is satisfiable. Let $\vec{x}$ be the satisfying assignment of φ. Obviously, under the primary inputs $\vec{x}$, the output $o_{\mathtt{flag}}$ is $\mathtt{0}$ and the outputs $\{o_i \mid 1 \leq i \leq 2\mathtt{n}_e + 1\}$ are $\mathtt{1}$ in all the clock cycles. Consider the fault event $\mathsf{e}(2, r_1, \tau_{bf})$. Along the sequence of primary outputs $[\![\mathcal{S}'[\mathsf{e}(2, r_1, \tau_{bf})]]\!](\vec{x})$, the output $o_{\mathtt{flag}}$ is $\mathtt{0}$ at the first two clock cycles and becomes $\mathtt{1}$ at the 3-rd clock cycle. However, the output o_1 differs in $[\![\mathcal{S}']\!](\vec{x})$ and $[\![\mathcal{S}'[\mathsf{e}(2, r_1, \tau_{bf})]]\!](\vec{x})$ at the 2nd clock cycle due to the bit-flip fault injection on the memory gate r_1. Thus, $\langle \mathcal{S}', \emptyset \rangle \not\models \zeta(\mathtt{n}_e, 1, \{\tau_{bf}\}, \mathtt{m})$, i.e., the circuit $\mathcal{S}'$ is not fault-resistant w.r.t. the blacklist $\mathbf{B} = \emptyset$ and the fault-resistance model $\zeta(\mathtt{n}_e, 1, \{\tau_{bf}\}, \mathtt{m})$.

($\Rightarrow$) Suppose φ is unsatisfiable. Obviously, under any primary inputs $\vec{x}$, all the primary outputs $\{o_{\mathtt{flag}}, o_i \mid 1 \leq i \leq 2\mathtt{n}_e + 1\}$ are $\mathtt{0}$ in all the clock cycles. For any fault vector $\mathsf{V}(\mathcal{S}', \mathbf{B}, T) \in [\![\zeta(\mathtt{n}_e, 1, \{\tau_{bf}\}, \mathtt{m})]\!]$, at most $\mathtt{n}_e$ memory gates can be bit-flipped in one single clock cycle. If some memory gates in R_1 are bit-flipped at the 2nd clock cycle, then the output $o_{\mathtt{flag}}$ is $\mathtt{1}$ at the 2nd clock cycle, successfully detecting the fault injection. If no memory gates of R_1 are bit-flipped at the 2nd clock cycle and some memory gates in R_2 are bit-flipped at the 3rd clock cycle, the primary outputs $\{o_i \mid 1 \leq i \leq 2\mathtt{n}_e + 1\}$ are $\mathtt{0}$ at the 2nd clock cycle, $o_{\mathtt{flag}}$ is $\mathtt{1}$ at the 3rd clock cycle successfully detecting the fault injection, although some primary outputs of $\{o_i \mid 1 \leq i \leq 2\mathtt{n}_e + 1\}$ become $\mathtt{1}$ at the 3-rd clock cycle. Hence $\langle \mathcal{S}', \emptyset \rangle \models \zeta(\mathtt{n}_e, 1, \{\tau_{bf}\}, \mathtt{m})$, i.e., the circuit $\mathcal{S}'$ is fault-resistant w.r.t. the blacklist $\mathbf{B} = \emptyset$ and the fault-resistance model $\zeta(\mathtt{n}_e, 1, \{\tau_{bf}\}, \mathtt{m})$. □

3.2 An Illustrating Example

Consider the S-box used in the cipher RECTANGLE [ZBL+15], which is a 4-bit to 4-bit mapping $S : \mathbb{B}^4 \to \mathbb{B}^4$ given in Table 1 (the top two rows). It can be implemented in a combinational circuit as shown in Fig. 2 (grey-area). It has four 1-bit inputs $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$ denoting the binary representation of the 4-bit input $\vec{x}$, and four 1-bit outputs $\{\mathtt{w}, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$ denoting the binary representation of the 4-bit output $S(\vec{x})$, where $\mathtt{a}$ and $\mathtt{w}$ are the most

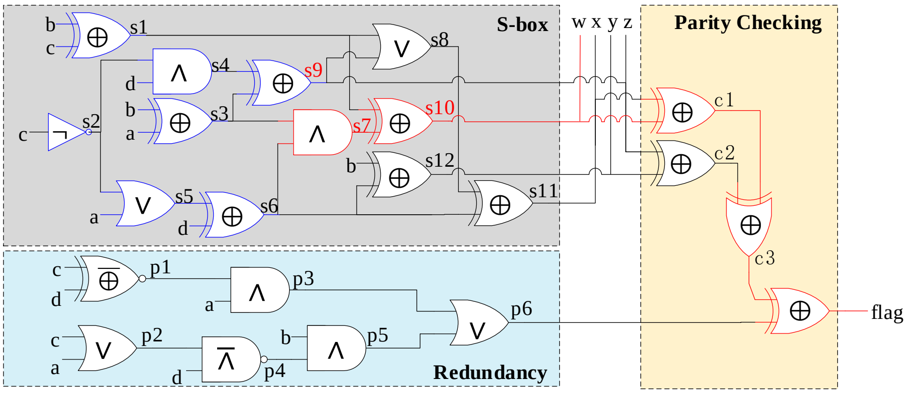


Figure 2: Circuit representation of the illustrating example.

```
RECTANGLE S-box
Input: a,b,c,d
Output: w,x,y,z,flag
 s1  = b ⊕ c
 s2  = ¬ c
 s3  = b ⊕ a
 s4  = s2 ∧ d
 s5  = s2 ∨ a
 z   = s3 ⊕ s4
 s6  = s5 ⊕ d
 s7  = s3 ∧ s6
 s8  = s1 ∨ z
 w   = s1 ⊕ s7
 x   = s8 ⊕ s6
 y   = b ⊕ s6
 p1  = c ⊕̄ d
 p2  = a ∨ c
 p3  = a ∧ p1
 p4  = p2 ⊼ d
 p5  = p4 ∧ b
 p6  = p3 ∨ p5
 c1  = w ⊕ x
 c2  = y ⊕ z
 c3  = c1 ⊕ c2
 flag = c3 ⊕ p6
```

Figure 3: Pseudo-code of the illustrating example.

significant bits. The values of the inputs `a,b,c` and `d` depend upon the secret key. The corresponding pseudo-code of the illustrating example is given in Fig. 3, where the left two columns implements the function of the S-box and the right column implements a single-bit parity protection mechanism.

If a fault with fault type τ is injected on the gate `s7` (i.e., the gate whose output is `s7`), its function $\mathtt{g}(\mathtt{s7})$ is changed from $\wedge$ to $\tau(\wedge)$. As highlighted in red color in Fig. 2, the effect of this fault will be propagated to the output `w`. We denote by $S[\mathtt{s7}, \tau]$ the faulty S-box, given in Table 1 for each $\tau \in \mathcal{T}$, where the faulty output is highlighted in **bold**. Since the distribution of the XOR-difference $S[\mathtt{s7}, \tau](\vec{x}) \oplus S(\vec{x})$ is biased, the adversary can narrow down the solutions for $\vec{x}$ according to the value of $S[\mathtt{s7}, \tau](\vec{x}) \oplus S(\vec{x})$ which is known to the adversary. Finally, the adversary solves $\vec{x}$ uniquely, based on which a round key can be obtained (Details refer to [Bak22]).

To thwart single-bit fault injection attacks, one may adopt a single-bit parity protection mechanism [KKG03, BBK+03], as shown in Fig. 2. The sub-circuit in the blue-area is a redundancy part which computes the Hamming weight of the output of the S-box from the input but independent on the sub-circuit in the grey-area, i.e., `p6`. The sub-circuit in the yellow-area checks the parity of the Hamming weights of $S(\vec{x})$ computed in two independent sub-circuits, i.e., $\mathtt{flag} = \mathtt{p6} \oplus \mathtt{w} \oplus \mathtt{x} \oplus \mathtt{y} \oplus \mathtt{z}$. If no faults occur, `flag` is `0`.

Re-consider the fault injected on the gate `s7`. We can see that either `flag` becomes `1`, i.e., this fault injection can be successfully detected, or the outputs of $S[\mathtt{s7}, \tau](\vec{x})$ and $S(\vec{x})$ are the same, thus the fault injection is ineffective.

However, the entire circuit is still vulnerable against single-bit fault injection attacks, as one single-bit fault injection can yield an even number of faulty output bits so that the Hamming weight of the faulty output remains the same. For instance, the fault injection on the gate `s9` will affect both the outputs `x` and `z`. As shown in Table 1, the fault injection cannot be successfully detected if one of the following items holds:

- $\tau = \tau_s \wedge \vec{x} \in \{0000, 1001, 1110, 1111\}$,

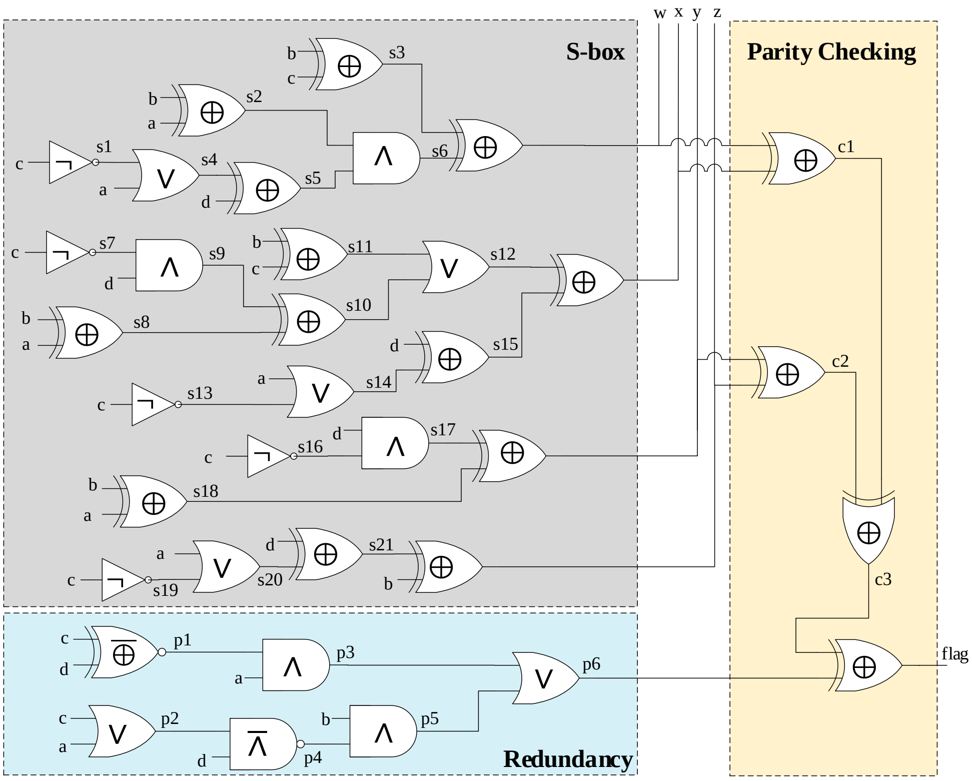


Figure 4: Circuit representation of the revised illustrating example.

- $\tau = \tau_r \wedge \vec{x} \in \{0001, 0110, 0111, 1000\}$,
- $\tau = \tau_{bf} \wedge \vec{x} \in \{0000, 0001, 0110, 0111, 1000, 1001, 1110, 1111\}$.

It is fault-resistant against single-bit fault injection attacks when the blacklist **B** includes all the logic gates in the parity checking (i.e., yellow-area) and all the logic gates in the S-box (i.e., grey-area) whose out-degree is larger than 2 (highlighted in blue color in Fig. 2). This issue also could be avoided by leveraging the independence property defined by [AMR+20], to ensure an n-bit fault injection attack only affects at most n output bits, at the cost of the circuit size.

```
RECTANGLE S-box
Input: a,b,c,d
Output: w,x,y,z,flag
 s1 = ¬ c
 s2 = a ⊕ b
 s3 = b ⊕ c
 s4 = s1 ∨ a
 s5 = s4 ⊕ d
 s6 = s2 ∧ s5
 s7 = ¬ c
 s8 = a ⊕ b
 s9 = s7 ∧ d
 s10 = s8 ⊕ s9
 s11 = b ⊕ c
 s12 = s10 ∨ s11
 s13 = ¬ c
 s14 = a ∨ s13
 s15 = d ⊕ s14
 s16 = ¬ c
 s17 = d ∧ s16
 s18 = a ⊕ b
 s19 = ¬ c
 s20 = a ∨ s19
 s21 = d ⊕ s20
 w = s3 ⊕ s6
 x = s12 ⊕ s15
 y = s17 ⊕ s18
 z = s12 ⊕ b
 p1 = c ⊕̅ d
 p2 = a ∨ c
 p3 = a ∧ p1
 p4 = p2 ∧̅ d
 p5 = p4 ∧ b
 p6 = p3 ∨ p5
 c1 = w ⊕ x
 c2 = y ⊕ z
 c3 = c1 ⊕ c2
 flag = c3 ⊕ p6
```

Figure 5: Pseudo-code of the revised illustrating example.

Revised implementation. The circuit representation of the revised implementation of the RECTANGLE S-box is shown in Fig. 4 and its pseudo-code is shown in Fig. 5, following the independence property defined by [AMR+20].

This circuit is fault-resistant against single-bit fault injection attacks when the blacklist **B** includes *only* the logic gates in the parity checking. We can observe that any fault injection on one single logic gate in the redundancy part does not change any of the outputs $\{\mathtt{w}, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$, any fault injection on one single logic gate in the S-box part only change one of the outputs $\{\mathtt{w}, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$ and also changes the error flag output `flag`. Thus, the revised implementation is fault-resistant w.r.t. the blacklist **B** and the fault-resistance model $\zeta(1, 1, \mathcal{T}, \mathtt{1})$, where **B** only contains the logic gates in the parity checking.

4 SAT-based Formal Verification

We propose an SAT-based countermeasure verification approach, which reduces the fault-resistance verification problems to SAT solving.

4.1 Overview

An overview of our approach is depicted in Fig. 6. Given a circuit $\mathcal{S}$ (without any countermeasures), a protected circuit $\mathcal{S}'$ (i.e., $\mathcal{S}$ with a countermeasure), a blacklist **B** of gates on which faults cannot be injected, and a fault-resistance model $\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$, FIRMER outputs a report on whether the circuit $\mathcal{S}'$ is fault-resistant.

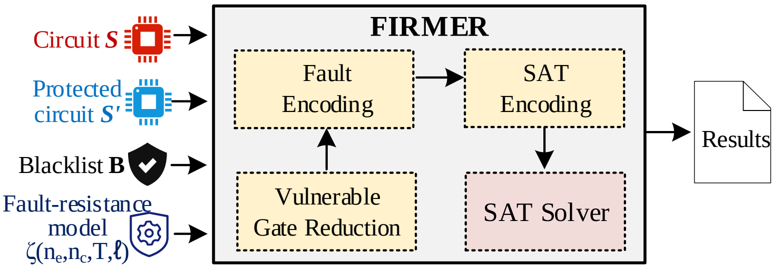


Figure 6: Framework of our verification approach.

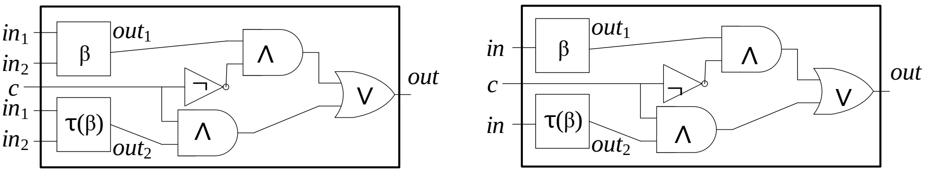
FIRMER consists of three key components: vulnerable gate reduction, fault encoding and SAT encoding. The vulnerable gate reduction safely reduces the number of vulnerable gates, thus reducing the size of the resulting Boolean formulas and improving efficiency. The fault encoding replaces each vulnerable gate with a gadget (i.e., sub-circuit) with additional primary inputs controlling whether a fault is injected and selecting a fault type. The SAT encoding is an extension of the one for checking functional equivalence, where (i) the maximum number of fault events per clock cycle and the maximum number of clock cycles in which fault events can occur are both expressed by constraints over control inputs, and (ii) a constraint on the error flag output is added.

Below, we present the details of our fault encoding method, SAT encoding method and vulnerable gate reduction

4.2 Fault Encoding

Gadgets. To encode a fault injection on a gate β with fault type $\tau \in \mathcal{T}$ and gate function $\mathtt{g}(\beta) = \bullet$, we define a gadget $G_{\beta,\tau}$ shown in Fig. 7. Note that $\tau(\beta)$ denotes the faulty counterpart of the gate β w.r.t. the fault type τ, i.e., $\mathtt{g}(\tau(\beta)) = \tau(\bullet)$. Indeed, the gadget $G_{\beta,\tau}$ for a binary gate β defines a Boolean formula $[\![G_{\beta,\tau}]\!]$ with

$$[\![G_{\beta,\tau}]\!](in_1, in_2, c) = c \; ? \; (in_1 \diamond in_2) : (in_1 \bullet in_2),$$



Gadget $G_{\beta,\tau}$ for a binary gate β

Gadget $G_{\beta,\tau}$ for a unary gate β

Figure 7: Gadgets for encoding one fault type.

where $\diamond = \tau(\bullet)$, and c is a *control input* indicating whether a fault is injected or not. Namely, $G_{\beta,\tau}$ is equivalent to the faulty gate $\tau(\beta)$ if $c = \mathtt{1}$, otherwise $G_{\beta,\tau}$ is equivalent to the original gate β. Note that the incoming edges of $\tau(\beta)$ should be omitted if $\tau \in \{\tau_s, \tau_r\}$. The gadget $G_{\beta,\tau}$ for a unary gate β is defined similarly as:

$$[\![G_{\beta,\tau}]\!](in, c) = c \ ?\ (\diamond in) : (\bullet in).$$

We now generalize the gadget definition to accommodate different fault types $\mathcal{T} = \{\tau_s, \tau_r, \tau_{bf}\}$. Besides a control input c, selection inputs (b_1, b_2) are introduced to choose fault types. The gadget $G_{\beta,\mathcal{T}}$ for a binary logic gate β defines a Boolean formula $[\![G_{\beta,\mathcal{T}}]\!]$ such that

$$[\![G_{\beta,\mathcal{T}}]\!](in_1, in_2, c, b_1, b_2) = c \ ?\ \big(b_1 \ ?\ (b_2 \ ?\ (in_1 \diamond in_2) : (in_1 \dagger in_2)) : (in_1 \ddagger in_2)\big) : (in_1 \bullet in_2),$$

where $\diamond = \tau_s(\bullet)$, $\dagger = \tau_r(\bullet)$ and $\ddagger = \tau_{bf}(\bullet)$. Intuitively,

- $c = \mathtt{0}$ means that no fault is injected, i.e., $G_{\beta,\mathcal{T}}$ is equivalent to the gate β;
- $c = \mathtt{1}$ means that a fault is injected. Moreover, the selection inputs (b_1, b_2) are defined as:
  - if $b_1 = b_2 = \mathtt{1}$, then $G_{\beta,\mathcal{T}}$ becomes the faulty logic gate $\tau_s(\beta)$;
  - if $b_1 = \mathtt{1}$ and $b_2 = \mathtt{0}$, then $G_{\beta,\mathcal{T}}$ becomes the faulty logic gate $\tau_r(\beta)$;
  - if $b_1 = \mathtt{0}$, then $G_{\beta,\mathcal{T}}$ becomes to the faulty logic gate $\tau_{bf}(\beta)$;

The gadget $G_{\beta,\mathcal{T}}$ for a unary gate β can be defined as the Boolean formula $[\![G_{\beta,\mathcal{T}}]\!]$ such that

$$[\![G_{\beta,\mathcal{T}}]\!](in, c, b_1, b_2) = c \ ?\ \big(b_1 \ ?\ (b_2 \ ?\ (\diamond in) : (\dagger in)) : (\ddagger in)\big) : (\bullet in).$$

For a subset of fault types $T = \{\tau_1, \tau_2\} \subset \mathcal{T}$, the gadget $G_{\beta,T}$ for a binary or unary gate β can be defined accordingly such that

$$[\![G_{\beta,T}]\!](in_1, in_2, c, b) = c \ ?\ \big(b \ ?\ (in_1 \diamond in_2) : (in_1 \dagger in_2)\big) : (in_1 \bullet in_2),$$
$$[\![G_{\beta,T}]\!](in, c, b) = c \ ?\ \big(b \ ?\ (\diamond in) : (\dagger in)\big) : (\bullet in),$$

where $\diamond = \tau_1(\bullet)$ and $\dagger = \tau_2(\bullet)$.

We remark that the faulty counterpart $\tau(\beta)$ of a register β is implemented by adding a logic gate so that no additional registers are introduced. More specifically, $\tau_s(\beta)$ (resp. $\tau_r(\beta)$) is a constant logic gate that always outputs the signal `1` (resp. `0`), and $\tau_{bf}(\beta)$ is a `not` logic gate with the incoming edge from the output of the register β.

Conditionally-controlled faulty circuits. From the protected circuit $\mathcal{S}'$, we construct a conditionally-controlled faulty circuit $\mathcal{S}''$, where each vulnerable gate is replaced by a gadget defined above.

Fix a fault-resistance model $\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$. Assume that the control input c and the set of selection inputs of each gadget $G_{\beta,T}$ are distinct and different from the ones used in the circuit $\mathcal{S}'$. We define the conditionally-controlled faulty circuit $\mathcal{S}''$ w.r.t. $\mathbf{B}$ and $\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$ as

$$\mathcal{S}'[\mathbf{B}, \zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)] := (\mathcal{I} \uplus \mathcal{I}', \mathcal{O}', \mathcal{R}', \vec{s}'_0, \mathcal{C}''),$$

where $\mathcal{I}' = \bigcup_{i \in [k]} I''_i$ and $\mathcal{C}'' = \{C''_1, \cdots, C''_k\}$.

For every $i \in [k]$, the circuit $C''_i = (V'_i \uplus V''_i, I'_i \uplus I''_i, O'_i, E'_i \uplus E''_i, \mathsf{g}''_i)$ is obtained from the combinational circuit C'_i as follows:

> For every gate $\beta \in R'_{i-1} \cup V'_i \setminus (I'_i \cup O'_i)$, if $\beta \notin \mathbf{B}_\ell$, then β is replaced by the gadget $G_{\beta,T}$, the control and selection inputs of the gadget $G_{\beta,T}$ are added into I''_i, the gates and edges of $G_{\beta,T}$ are added into V''_i and E''_i respectively, the mapping g'_i is expanded to g''_i accordingly.

Intuitively, a fault vector $\mathsf{V}(\mathcal{S}', \mathbf{B}, T) \in [\![\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)]\!]$ is encoded as a sequence $(\vec{b}_1, \cdots, \vec{b}_k)$ of the primary inputs $\mathcal{I}'$ for controlling fault types such that $\mathsf{e}(\alpha, \beta, \tau) \in \mathsf{V}(\mathcal{S}', \mathbf{B}, T)$ iff the gadget $G_{\beta,T}$ is equivalent to the faulty gate $\tau(\beta)$ under the primary inputs $\vec{b}_\alpha$, i.e., the control input of $G_{\beta,T}$ is $\mathtt{1}$ and the selection inputs of $G_{\beta,T}$ choose $\tau(\beta)$. We note that if $\mathsf{e}(\alpha, \beta, \tau) \notin \mathsf{V}(\mathcal{S}', \mathbf{B}, T)$ for any $\tau \in T$, then the gadget $G_{\beta,T}$ is equivalent to the original gate β under the primary inputs $\vec{b}_\alpha$, i.e., the control input of the gadget $G_{\beta,T}$ is the signal $\mathtt{0}$.

We say that a fault vector $\mathsf{V}(\mathcal{S}', \mathbf{B}, T)$ and a sequence $(\vec{b}_1, \cdots, \vec{b}_k)$ of the primary inputs $\mathcal{I}'$ are *compatible* if the sequence $(\vec{b}_1, \cdots, \vec{b}_k)$ encodes the fault vector $\mathsf{V}(\mathcal{S}', \mathbf{B}, T)$. Note that a sequence $(\vec{b}_1, \cdots, \vec{b}_k)$ of the primary inputs $\mathcal{I}'$ determines a unique compatible fault vector $\mathsf{V}(\mathcal{S}', \mathbf{B}, T)$, but a fault vector $\mathsf{V}(\mathcal{S}', \mathbf{B}, T)$ determines a unique compatible sequence $(\vec{b}_1, \cdots, \vec{b}_k)$ of the primary inputs $\mathcal{I}'$ *only* if $T \subset \mathcal{T}$, because $G_{\beta,\mathcal{T}}$ is equivalent to the faulty logic gate $\tau_{bf}(\beta)$ if $b_1 = \mathtt{0}$ no matter the value of b_2. Thus, we can get:

Proposition 2. *The number of gates of the circuit $\mathcal{S}''$ (i.e., $\mathcal{S}'[\mathbf{B}, \zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)]$) is at most $6|T|$ times than that of the circuit $\mathcal{S}'$, and the following statements hold:*

1. *for each fault vector $\mathsf{V}(\mathcal{S}', \mathbf{B}, T) \in [\![\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)]\!]$, there exists a compatible sequence $(\vec{b}_1, \cdots, \vec{b}_k)$ of the primary inputs $\mathcal{I}'$ such that for each sequence $(\vec{x}_1, \cdots, \vec{x}_k)$ of primary inputs $\mathcal{I}$,*

$$[\![\mathcal{S}'[\mathsf{V}(\mathcal{S}', \mathbf{B}, T)]]\!](\vec{x}_1, \cdots, \vec{x}_k) = [\![\mathcal{S}'']\!]((\vec{x}_1, \vec{b}_1), \cdots, (\vec{x}_k, \vec{b}_k));$$

2. *for each sequence $(\vec{b}_1, \cdots, \vec{b}_k)$ of the primary inputs $\mathcal{I}'$, there exists a unique compatible fault vector $\mathsf{V}(\mathcal{S}', \mathbf{B}, T) \in [\![\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)]\!]$ such that for each sequence $(\vec{x}_1, \cdots, \vec{x}_k)$ of primary inputs $\mathcal{I}$,*

$$[\![\mathcal{S}'[\mathsf{V}(\mathcal{S}', \mathbf{B}, T)]]\!](\vec{x}_1, \cdots, \vec{x}_k) = [\![\mathcal{S}'']\!]((\vec{x}_1, \vec{b}_1), \cdots, (\vec{x}_k, \vec{b}_k)).$$

Hereafter, for any sequence $(\vec{b}_1, \cdots, \vec{b}_k)$ of the primary inputs $\mathcal{I}'$, we denote by $\sharp\mathtt{Clk}(\vec{b}_1, \cdots, \vec{b}_k)$ the number of clock cycles i such that at least one control input of $\vec{b}_i$ is $\mathtt{1}$, and by $\mathtt{MaxFEpClk}(\vec{b}_1, \cdots, \vec{b}_k)$ the maximum sum of the control inputs of $\vec{b}_i$ per clock cycle $i \in [k]$.

4.3 SAT Encoding

Recall that $\langle \mathcal{S}', \mathbf{B} \rangle \models \zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$ iff each fault vector $\mathsf{V}(\mathcal{S}', \mathbf{B}, T) \in [\![\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)]\!]$ is ineffective, i.e., for any sequence $(\vec{x}_1, \cdots, \vec{x}_k)$ of primary inputs, either $[\![\mathcal{S}']\!](\vec{x}_1, \cdots, \vec{x}_k) = [\![\mathcal{S}'[\mathsf{V}(\mathcal{S}', \mathbf{B}, T)]]\!](\vec{x}_1, \cdots, \vec{x}_k)$ or the fault is successfully detected by setting the error flag output $o_{\mathtt{flag}}$ in time. By Proposition 2, $\langle \mathcal{S}', \mathbf{B} \rangle \models \zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$ iff for any sequence

$((\vec{x}_1, \vec{b}_1), \cdots, (\vec{x}_k, \vec{b}_k))$ of primary inputs $\mathcal{I} \cup \mathcal{I}'$ such that $\sharp\texttt{Clk}(\vec{b}_1, \cdots, \vec{b}_k) \leq \texttt{n}_c$ and $\texttt{MaxFEpClk}(\vec{b}_1, \cdots, \vec{b}_k) \leq \texttt{n}_e$, either $[\![\mathcal{S}']\!](\vec{x}_1, \cdots, \vec{x}_k) = [\![\mathcal{S}'']\!]((\vec{x}_1, \vec{b}_1), \cdots, (\vec{x}_k, \vec{b}_k))$ or the fault is successfully detected by setting the error flag output $o_{\texttt{flag}}$ in time.

The above conditions can be reduced to the SAT problem by adapting the SAT encoding for equivalence checking [KvE02, KH03], with two additional constraints:

$$\sharp\texttt{Clk}(\vec{b}_1, \cdots, \vec{b}_k) \leq \texttt{n}_c \text{ and } \texttt{MaxFEpClk}(\vec{b}_1, \cdots, \vec{b}_k) \leq \texttt{n}_e.$$

Formally, the fault-resistance problem of the circuit $\mathcal{S}'$ can be formulated as:

$$\begin{array}{l} \forall \vec{x}_1, \cdots, \vec{x}_k \in \mathbb{B}^{|\mathcal{I}|}.\ \forall \vec{b}_1 \in \mathbb{B}^{|I''_1|}, \cdots, \forall \vec{b}_k \in \mathbb{B}^{|I''_k|}.\forall i \in [k].\ \forall o \in \mathcal{O} \setminus \{o_{\texttt{flag}}\}. \\ \Big(\sharp\texttt{Clk}(\vec{b}_1, \cdots, \vec{b}_k) \leq \texttt{n}_c \wedge \texttt{MaxFEpClk}(\vec{b}_1, \cdots, \vec{b}_k) \leq \texttt{n}_e\Big) \\ \qquad\qquad\qquad\qquad \Rightarrow \Big(\psi_{i,o} \neq \psi''_{i,o} \Rightarrow \exists j \in [i].\ \psi''_{j,o_{\texttt{flag}}}\Big) \end{array} \tag{1}$$

where $\psi_{i,o}$ is a Boolean formula that is satisfiable under an assignment $(\vec{x}_1, \cdots, \vec{x}_i)$ iff $[\![\mathcal{S}'_i]\!]_{\downarrow o}(\vec{x}_1, \cdots, \vec{x}_i) = \texttt{1}$, and $\psi''_{i,o}$ is a Boolean formula that is satisfiable under an assignment $((\vec{x}_1, \vec{b}_1), \cdots, (\vec{x}_i, \vec{b}_i))$ iff $[\![\mathcal{S}''_i]\!]_{\downarrow o}((\vec{x}_1, \vec{b}_1), \cdots, (\vec{x}_i, \vec{b}_i)) = \texttt{1}$. Note that $[\![\mathcal{S}'_i]\!]_{\downarrow o}(\vec{x}_1, \cdots, \vec{x}_i)$ (resp. $[\![\mathcal{S}''_i]\!]_{\downarrow o}((\vec{x}_1, \vec{b}_1), \cdots, (\vec{x}_i, \vec{b}_i))$) denotes the signal of the output o of the circuit $\mathcal{S}'_i$ (resp. $\mathcal{S}''_i$) at the i-th clock cycle.

Intuitively, Eqn.1 is valid iff for any sequence $((\vec{x}_1, \vec{b}_1), \cdots, (\vec{x}_k, \vec{b}_k))$ of primary inputs $\mathcal{I} \cup \mathcal{I}'$ such that $\sharp\texttt{Clk}(\vec{b}_1, \cdots, \vec{b}_k) \leq \texttt{n}_c$ and $\texttt{MaxFEpClk}(\vec{b}_1, \cdots, \vec{b}_k) \leq \texttt{n}_e$, if some primary output o (except for the error flag $o_{\texttt{flag}}$) differs at some clock cycle i, then the error flag $o_{\texttt{flag}}$ should be $\texttt{1}$ at some clock cycle j with $j \leq i$, i.e., the fault injection is detected in time.

By negating the above formula, the fault-resistance verification problem is reduced to the satisfiability of the Boolean formula (Ψ_{fr}):

$$\begin{array}{l} \Psi_{fr} := \Big(\ \Psi_{\texttt{n}_c} \wedge \Psi_{\texttt{n}_e} \wedge \bigvee_{i \in [k]} \bigvee_{o \in \mathcal{O} \setminus \{o_{\texttt{flag}}\}} \big(\psi_{i,o} \neq \psi''_{i,o} \wedge \bigwedge_{j \in [i]} \neg \psi''_{i,o_{\texttt{flag}}}\big),\ \Big) \quad \text{where} \\ \Psi_{\texttt{n}_c} := \big(\bigwedge_{i \in [k]} (d_i \Leftrightarrow \bigvee \vec{b}_{i,ctrl})\big) \wedge \sum_{i \in [k]} d_i \leq \texttt{n}_c, \qquad \Psi_{\texttt{n}_e} := \bigwedge_{i \in [k]} (\sum \vec{b}_{i,ctrl} \leq \texttt{n}_e) \end{array}$$

and for each $i \in [k]$, $\vec{b}_{i,ctrl}$ denotes the set of control inputs in the primary inputs $\vec{b}_i$. Intuitively, $\Psi_{\texttt{n}_c}$ encodes the constraint $\sharp\texttt{Clk}(\vec{b}_1, \cdots, \vec{b}_k) \leq \texttt{n}_c$, where for each $i \in [k]$, d_i is a fresh Boolean variable such that d_i is $\texttt{1}$ iff some control input in $\vec{b}_{i,ctrl}$ is $\texttt{1}$. Thus, $\sum_{i \in [k]} d_i$ is the total number of clock cycles during which at least one fault is injected on some gate. $\Psi_{\texttt{n}_e}$ encodes the constraint $\texttt{MaxFEpClk}(\vec{b}_1, \cdots, \vec{b}_k) \leq \texttt{n}_e$, where for each $i \in [k]$, $\sum \vec{b}_{i,ctrl}$ is the total number of faults injected at the i-th clock cycle.

Though cardinality constraints of the form $\sum_{i \in [n]} b_i \leq k$ are used in both $\Psi_{\texttt{n}_c}$ and $\Psi_{\texttt{n}_e}$, they can be efficiently translated into Boolean formulas in polynomial time, and the size of the resulting Boolean formula is also polynomial in the size of the cardinality constraint [ES06, Wyn18]. In our implementation, we use the sorting network implemented in Z3 [dMB08] for translating cardinality constraints into Boolean formulas.

Proposition 3. $\langle \mathcal{S}', \mathbf{B} \rangle \models \zeta(\texttt{n}_e, \texttt{n}_c, T, \ell)$ *iff the formula* Ψ_{fr} *is unsatisfiable, where the size of* Ψ_{fr} *is polynomial in the size of the circuit* $\mathcal{S}'$.

Example 1. Consider the fault-resistance model $\zeta(1, 1, \mathcal{T}, \texttt{1})$. Suppose S is the circuit in Fig. 2 (grey-area), S' is the entire circuit in Fig. 2, and the blacklist $\mathbf{B}$ contains all the logic gates in the redundancy and parity checking parts. The Boolean formula Ψ_{fr} of the example is

$$\Psi_{fr} := \Psi_{\texttt{n}_c} \wedge \Psi_{\texttt{n}_e} \wedge \big(\bigvee_{o \in \{\texttt{w},\texttt{x},\texttt{y},\texttt{z}\}} \psi_{1,o} \neq \psi''_{1,o}\big) \wedge \neg \psi''_{1,\texttt{flag}} \bigwedge_{i=1}^{12} \phi_i$$

where

$$\begin{array}{ll}
\Psi_{\mathtt{n}_c} := \big(d_1 \Leftrightarrow \bigvee_{i=1}^{12} c_i\big) \wedge d_1 \leq 1 & \Psi_{\mathtt{n}_e} := (\sum_{i=1}^{12} c_i \leq 1) \\
\psi_{1,\mathtt{x}} := ((b \oplus c) \vee (b \oplus a \oplus (\neg c \wedge d))) \oplus ((\neg c \vee a) \oplus d) & \psi_{1,\mathtt{y}} := b \oplus (\neg c \vee a) \oplus d \\
\psi_{1,\mathtt{w}} := (b \oplus c) \oplus ((b \oplus a) \wedge ((\neg c \vee a) \oplus d)) & \psi_{1,\mathtt{z}} := b \oplus a \oplus (\neg c \wedge d) \\
\phi_1 := g_1 \Leftrightarrow G''_{\oplus,\mathcal{T}}(b, c, c_1, b_{1,1}, b_{1,2}) & \psi''_{1,\mathtt{w}} := g_{10} \\
\phi_2 := g_2 \Leftrightarrow G''_{\neg,\mathcal{T}}(c, c_2, b_{2,1}, b_{2,2}) & \psi''_{1,\mathtt{y}} := g_{12} \\
\phi_3 := g_3 \Leftrightarrow G''_{\oplus,\mathcal{T}}(b, a, c_3, b_{3,1}, b_{3,2}) & \psi''_{1,\mathtt{x}} := g_{11} \\
\phi_4 := g_4 \Leftrightarrow G''_{\wedge,\mathcal{T}}(g_2, d, c_4, b_{4,1}, b_{4,2}) & \psi''_{1,\mathtt{z}} := g_9 \\
\phi_5 := g_5 \Leftrightarrow G''_{\vee,\mathcal{T}}(g_2, a, c_5, b_{5,1}, b_{5,2}) & \phi_6 := g_6 \Leftrightarrow G''_{\oplus,\mathcal{T}}(g_5, d, c_6, b_{6,1}, b_{6,2}) \\
\phi_{10} := g_{10} \Leftrightarrow G''_{\oplus,\mathcal{T}}(g_1, g_7, c_{10}, b_{10,1}, b_{10,2}) & \phi_7 := g_7 \Leftrightarrow G''_{\wedge,\mathcal{T}}(g_3, g_6, c_7, b_{7,1}, b_{7,2}) \\
\phi_{11} := g_{11} \Leftrightarrow G''_{\oplus,\mathcal{T}}(g_6, g_8, c_{11}, b_{11,1}, b_{11,2}) & \phi_8 := g_8 \Leftrightarrow G''_{\vee,\mathcal{T}}(g_1, g_9, c_8, b_{8,1}, b_{8,2}) \\
\phi_{12} := g_{12} \Leftrightarrow G''_{\oplus,\mathcal{T}}(g_6, b, c_{12}, b_{12,1}, b_{12,2}) & \phi_9 := g_9 \Leftrightarrow G''_{\oplus,\mathcal{T}}(g_3, g_4, c_9, b_{9,1}, b_{9,2})
\end{array}$$

$$\psi''_{1,\mathtt{flag}} := g_9 \oplus g_{10} \oplus g_{11} \oplus g_{12} \oplus \Big(\big(a \wedge (c \overline{\oplus} d)\big) \vee \big(((a \vee c) \overline{\wedge} d) \wedge b\big)\Big).$$

Note that g_i for each $i \in [12]$ is a fresh Boolean variable as a shortcut of a common gadget via ϕ_i, $b_{i,1}$ and $b_{i,2}$ (resp. c_i) for each $i \in [12]$ are fresh Boolean variables denoting the selection inputs (resp. control input) of the corresponding gadget, $\Psi_{\mathtt{n}_c}$ can be removed from Ψ_{fr} since it always holds, and $\Psi_{\mathtt{n}_e}$ can be efficiently translated into an equivalent Boolean formula.

We can show that Ψ_{fr} is satisfiable, thus S' is not fault-resistant w.r.t. the blacklist $\mathbf{B}$ and $\zeta(1, 1, \mathcal{T}, \mathtt{1})$. Note that in practice, $\mathbf{B}$ only contains all the logic gates in the parity checking. For simplicity, $\mathbf{B}$ also contains all the logic gates in the redundancy part in this example. □

4.4 Vulnerable Gate Reduction

Consider a fault event $\mathsf{e}(\alpha, \beta, \tau)$ to the circuit $\mathcal{S}'$. For any fixed sequence of primary inputs $(\vec{x}_1, \cdots, \vec{x}_k)$, if the output signal of the gate β does not change, $\mathsf{e}(\alpha, \beta, \tau)$ will not affect the primary outputs and thus can be omitted; otherwise, the effect of $\mathsf{e}(\alpha, \beta, \tau)$ must be propagated to the successor gates.

Assume the output of the gate β is *only* connected to one vulnerable logic gate β'. If the output signal of β' does not change, the effect of the fault event $\mathsf{e}(\alpha, \beta, \tau)$ terminates at β', thus $\mathsf{e}(\alpha, \beta, \tau)$ can be omitted. If it changes, it is flipped either from `1` to `0` or from `0` to `1`, the same effect can be achieved by applying the fault event $\mathsf{e}(\alpha, \beta', \tau_{bf})$, or the fault event $\mathsf{e}(\alpha, \beta', \tau_s)$ if it is flipped from `0` to `1` or the fault event $\mathsf{e}(\alpha, \beta', \tau_r)$ if it is flipped from `1` to `0`. As a result, it suffices to consider fault injections on the gate β' instead of both β and β' when $\tau_{bf} \in T$ or $\{\tau_s, \tau_r\} \subseteq T$, which reduces the number of vulnerable gates.

Theorem 2. *Consider a fault-resistance model $\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$ such that $\tau_{bf} \in T$ or $\{\tau_s, \tau_r\} \subseteq T$, and $\ell \in \{\mathtt{l}, \mathtt{lm}\}$. Let*

$$\mathsf{V}_1(\mathcal{S}', \mathbf{B}, T) = \mathsf{V}(\mathcal{S}', \mathbf{B}, T) \cup \{\mathsf{e}(\alpha, \beta, \tau)\} \in [\![\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)]\!]$$

be an effective fault vector on the circuit $\mathcal{S}'$. If the output of the gate β is only connected to one logic gate β' and $\beta' \notin \mathbf{B}$, then there exists a fault vector $\mathsf{V}'(\mathcal{S}', \mathbf{B}, T) \subseteq \mathsf{V}(\mathcal{S}', \mathbf{B}, T) \cup \{\mathsf{e}(\alpha, \beta', \tau')\}$ for some $\tau' \in T$ such that $\mathsf{V}'(\mathcal{S}', \mathbf{B}, T)$ is also effective on the circuit $\mathcal{S}'$.

Moreover, if $\mathsf{V}(\mathcal{S}', \mathbf{B}, T) = \emptyset$, then $\{\mathsf{e}(\alpha, \beta', \tau')\}$ for some $\tau' \in T$ is effective on the circuit $\mathcal{S}'$.

Proof. Suppose $\mathsf{V}_1(\mathcal{S}', \mathbf{B}, T)$ is an effective fault vector on $\mathcal{S}'$. There exists a sequence of primary inputs $(\vec{x}_1, \cdots, \vec{x}_k)$ such that the sequence of primary outputs $[\![\mathcal{S}']\!](\vec{x}_1, \cdots, \vec{x}_k)$ and $[\![\mathcal{S}'[\mathsf{V}_1(\mathcal{S}', \mathbf{B}, T)]]\!](\vec{x}_1, \cdots, \vec{x}_k)$ differ at some clock cycle before the error flag output $o_{\mathtt{flag}}$ differs. We proceed by distinguishing whether the output signal of the gate β' differs in the circuits $\mathcal{S}'$ and $\mathcal{S}'[\mathsf{V}_1(\mathcal{S}', \mathbf{B}, T)]$ under the same sequence of primary inputs $(\vec{x}_1, \cdots, \vec{x}_k)$.

- If the output signal of the gate β' is the same in the circuits $\mathcal{S}'$ and $\mathcal{S}'[\mathsf{V}_1(\mathcal{S}', \mathbf{B}, T)]$ under the same sequence of primary inputs $(\vec{x}_1, \cdots, \vec{x}_k)$, then the effect of the fault event $\mathsf{e}(\alpha, \beta, \tau)$ is stopped at the gate β', as the output of the gate β is only connected to the gate β'. Thus, the sequences of primary outputs $[\![\mathcal{S}'[\mathsf{V}(\mathcal{S}', \mathbf{B}, T)]]\!](\vec{x}_1, \cdots, \vec{x}_k)$ and $[\![\mathcal{S}'[\mathsf{V}_1(\mathcal{S}', \mathbf{B}, T)]]\!](\vec{x}_1, \cdots, \vec{x}_k)$ are the same, implying that the sequences of primary outputs $[\![\mathcal{S}'[\mathsf{V}(\mathcal{S}', \mathbf{B}, T)]]\!](\vec{x}_1, \cdots, \vec{x}_k)$ and $[\![\mathcal{S}']\!](\vec{x}_1, \cdots, \vec{x}_k)$ differ at some clock cycle before the error flag output $o_{\mathtt{flag}}$ differs. The result follows immediately.

- If the output signal of the gate β' differs in the circuits $\mathcal{S}'$ and $\mathcal{S}'[\mathsf{V}_1(\mathcal{S}', \mathbf{B}, T)]$ under the same sequence of primary inputs $(\vec{x}_1, \cdots, \vec{x}_k)$, then the fault propagation from the fault event $\mathsf{e}(\alpha, \beta, \tau)$ flips the output signal of the gate β'. Let $\mathsf{V}_2(\mathcal{S}', \mathbf{B}, T) = \mathsf{V}(\mathcal{S}', \mathbf{B}, T) \cup \{\mathsf{e}(\alpha, \beta', \tau')\}$, where $\tau' = \tau_{bf}$ if $\tau_{bf} \in T$, otherwise $\tau = \tau_s$ if the output signal of the gate β' flips from $\mathtt{0}$ to $\mathtt{1}$ due to the fault event $\mathsf{e}(\alpha, \beta, \tau)$ and $\tau = \tau_r$ if the output signal of the gate β' flips from $\mathtt{1}$ to $\mathtt{0}$ due to the fault event $\mathsf{e}(\alpha, \beta, \tau)$. Then, the sequences of primary outputs $[\![\mathcal{S}'[\mathsf{V}_1(\mathcal{S}', \mathbf{B}, T)]]\!](\vec{x}_1, \cdots, \vec{x}_k)$ and $[\![\mathcal{S}'[\mathsf{V}_2(\mathcal{S}', \mathbf{B}, T)]]\!](\vec{x}_1, \cdots, \vec{x}_k)$ are the same, as the output of the gate β is only connected to the gate β'. Thus, $[\![\mathcal{S}']\!](\vec{x}_1, \cdots, \vec{x}_k)$ and $[\![\mathcal{S}'[\mathsf{V}_2(\mathcal{S}', \mathbf{B}, T)]]\!](\vec{x}_1, \cdots, \vec{x}_k)$ differ at some clock cycle before the error flag output $o_{\mathtt{flag}}$ differs. The result follows immediately.

Moreover, if $\mathsf{V}(\mathcal{S}', \mathbf{B}, T) = \emptyset$, the output signal of the gate β' must differ in the circuits $\mathcal{S}'$ and $\mathcal{S}'[\mathsf{V}_1(\mathcal{S}', \mathbf{B}, T)]$ under the same sequence of primary inputs $(\vec{x}_1, \cdots, \vec{x}_k)$, otherwise $\mathsf{V}_1(\mathcal{S}', \mathbf{B}, T)$ is ineffective on the circuit $\mathcal{S}'$. The result follows from the fact that $\mathsf{V}_2(\mathcal{S}', \mathbf{B}, T) = \{\mathsf{e}(\alpha, \beta', \tau')\}$. □

Let $\mathbf{B}'$ be the set of gates β such that the output of β is only connected to one logic gate $\beta' \notin \mathbf{B}$, which can be computed by a graph traversal of the circuit $\mathcal{S}'$. By Theorem 2, $\mathbf{B}'$ can be safely merged with the blacklist $\mathbf{B}$ while no protections are required for those gates. Moreover, $\langle \mathcal{S}', \mathbf{B} \rangle \models \zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$ entails $\langle \mathcal{S}', \mathbf{B} \cup \mathbf{B}' \rangle \models \zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$. We get that:

Corollary 1. *Given a fault-resistance model $\zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$ such that $\tau_{bf} \in T$ or $\{\tau_s, \tau_r\} \subseteq T$, and $\ell \in \{\mathtt{l}, \mathtt{lm}\}$, $\langle \mathcal{S}', \mathbf{B} \cup \mathbf{B}' \rangle \models \zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$ iff $\langle \mathcal{S}', \mathbf{B} \rangle \models \zeta(\mathtt{n}_e, \mathtt{n}_c, T, \ell)$.*

We remark that the vulnerable gate reduction does not consider conditional fault propagation because the inputs of the gates are not fixed to constants 1/0 when considering all the valid inputs to the circuit. It could be adapted if some inputs of AND/OR gates are fixed.

Example 2. Consider the fault-resistance model $\zeta(1, 1, \mathcal{T}, \mathtt{l})$, the circuit S in Fig. 2 (grey-area), and the entire circuit S' in Fig. 2. All the gates in the redundancy part except for the gate `p6` (i.e., the gate whose output is `p6`) can be added into $\mathbf{B}'$, as the effect of an effective fault injection on any of those gates can be achieved by at most one bit-flip fault injection on `p6`. Note that `p6` itself cannot be added into $\mathbf{B}'$ because the gate `flag` is in $\mathbf{B}$. Similarly, the gates `s4,s5,s7,s8` in the grey-area can be added into $\mathbf{B}'$, but the other gates cannot as their outputs are connected to more than one gate or some outputs of $\{\mathtt{w}, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$. Now, the fault-resistance verification problem w.r.t. the fault-resistance model $\zeta(1, 1, \mathcal{T}, \mathtt{l})$ and the blacklist $\mathbf{B}$ is reduced to SAT solving of the Boolean formula (Ψ''_{fr})

$$\Psi''_{fr} := \Psi'_{\mathtt{n}_c} \wedge \Psi'_{\mathtt{n}_e} \wedge \Big(\bigvee_{o \in \{\mathtt{w},\mathtt{x},\mathtt{y},\mathtt{z}\}} \psi_{1,o} \neq \psi''_{1,o} \Big) \wedge \neg \psi''_{1,\mathtt{flag}} \wedge \bigwedge_{i=1}^{12} \phi'_i,$$

where $\phi'_i := \phi_i$ for $i \in Z = \{1, 2, 3, 6, 9, 10, 11, 12\}$, $\phi'_4 := g_4 \Leftrightarrow g_2 \wedge d$, $\phi'_5 := g_5 \Leftrightarrow g_2 \vee a$, $\phi'_7 := g_7 \Leftrightarrow g_3 \wedge g_6$, $\phi'_8 := g_8 \Leftrightarrow g_1 \wedge g_9$, $\Psi'_{\mathtt{n}_c} := (d_1 \Leftrightarrow \bigvee_{i \in Z} c_i) \wedge d_1 \leq 1$ and $\Psi'_{\mathtt{n}_e} := (\sum_{i \in Z} c_i \leq 1)$. □

Table 2: Benchmark statistics, where Ri denotes i rounds, bi denotes countermeasure for i-bit faults, and D (resp. C) denotes detection-(resp. correction-)based countermeasure.

Name	#Clk	|B|	#in	#out	#gate	#and	#nand	#or	#nor	#xor	#xnor	#not	#reg
AES-R1	1	0	256	128	21201	464	7936	592	8480	464	560	2705	0
AES-R1-b1-D	1	432	256	129	24864	576	9446	560	9705	828	852	2897	0
AES-R1-b2-D	1	1055	256	129	34159	704	12698	833	13012	1440	1584	3888	0
AES-R2	2	0	384	128	42112	928	15872	1184	16960	992	1056	4864	256
AES-R2-b1-D	2	865	384	129	50017	1152	18892	1121	19410	1656	1704	5794	288
AES-R2-b2-D	2	2111	384	129	68703	1408	25396	1667	26024	2880	3168	7776	384
CRAFT-R1	2	0	128	64	480	0	160	16	80	80	32	48	64
CRAFT-R1-b1-C	2	192	128	64	3140	0	864	48	656	460	728	272	112
CRAFT-R1-b2-C	2	1312	128	64	20948	336	7856	384	6576	1484	2056	2080	176
CRAFT-R1-b1-D	2	159	128	65	925	41	155	65	148	187	193	56	80
CRAFT-R1-b2-D	2	383	128	65	1522	49	266	49	211	201	539	95	112
CRAFT-R1-b3-D	2	511	128	65	1807	48	282	97	292	240	640	80	128
CRAFT-R2	3	0	192	64	960	0	320	32	160	160	64	96	128
CRAFT-R2-b1-C	3	192	192	64	5848	0	1680	96	1248	808	1264	528	224
CRAFT-R2-b2-C	3	1312	192	64	40184	672	15152	752	12576	2680	3936	4064	352
CRAFT-R2-b1-D	3	400	192	65	1880	100	292	120	311	382	394	121	160
CRAFT-R2-b2-D	3	959	192	65	3139	97	553	100	449	434	1094	188	224
CRAFT-R3	4	0	256	64	1440	0	480	48	240	240	96	144	192
CRAFT-R3-b3-D	4	1791	256	65	5567	192	954	193	836	672	2048	288	384
CRAFT-R4	5	0	320	64	1920	0	640	64	320	320	128	192	256
CRAFT-R4-b3-D	5	2303	320	65	7295	256	1242	257	1124	944	2576	384	512
LED-R1	1	0	128	64	976	16	272	16	32	288	304	48	0
LED-R1-b1-D	1	240	128	65	1552	16	346	32	53	416	608	81	0
LED-R1-b2-D	1	575	128	65	2463	17	479	64	111	512	1168	112	0
LED-R2	2	0	128	64	1952	32	544	32	64	512	608	96	64
LED-R2-b1-D	2	480	128	65	2976	32	690	64	109	760	1112	129	80
LED-R2-b2-D	2	1151	128	65	4687	33	951	128	231	984	2072	176	112
LED-R3	3	0	128	64	2928	48	816	48	96	736	912	144	128
LED-R3-b1-D	3	720	128	65	4400	48	1030	96	169	1116	1604	177	160
LED-R3-b2-D	3	1727	128	65	6911	49	1411	192	363	1492	2940	240	224
GIFT-R1	1	0	192	64	416	16	128	0	64	0	80	64	64
GIFT-R1-b1-D	1	240	192	65	1152	16	266	16	213	96	336	129	80
GIFT-R1-b2-D	1	575	192	65	2047	33	431	32	415	80	768	176	112
GIFT-R2	2	0	192	64	832	32	256	0	128	0	160	128	128
GIFT-R2-b1-D	2	544	192	65	2320	32	553	33	469	168	616	289	160
GIFT-R2-b2-D	2	1343	192	65	4191	65	924	67	959	136	1368	448	224
PRESENT-R1	1	0	192	64	544	16	272	16	32	48	48	48	64
PRESENT-R1-b1-D	1	240	192	65	1464	20	550	40	213	156	264	141	80
PRESENT-R1-b2-D	1	575	192	65	2567	41	863	56	431	164	668	232	112
PRESENT-R2	2	0	192	64	1088	32	544	32	64	96	96	96	128
PRESENT-R2-b1-D	2	540	192	64	2940	40	1143	95	429	289	473	311	160
PRESENT-R2-b2-D	2	1331	192	64	5219	81	1854	157	871	307	1171	554	224
SIMON-R1	1	0	192	64	160	0	32	0	0	32	32	0	64
SIMON-R1-b1-D	1	288	192	65	608	0	124	12	23	130	190	49	80
SIMON-R1-b2-D	1	719	192	65	1239	1	269	36	85	110	514	112	112
SIMON-R2	2	0	192	64	320	0	64	0	0	64	64	0	128
SIMON-R2-b1-D	2	516	192	65	1108	1	195	0	63	252	370	67	160
SIMON-R2-b2-D	2	1259	192	65	2203	4	378	0	221	244	998	221	224

5 Evaluation

We have implemented our approaches as an open-source tool FIRMER. Given circuits $\mathcal{S}$ and $\mathcal{S}'$ in Verilog gate-level netlists, and a configuration file describing the blacklist and fault-resistance model, FIRMER verifies the fault-resistance of $\mathcal{S}'$. FIRMER first expresses the constraints in quantifier-free bit-vector theory (QF_BV) using our SAT encoding and then translates to Boolean formulas (in the DIMACS format) via Z3 4.11.2.0 with `card2bv` and `tseitin-cnf` options [dMB08]. Those Boolean formulas can be solved by off-the-shelf SAT solvers. Currently, FIRMER uses the parallel SAT solver Glucose 4.2.1 [AS18].

Benchmarks. We use the VHDL implementations of six cryptographic algorithms (i.e., CRAFT, LED, AES, GIFT, PRESENT and SIMON), taken from [AMR$^+$20, SRM20]. Among all the available benchmarks of [AMR$^+$20, SRM20], CRAFT, LED, and AES are the same as FIVER; GIFT is the newest algorithm; PRESENT and SIMON are the most widely cited algorithms in the literature. The VHDL implementations are unrolled and transformed into Verilog gate-level netlists using the Synopsys design compiler (version O-2018.06-SP2). The CRAFT benchmarks adopt both detection-based (D) and correction-based (C) countermeasures while the others adopt a detection-based countermeasure. To thoroughly evaluate the scalability of FIRMER, these benchmarks vary with the number of rounds (Ri) and the maximal number of protected faulty bits (bi) (i.e., the circuit $\mathcal{S}'$ is claimed to be fault-resistant), in particular, CRAFT-R3-b3-D and CRAFT-R4-b3-D are added as large benchmarks with detection countermeasures for evaluating the scalability of FIRMER in terms of rounds and maximal number of protected faulty bits. As a result, the number of gates ranges from 608 to 68,703. The blacklists are the same as the ones used in FIVER. The detailed statistics of the benchmark are given in Table 2, where the first three columns respectively give the benchmark name, number of clock cycles, size of the blacklist $\mathbf{B}$, and the other columns give the numbers of primary inputs, primary outputs, gates and each specific gate. Circuits without any protections are given as reference.

Setup. We compare FIRMER with (1) the state-of-the-art verifier FIVER [RBSS$^+$21] that utilizes BDD and (2) an SMT-based approach which directly checks the constraints generated by our encoding method without translating to Boolean formulas for which we use the SMT solver Bitwuzla 1.0-prerelease [NP23], the winner of QF_BV (Single Query Track) Division at SMT-COMP 2021 and 2022. We only report the results for fault-resistance models with all the fault types $\mathcal{T} = \{\tau_s, \tau_r, \tau_{bf}\}$, which are more important than the others with fewer fault types according to Proposition 1. The results w.r.t. fault-resistance models limited to the fault type τ_{bf} are given in Appendix B from which a similar conclusion can be drawn.

The experiments were conducted on a machine with Intel Xeon Gold 6342 2.80GHz CPU, 1TB RAM, and Ubuntu 20.04.1. (Note that 1TB RAM is the memory of the machine used for evaluation instead of memory consumption and the maximum amount of consumed memory in our experiments is less than 1GB for single-thread setting.) Each verification task was run with 24 hours timeout (real time). The SAT solver Glucose 4.2.1 is run with 8 threads and default parameters unless stated otherwise; the SMT solver Bitwuzla 1.0-prerelease is run with a single thread and default parameters as there are no promising parallel SMT solvers for (quantifier-free) bit-vector theory; the BDD-based verifier FIVER is run with 8 threads, 8 GB RAM for its internal computation on BDD, and an early-abort strategy (i.e., setting `interrupt` to `true` so that the analysis is stopped when an effective fault vector is found). A detailed technical comparison between FIVER and FIRMER is given in Section 6.

Results. The results are reported in Table 3. Columns "#Var" and "#Cls" respectively give the numbers of Boolean variables and Clauses of the resulting Boolean formula. Columns "2CNF" and "Solving" give the execution time of building and solving Boolean formulas in seconds, respectively. Columns "Total" and "Time" show the total execution time in seconds. Mark ✓ (resp. ✗) indicates that the protected circuit $\mathcal{S}'$ is fault-resistant (resp. not fault-resistant) w.r.t. the given fault-resistance model in column "Model".

Overall, FIRMER (i.e., SAT-based approach) solved all the verification tasks, the SMT-based approach ran out of time on one verification task (24 hours per task), while FIVER ran out of time on 31 verification tasks. The SAT/SMT-based approach becomes

Table 3: Verification results, where R denotes Result, and DR denotes Desired Result.

Name	Model	FIRMER (SAT)					SMT	FIVER	R	DR
		#Var	#Cls	2CNF	Solving	Total	Time	Time		
AES-R1-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	57340	310569	1.97	193.27	195.24	4856.58	**59.49**	✓	✓
AES-R1-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	69088	246108	1.96	0.61	**2.57**	3650.83	34744.40	✗	✗
AES-R1-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	86333	329543	4.84	1845.23	**1850.07**	23542.57	timeout	✓	✓
AES-R1-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	90428	354018	4.38	0.91	**5.28**	3386.38	timeout	✗	✗
AES-R2-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	120044	658527	33.76	1706.67	**1740.43**	48359.73	timeout	✓	✓
AES-R2-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	138266	579518	33.82	2.13	**35.95**	32302.85	timeout	✗	✗
AES-R2-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	185415	740484	79.86	14243.80	**14323.66**	timeout	timeout	✓	✓
AES-R2-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	194500	807305	80.19	3.50	**83.69**	95.05	timeout	✗	✗
CRAFT-R1-b1-C	$\zeta(1,1,\mathcal{T},\texttt{1m})$	6767	33938	0.13	0.66	0.79	9.81	**0.14**	✓	✓
CRAFT-R1-b1-C	$\zeta(2,1,\mathcal{T},\texttt{1m})$	7608	32039	0.14	0.06	**0.20**	0.28	1.34	✗	✗
CRAFT-R1-b2-C	$\zeta(2,1,\mathcal{T},\texttt{1m})$	52901	234385	2.79	130.00	**132.79**	2898.55	338.23	✓	✓
CRAFT-R1-b2-C	$\zeta(3,1,\mathcal{T},\texttt{1m})$	55099	258959	2.80	0.59	**3.39**	21.49	4491.08	✗	✗
CRAFT-R1-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	2255	10735	0.03	0.10	0.13	0.33	**0.04**	✓	✓
CRAFT-R1-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	2567	10316	0.03	0.03	0.06	**0.01**	0.35	✗	✗
CRAFT-R1-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	3356	13780	0.02	0.22	**0.24**	1.02	1.13	✓	✓
CRAFT-R1-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	3538	15058	0.03	0.03	**0.06**	0.20	117.54	✗	✗
CRAFT-R1-b3-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	3974	19042	0.03	0.43	**0.46**	4.43	8007.42	✓	✓
CRAFT-R1-b3-D	$\zeta(4,1,\mathcal{T},\texttt{1m})$	4132	20428	0.04	0.03	**0.07**	0.34	82955.17	✗	✗
CRAFT-R2-b1-C	$\zeta(1,1,\mathcal{T},\texttt{1m})$	12644	63451	0.36	11.27	11.63	101.19	**4.20**	✓	✓
CRAFT-R2-b1-C	$\zeta(2,1,\mathcal{T},\texttt{1m})$	14215	59873	0.31	0.14	**0.45**	0.60	394.79	✗	✗
CRAFT-R2-b2-C	$\zeta(2,1,\mathcal{T},\texttt{1m})$	104139	452464	6.68	2054.13	**2060.81**	13585.63	5012.34	✓	✓
CRAFT-R2-b2-C	$\zeta(3,1,\mathcal{T},\texttt{1m})$	108384	498341	7.74	2.67	**10.41**	3000.80	29120.79	✗	✗
CRAFT-R2-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	4195	20279	0.06	0.88	**0.94**	6.16	1.04	✓	✓
CRAFT-R2-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	4795	19338	0.07	0.03	**0.10**	0.18	23.63	✗	✗
CRAFT-R2-b1-D	$\zeta(1,2,\mathcal{T},\texttt{1m})$	4195	20279	0.06	1.76	**1.82**	8.25	1291.12	✓	✓
CRAFT-R2-b1-D	$\zeta(1,3,\mathcal{T},\texttt{1m})$	4194	20277	0.06	1.35	**1.41**	7.19	timeout	✓	✓
CRAFT-R2-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	5959	25370	0.09	3.25	**3.34**	34.93	88.35	✓	✓
CRAFT-R2-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	6268	27935	0.11	0.06	**0.17**	1.38	9141.86	✗	✗
CRAFT-R2-b2-D	$\zeta(2,2,\mathcal{T},\texttt{1m})$	5959	25370	0.11	3.47	**3.58**	43.36	timeout	✓	✓
CRAFT-R3-b3-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	10364	48541	0.20	37.66	**37.86**	414.00	timeout	✓	✓
CRAFT-R3-b3-D	$\zeta(4,1,\mathcal{T},\texttt{1m})$	10760	51937	0.21	0.10	**0.31**	4.69	timeout	✗	✗
CRAFT-R3-b3-D	$\zeta(3,2,\mathcal{T},\texttt{1m})$	10366	48560	0.20	57.54	**57.74**	393.80	timeout	✓	✓
CRAFT-R3-b3-D	$\zeta(3,3,\mathcal{T},\texttt{1m})$	10364	48541	0.21	36.83	**37.04**	357.59	timeout	✓	✓
CRAFT-R3-b3-D	$\zeta(3,4,\mathcal{T},\texttt{1m})$	10363	48537	0.20	31.64	**31.84**	385.19	timeout	✓	✓
CRAFT-R4-b3-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	13516	63332	0.61	61.46	**62.07**	1188.03	timeout	✓	✓
CRAFT-R4-b3-D	$\zeta(4,1,\mathcal{T},\texttt{1m})$	14039	67789	0.63	0.14	**0.77**	8.25	timeout	✗	✗
CRAFT-R4-b3-D	$\zeta(3,2,\mathcal{T},\texttt{1m})$	13518	63363	0.61	155.14	**155.75**	978.78	timeout	✓	✓
CRAFT-R4-b3-D	$\zeta(3,3,\mathcal{T},\texttt{1m})$	13518	63363	0.61	81.47	**82.08**	900.28	timeout	✓	✓
CRAFT-R4-b3-D	$\zeta(3,4,\mathcal{T},\texttt{1m})$	13516	63332	0.62	70.53	**71.15**	916.13	timeout	✓	✓
CRAFT-R4-b3-D	$\zeta(3,5,\mathcal{T},\texttt{1m})$	13515	63325	0.61	54.55	**55.16**	922.49	timeout	✓	✓
LED-R1-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	2673	13295	0.05	1.07	1.12	11.63	**0.23**	✓	✓
LED-R1-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	3038	12073	0.05	0.02	**0.07**	0.85	8.39	✗	✗
LED-R1-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	3726	16057	0.08	2.01	**2.09**	33.40	10.18	✓	✓
LED-R1-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	3853	16852	0.09	0.02	**0.11**	0.83	2292.60	✗	✗
LED-R2-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	5215	27405	0.12	9.64	**9.76**	139.78	timeout	✓	✓
LED-R2-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	5783	26245	0.12	0.05	**0.17**	1.68	timeout	✗	✗
LED-R2-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	7300	34132	0.18	13.67	**13.85**	336.45	timeout	✓	✓
LED-R2-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	7554	36930	0.18	0.06	**0.24**	2.78	timeout	✗	✗
LED-R3-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	7674	40655	0.14	36.57	**36.71**	588.07	timeout	✓	✓
LED-R3-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	8528	38988	0.14	0.11	**0.25**	2.90	timeout	✗	✗
LED-R3-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	10924	49866	0.24	47.75	**47.99**	860.20	timeout	✓	✓
LED-R3-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	11305	53487	0.25	0.12	**0.37**	3.72	timeout	✗	✗
GIFT-R1-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	1871	9060	0.07	0.09	0.16	1.85	**0.02**	✓	✓
GIFT-R1-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	2157	8554	0.08	0.03	**0.12**	1.37	0.14	✗	✗
GIFT-R1-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	2975	13226	0.13	0.20	**0.33**	11.55	0.81	✓	✓
GIFT-R1-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	3101	14616	0.13	0.04	**0.17**	9.96	42.14	✗	✗
GIFT-R2-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	3584	17658	0.15	0.27	0.42	7.00	**0.17**	✓	✓
GIFT-R2-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	4169	16645	0.15	0.05	**0.20**	5.18	3.33	✗	✗
GIFT-R2-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	5490	24697	0.30	0.66	**0.96**	22.97	15.59	✓	✓
GIFT-R2-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	5743	27294	0.30	0.08	0.38	**0.35**	1278.12	✗	✗
PRESENT-R1-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	2324	11145	0.09	0.10	0.19	2.50	**0.03**	✓	✓
PRESENT-R1-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	2619	10768	0.08	0.03	**0.11**	1.69	0.16	✗	✗
PRESENT-R1-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	3875	17048	0.14	0.24	**0.38**	8.63	0.95	✓	✓
PRESENT-R1-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	4065	19014	0.14	0.05	**0.19**	8.47	42.61	✗	✗

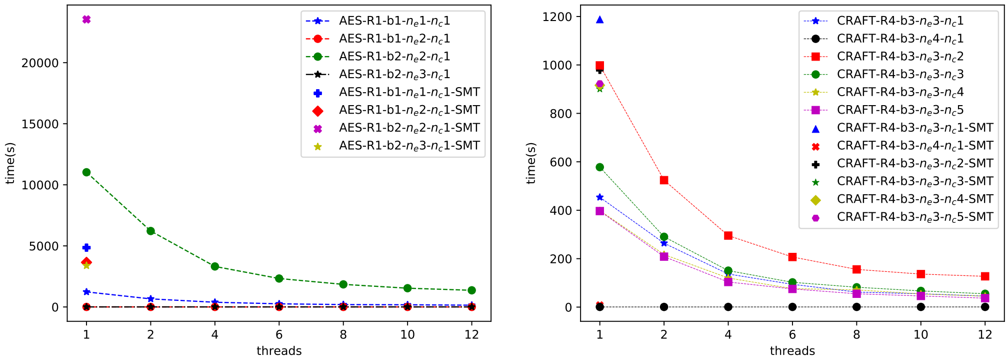
Continued on next page

Table 3 – continued from previous page

Name	Model	FIRMER (SAT)					SMT	FIVER	R	DR
		#Var	#Cls	2CNF	Solving	Total	Time	Time		
PRESENT-R2-b1-D	$\zeta(1,1,\mathcal{T},\mathtt{lm})$	4519	22601	0.16	0.42	0.58	6.88	**0.22**	✓	✓
PRESENT-R2-b1-D	$\zeta(2,1,\mathcal{T},\mathtt{lm})$	5097	21700	0.17	0.06	**0.23**	6.19	4.76	✗	✗
PRESENT-R2-b2-D	$\zeta(2,1,\mathcal{T},\mathtt{lm})$	7645	32847	0.30	0.67	**0.97**	19.47	16.09	✓	✓
PRESENT-R2-b2-D	$\zeta(3,1,\mathcal{T},\mathtt{lm})$	7978	35924	0.31	0.08	**0.39**	0.48	1259.90	✗	✗
SIMON-R1-b1-D	$\zeta(1,1,\mathcal{T},\mathtt{lm})$	1346	6162	0.04	0.09	0.13	0.93	**0.01**	✓	✓
SIMON-R1-b1-D	$\zeta(2,1,\mathcal{T},\mathtt{lm})$	1556	5893	0.05	0.02	**0.07**	0.84	0.17	✗	✗
SIMON-R1-b2-D	$\zeta(2,1,\mathcal{T},\mathtt{lm})$	2223	9341	0.08	0.15	**0.23**	2.66	0.78	✓	✓
SIMON-R1-b2-D	$\zeta(3,1,\mathcal{T},\mathtt{lm})$	2333	10443	0.08	0.02	**0.10**	2.02	65.93	✗	✗
SIMON-R2-b1-D	$\zeta(1,1,\mathcal{T},\mathtt{lm})$	2504	11819	0.07	0.20	**0.27**	3.64	timeout	✓	✓
SIMON-R2-b1-D	$\zeta(2,1,\mathcal{T},\mathtt{lm})$	2838	11420	0.07	0.03	**0.10**	2.59	timeout	✗	✗
SIMON-R2-b2-D	$\zeta(2,1,\mathcal{T},\mathtt{lm})$	4071	16900	0.12	0.63	**0.75**	9.82	timeout	✓	✓
SIMON-R2-b2-D	$\zeta(3,1,\mathcal{T},\mathtt{lm})$	4276	18697	0.13	0.05	**0.18**	6.56	timeout	✗	✗

increasingly efficient than FIVER with the increase of round numbers (i.e., Ri) and the maximal number of protected faulty bits (i.e., bj). FIRMER is significantly more efficient than the SMT-based approach on relatively larger benchmarks (e.g., AES-R1-b1, AES-R1-b2, AES-R2-b1, AES-R2-b2, CRAFT-R1-b2-C, CRAFT-R2-b2-C, CRAFT-R3-b3-D, CRAFT-R4-b3-D, LED-R2-b1-D, etc.) while they are comparable on smaller benchmarks.

Interestingly, we find that (i) implementations with correction-based countermeasures are more difficult to prove than those with detection-based countermeasures (e.g., CRAFT-Ri-bj-C vs. CRAFT-Ri-bj-D, for $i = 1, 2$ and $j = 1, 2$), because implementing correction-based countermeasures require more gates; (ii) FIRMER is more efficient at disproving than proving fault-resistance, because it is often more difficult to prove UNSAT instances than finding satisfying assignments for SAT instances in CDCL SAT solvers. (iii) FIRMER often scales very well with the increase of the round numbers (i.e., Ri for $i = 1, 2, 3, 4$), the maximal number of protected faulty bits (i.e., bj for $j = 1, 2, 3$), the maximum number of fault events per clock cycle (i.e., $\mathtt{n}_e$) and the maximum number of clock cycles in which fault events can occur (i.e., $\mathtt{n}_c$), but FIVER has very limited scalability because FIVER may have to build more BDD models, the size of which may dramatically increase.



(a) AES-R1-b1-D and AES-R1-b2-D

(b) CRAFT-R4-b3-D

Figure 8: Results with different number of threads

Different threads. The experiments reported above are based on 8 threads. To understand the effect of the number of threads, we evaluate FIRMER on AES-R1-b1-D, AES-R1-b2-D, and CRAFT-R4-b3-D by varying the number of threads from 1 to 12. The results are depicted in Fig. 8(a) and Fig. 8(b), respectively, where $\mathtt{n}_e i$ and $\mathtt{n}_c j$ denote the fault-resistance mode $\zeta(i, j, \mathcal{T}, \mathtt{lm})$. Detailed results are given in Appendix A. In general, on the fault-resistant benchmarks, with the increase of the number of threads, FIRMER becomes increasingly more efficient but the improvement diminishes. Besides the great

Table 4: Evaluation of FIRMER with different SAT and SMT solvers.

Benchmark		SMT Solver					SAT Solver		
		Z3	Bitwuzla	STP	Yices	Yices-CaDiCal	CaDiCal	Glucose	
								1 Thread	8 Thread
AES-R1-b1-n$_e$1-n$_c$1		timeout	4,856.58	23,109.85	16,163.72	11,290.39	893.63	1,234.31	195.24
AES-R1-b1-n$_e$2-n$_c$1		13,839.48	3,650.83	5.62	15,124.50	7,923.09	5.34	2.34	2.57
AES-R1-b2-n$_e$2-n$_c$1		timeout	23,542.57	timeout	timeout	40,131.98	12,283.65	11,028.35	1,850.07
AES-R1-b2-n$_e$3-n$_c$1		67,045.41	3,386.38	6,194.25	30,926.79	12,319.77	5.63	5.16	5.28
CRAFT-R4-b3	n$_e$3-n$_c$1	30,747.60	1,188.03	15,132.7	1,136.39	420.54	258.94	453.57	62.07
	n$_e$4-n$_c$1	64.20	8.25	15.02	27.35	2.43	0.65	0.73	0.77
	n$_e$3-n$_c$2	timeout	978.78	timeout	2,435.13	467.50	329.24	998.18	155.75
	n$_e$3-n$_c$3	timeout	900.28	83,138.92	1,714.56	360.00	289.67	577.90	82.08
	n$_e$3-n$_c$4	timeout	916.13	45,375.81	1,084.05	343.49	291.83	398.52	71.15
	n$_e$3-n$_c$5	timeout	922.49	40,622.68	1,215.18	402.85	285.91	396.56	55.16

advance in SAT solving, it is another reason that SAT solving outperforms SMT solving due to the lack of promising parallel SMT solvers for (quantifier-free) bit-vector theory. On the non-fault-resistant benchmarks, multi-threading does *not* improve performance (rather slightly worsens the performance), because these benchmarks are easy to be disproved and thread scheduling causes overhead.

Comparisons of different SAT/SMT solvers. To understand the performances of SAT and SMT solvers, we further evaluate FIRMER on AES-R1-b1-D, AES-R1-b2-D, and CRAFT-R4-b3-D using various SAT and SMT solvers. Besides Bitwuzla, we additionally consider three widely used SMT solvers, i.e., Z3, STP 2.3.3 [GD07], and Yices 2.6.4 [Dut14] with their default SAT engines, where STP is the winner of QF_BV (Single Query Track) Division at SMT-COMP 2023. We also configure Yices with the promising sequential SAT solver CaDiCal 1.9.5 [BFFH20] as the underlying SAT engine, denoted by Yices-CaDiCal. (CaDiCal is superior to all the other 15 state-of-the-art SAT solvers on Bit-vector problems in [Dut20], thus it is also utilized for Boolean satisfiability checking.) All these solvers are run with a single thread and default parameters (except that the SAT solver Glucose is run with 8 threads). The results are reported in Table 4, where the execution time of SAT Solvers includes the execution time of bit-blasting. We can observe that the SAT solvers almost always perform better than the SMT solvers in our application. In particular, Glucose with 8 threads is significantly more efficient than the others. The performance discrepancy between the SMT and SAT solvers (between Yices-CaDiCAL and CaDiCAL) suggests that the heuristic optimization techniques used in SMT solvers are ineffective and instead, bring significant overhead to our application. We recommend translating SMT constraints into Boolean formulas via bit-blasting and then solving them using parallel SAT solvers. We also find that the performance of FIRMER differs in SAT/SMT solvers. In detail, the SMT solvers Bitwuzla and Yices-CaDical with ups and downs on both sides often significantly outperform the other three SMT solvers (i.e., Z3, STP and Yices), because of the difference between underlying decision procedures and heuristic optimization techniques; the SAT solver CaDiCAL often performs better than Glucose with a single thread but always performs worse than Glucose with 8 threads, because Glucose is optimized for parallel computing while CaDiCAL is optimized for sequential computing. We remark that the parameters of these SAT/SMT solvers may be tuned to improve efficiency, which is costly and out of the scope of the current paper.

Vulnerable gate reduction. To understand the effectiveness of the vulnerable gate reduction for accelerating fault-resistance verification, we evaluate FIRMER with/without

Table 5: Results of FIRMER (SAT) *with/without* vulnerable gate reduction.

Name	Model	Without reduction				With reduction				R	DR
		#Var	#Cls	#Gate	Time	#Var	#Cls	#Gate	Time		
AES-R1-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	149993	749449	24432	2334.24	57340	310569	**7920**	**195.24**	✓	✓
AES-R1-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	172910	589178	24432	2.74	69088	246108	**7920**	**2.57**	✗	✗
AES-R1-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	247606	698415	33104	timeout	86333	329543	**10640**	**1850.07**	✓	✓
AES-R1-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	263989	748906	33104	**5.10**	90428	354018	**10640**	5.28	✗	✗
AES-R2-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	317669	1620894	49152	8473.38	120044	658527	**15872**	**1740.43**	✓	✓
AES-R2-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	380116	1326203	49152	44.67	138266	579518	**15872**	**35.95**	✗	✗
AES-R2-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	501449	1786080	66592	timeout	185415	740484	**21408**	**14323.66**	✓	✓
AES-R2-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	526182	2044293	66592	98.15	194500	807305	**21408**	**83.69**	✗	✗
CRAFT-R1-b1-C	$\zeta(1,1,\mathcal{T},\texttt{1m})$	19233	92262	2948	4.24	6767	33938	**760**	**0.79**	✓	✓
CRAFT-R1-b1-C	$\zeta(2,1,\mathcal{T},\texttt{1m})$	21882	82851	2948	**0.15**	7608	32039	**760**	0.20	✗	✗
CRAFT-R1-b2-C	$\zeta(2,1,\mathcal{T},\texttt{1m})$	148552	558649	19636	2519.00	52901	234385	**5672**	**132.79**	✓	✓
CRAFT-R1-b2-C	$\zeta(3,1,\mathcal{T},\texttt{1m})$	157002	642847	19636	**2.84**	55099	258959	**5672**	3.39	✗	✗
CRAFT-R1-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	5185	23589	766	0.30	2255	10735	**274**	**0.13**	✓	✓
CRAFT-R1-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	6021	21519	766	**0.05**	2567	10316	**274**	0.06	✗	✗
CRAFT-R1-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	8402	31227	1139	1.64	3356	13780	**376**	**0.24**	✓	✓
CRAFT-R1-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	8900	35991	1139	0.07	3538	15058	**376**	**0.06**	✗	✗
CRAFT-R1-b3-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	10236	41474	1296	7.62	3974	19042	**448**	**0.46**	✓	✓
CRAFT-R1-b3-D	$\zeta(4,1,\mathcal{T},\texttt{1m})$	10714	45532	1296	0.07	4132	20428	**448**	0.07	✗	✗
CRAFT-R2-b1-C	$\zeta(1,1,\mathcal{T},\texttt{1m})$	36565	176024	5656	60.97	12644	63451	**1440**	**11.63**	✓	✓
CRAFT-R2-b1-C	$\zeta(2,1,\mathcal{T},\texttt{1m})$	42083	157194	5656	**0.34**	14215	59873	**1440**	0.45	✗	✗
CRAFT-R2-b2-C	$\zeta(2,1,\mathcal{T},\texttt{1m})$	291588	1099486	38872	timeout	104139	452464	**11200**	**2060.81**	✓	✓
CRAFT-R2-b2-C	$\zeta(3,1,\mathcal{T},\texttt{1m})$	308229	1269539	38872	**7.80**	108384	498341	**11200**	10.41	✗	✗
CRAFT-R2-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	9960	45624	1480	4.98	4195	20279	**508**	**0.94**	✓	✓
CRAFT-R2-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	11849	41318	1480	**0.06**	4795	19338	**508**	0.10	✗	✗
CRAFT-R2-b1-D	$\zeta(1,2,\mathcal{T},\texttt{1m})$	9960	45624	1480	7.22	4195	20279	**508**	**1.82**	✓	✓
CRAFT-R2-b1-D	$\zeta(1,3,\mathcal{T},\texttt{1m})$	9959	45622	1480	6.35	4194	20277	**508**	**1.41**	✓	✓
CRAFT-R2-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	16137	59108	2180	60.37	5959	25370	**672**	**3.34**	✓	✓
CRAFT-R2-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	17146	67617	2180	**0.09**	6268	27935	**672**	0.17	✗	✗
CRAFT-R2-b2-D	$\zeta(2,2,\mathcal{T},\texttt{1m})$	16137	59108	2180	91.31	5959	25370	**672**	**3.58**	✓	✓
CRAFT-R3-b3-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	29109	119401	3776	7049.02	10364	48541	**1104**	**37.86**	✓	✓
CRAFT-R3-b3-D	$\zeta(4,1,\mathcal{T},\texttt{1m})$	30481	131197	3776	**0.25**	10760	51937	**1104**	0.31	✗	✗
CRAFT-R3-b3-D	$\zeta(3,2,\mathcal{T},\texttt{1m})$	29111	119420	3776	3547.68	10366	48560	**1104**	**57.74**	✓	✓
CRAFT-R3-b3-D	$\zeta(3,3,\mathcal{T},\texttt{1m})$	29109	119401	3776	2361.28	10364	48541	**1104**	**37.04**	✓	✓
CRAFT-R3-b3-D	$\zeta(3,4,\mathcal{T},\texttt{1m})$	29108	119397	3776	1513.75	10363	48537	**1104**	**31.84**	✓	✓
CRAFT-R4-b3-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	38299	158236	4992	6074.49	13516	63332	**1440**	**62.07**	✓	✓
CRAFT-R4-b3-D	$\zeta(4,1,\mathcal{T},\texttt{1m})$	40118	173925	4992	**0.76**	14039	67789	**1440**	0.77	✗	✗
CRAFT-R4-b3-D	$\zeta(3,2,\mathcal{T},\texttt{1m})$	38301	158267	4992	15014.46	13518	63363	**1440**	**155.75**	✓	✓
CRAFT-R4-b3-D	$\zeta(3,3,\mathcal{T},\texttt{1m})$	38301	158267	4992	5448.20	13518	63363	**1440**	**82.08**	✓	✓
CRAFT-R4-b3-D	$\zeta(3,4,\mathcal{T},\texttt{1m})$	38299	158236	4992	4292.37	13516	63332	**1440**	**71.15**	✓	✓
CRAFT-R4-b3-D	$\zeta(3,5,\mathcal{T},\texttt{1m})$	38298	158229	4992	4009.42	13515	63325	**1440**	**55.16**	✓	✓
LED-R1-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	8306	39057	1312	4.81	2673	13295	**240**	**1.12**	✓	✓
LED-R1-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	9749	31481	1312	0.12	3038	12073	**240**	**0.07**	✗	✗
LED-R1-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	14433	41129	1888	24.79	3726	16057	**336**	**2.09**	✓	✓
LED-R1-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	15296	43876	1888	0.18	3853	16852	**336**	**0.11**	✗	✗
LED-R2-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	16464	79748	2496	65.85	5215	27405	**480**	**9.76**	✓	✓
LED-R2-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	18527	72422	2496	0.25	5783	26245	**480**	**0.17**	✗	✗
LED-R2-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	27208	95236	3536	170.95	7300	34132	**672**	**13.85**	✓	✓
LED-R2-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	28694	106194	3536	0.38	7554	36930	**672**	**0.24**	✗	✗
LED-R3-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	24168	117662	3680	233.72	7674	40655	**720**	**36.71**	✓	✓
LED-R3-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	27669	105378	3680	**0.21**	8528	38988	**720**	0.25	✗	✗
LED-R3-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	39256	140834	5184	343.16	10924	49866	**1008**	**47.99**	✓	✓
LED-R3-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	41301	158823	5184	**0.31**	11305	53487	**1008**	0.37	✗	✗
GIFT-R1-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	5704	26145	912	0.36	1871	9060	**224**	**0.16**	✓	✓
GIFT-R1-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	7061	23130	912	0.18	2157	8554	**224**	**0.12**	✗	✗
GIFT-R1-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	10319	38706	1472	1.19	2975	13226	**336**	**0.33**	✓	✓
GIFT-R1-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	10909	45280	1472	0.36	3101	14616	**336**	**0.17**	✗	✗
GIFT-R2-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	11111	51407	1776	1.16	3584	17658	**432**	**0.42**	✓	✓
GIFT-R2-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	13653	45509	1776	0.41	4169	16645	**432**	**0.20**	✗	✗
GIFT-R2-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	20198	74433	2848	5.29	5490	24697	**624**	**0.96**	✓	✓
GIFT-R2-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	21299	86374	2848	0.77	5743	27294	**624**	**0.38**	✗	✗
PRESENT-R1-b1-D	$\zeta(1,1,\mathcal{T},\texttt{1m})$	7744	36218	1224	0.49	2324	11145	**260**	**0.19**	✓	✓
PRESENT-R1-b1-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	8972	31884	1224	0.21	2619	10768	**260**	**0.11**	✗	✗
PRESENT-R1-b2-D	$\zeta(2,1,\mathcal{T},\texttt{1m})$	14811	49592	1992	1.55	3875	17048	**448**	**0.38**	✓	✓
PRESENT-R1-b2-D	$\zeta(3,1,\mathcal{T},\texttt{1m})$	15649	55766	1992	0.36	4065	19014	**448**	**0.19**	✗	✗

Continued on next page

Table 5 – continued from previous page

Name	Model	Without reduction				With reduction				R	DR
		#Var	#Cls	#Gate	Time	#Var	#Cls	#Gate	Time		
PRESENT-R2-b1-D	$\zeta(1,1,\mathcal{T},\texttt{lm})$	15141	71892	2400	1.71	4519	22601	**504**	**0.58**	✓	✓
PRESENT-R2-b1-D	$\zeta(2,1,\mathcal{T},\texttt{lm})$	17428	63730	2400	0.41	5097	21700	**504**	**0.23**	✗	✗
PRESENT-R2-b2-D	$\zeta(2,1,\mathcal{T},\texttt{lm})$	28676	97703	3888	5.70	7645	32847	**848**	**0.97**	✓	✓
PRESENT-R2-b2-D	$\zeta(3,1,\mathcal{T},\texttt{lm})$	30385	110124	3888	0.75	7978	35924	**848**	**0.39**	✗	✗
SIMON-R1-b1-D	$\zeta(1,1,\mathcal{T},\texttt{lm})$	2123	9582	320	0.15	1346	6162	**176**	**0.13**	✓	✓
SIMON-R1-b1-D	$\zeta(2,1,\mathcal{T},\texttt{lm})$	2564	8797	320	**0.07**	1556	5893	**176**	**0.07**	✗	✗
SIMON-R1-b2-D	$\zeta(2,1,\mathcal{T},\texttt{lm})$	3871	14581	520	0.37	2223	9341	**272**	**0.23**	✓	✓
SIMON-R1-b2-D	$\zeta(3,1,\mathcal{T},\texttt{lm})$	4077	16627	520	0.29	2333	10443	**272**	**0.10**	✗	✗
SIMON-R2-b1-D	$\zeta(1,1,\mathcal{T},\texttt{lm})$	3869	18102	592	0.29	2504	11819	**336**	**0.27**	✓	✓
SIMON-R2-b1-D	$\zeta(2,1,\mathcal{T},\texttt{lm})$	4582	16588	592	0.13	2838	11420	**336**	**0.10**	✗	✗
SIMON-R2-b2-D	$\zeta(2,1,\mathcal{T},\texttt{lm})$	7088	26327	944	1.37	4071	16900	**496**	**0.75**	✓	✓
SIMON-R2-b2-D	$\zeta(3,1,\mathcal{T},\texttt{lm})$	7461	29628	944	0.24	4276	18697	**496**	**0.18**	✗	✗

the vulnerable gate reduction using 8 threads for SAT solving. The results are reported in Table 5, where the columns (#Gate) give the number of vulnerable gates that should be considered when verifying fault-resistance. We can observe that the vulnerable gate reduction can significantly reduce the number of vulnerable gates, achieving over 70% reduction rate on average. Consequently, it significantly reduces the size of the resulting Boolean formulas. Interestingly, reducing the size of the resulting Boolean formula does not necessarily improve the overall verification efficiency. Indeed, the vulnerable gate reduction is very effective in proving fault-resistant benchmarks irrespective of the adopted countermeasure, fault-resistance model, fault-resistance and size of the benchmarks, but sightly worsens the performance for disproving non-fault-resistant benchmarks as they are easy to disprove and the vulnerable gate reduction itself has some overhead.

6 Related work

Various equivalence checking techniques have been proposed such as SAT/SMT-based ones (e.g., [GPB01, KH03, KSHK07, AA17, MCBE06]) and BDD-based ones (e.g., [Pix92, vE98, Bry86, KK97]). Similarly, SAT/SMT-based (e.g., [BCCZ99, MSKM16, JKSC08, BCC+99, BCF+07, BM10, Bra11, LS14]) and BDD-based (e.g., [BCM+90, BCL+94, CG18]) safety verification techniques have been proposed via model-checking, where safety properties are expressed as assertions using temporal logic. However, they are orthogonal to our work and these approaches cannot be *directly* applied to check fault-resistance, though our SAT encoding method is inspired by the existing SAT-based ones for checking equivalence. Indeed, the fault-resistance problem is significantly different from the functional equivalence problem and cannot be easily expressed as a safety property. Moreover, our vulnerable gate reduction which is very effective in improving verification efficiency cannot be leveraged using existing tools.

Due to the severity of fault injection attacks, many simulation- and SAT-based approaches have been proposed to find effective fault vectors or check the effectiveness of a user-specified fault vector (e.g., [SKK13, BGE+17, SMD18, AWMN20, KRH17, SSR+20, WLR+21, NOV+22, BDF+14, FM14, WMGF18, WGF19, WGF20]). Though promising, it is infeasible, if not impossible, to verify a given circuit considering all possible fault vectors that could occur under all valid input combinations. To fill this gap, a BDD-based approach FIVER was proposed [RBSS+21], which exclusively focuses on fault-resistance verification.

In general, for each possible fault vector, FIVER builds a BDD model to represent the concrete faulty circuit w.r.t. the fault vector and analyzes fault-resistance w.r.t. the fault vector by comparing the BDD model with the BDD model of the original circuit. FIVER can efficiently find all the effective fault vectors when the BDD model is successfully constructed. For a fair comparison, we adopted the early-abort strategy of FIVER in our experiments which stops the analysis process when an effective fault vector is found. FIVER can also be used to compute the maximal number of fault events per clock cycle for which the circuit is fault-resistant. Though several optimizations were proposed, it suffers from the combinatorial exploration problem with the increase of fault types, vulnerable gates and clock cycles, in particular, it may fail to construct BDD models when the size of the circuit increases, thus is limited in efficiency and scalability. In contrast, our goal is to prove/disprove fault-resistance instead of finding all the effective fault vectors w.r.t. a given fault-resistance model or computing the maximal number of fault events per clock cycle for which the circuit is fault-resistant. We propose to reduce the problem to SAT solving instead of using BDD to fully utilize the conflict-driven clause learning feature of modern SAT solvers. Thanks to our novel fault event encoding method and effective vulnerable gate reduction, our approach does not need to explicitly enumerate all the possible fault vectors. Thus, FIRMER scales very well and is significantly more efficient than FIVER on relatively larger benchmarks. In summary, the BDD-based approach, FIVER, is exclusively able to find all the effective fault injections and compute the maximal number of fault events per clock cycle, and is suitable for proving/disproving fault-resistance on small benchmarks. Instead, our SAT-based approach, FIRMER, is better particularly on relatively large benchmarks, but currently only reports one effective fault vector if the circuit is not fault-resistant, and cannot compute the maximal number of fault events per clock cycle for which the circuit is fault-resistant. We remark that with our fault event encoding method (without vulnerable gate reduction), one could generate all effective fault vectors by applying ALLSAT (e.g. [LMZY22]) on the resulting Boolean formula, though it may be costly. Indeed, ALLSAT can find all the satisfying assignments for a given Boolean formula. Our approach could also be used for computing the maximal number of fault events per clock cycle for which the circuit is fault-resistant by iteratively increasing this number from 1. We leave them as interesting future work.

In the concurrent work [TAC+23], an approach was proposed for formal analysis of a full system composed of hardware and software components under microarchitecture-level fault injections. In contrast, we focus on cryptographic circuits against gate-level fault injections. Besides this, there are several key differences with our work. (1) The extension of fault injections to combinational circuits is not formalized in [TAC+23] although implemented in their tool; we present the detailed encoding. (2) [TAC+23] considered different fault types for verifying reachability properties but only added an extra variable to control the fault injection; we additionally used variables to choose fault types for verifying equivalence-like properties. Explicitly enumerating all the combinations of fault types for full coverage and equivalence-like properties using [TAC+23] is not scalable. (3) [TAC+23] relies on off-the-shelf SMT-based model-checkers; we directly reduce the problem to SAT/SMT solving. (4) We further present the coNP-completeness result and a novel vulnerable gate reduction which is very effective in proving fault-resistant benchmarks.

Countermeasure synthesis techniques have been proposed to repair flaws (e.g., [EWW16, WLR+21, RRHB20]). However, they do not provide security guarantees (e.g., [WLR+21, RRHB20]) or are limited to one specific type of fault injection attacks (e.g., clock glitches in [EWW16]) and thus are still vulnerable to other fault injection attacks.

7 Conclusion

We have formalized the fault-resistance verification problem and proved that it is coNP-complete for the first time. We proposed novel fault encoding and SAT encoding methods to reduce the fault-resistance verification problem to the SAT problem so that state-of-the-art SAT solvers can be harnessed. We also presented a novel vulnerable gate reduction technique to effectively reduce the number of vulnerable gates, which can significantly improve verification efficiency. We implemented our approach in an open-source tool FIRMER which has been extensively evaluated on a set of real-world cryptographic circuits, demonstrating its effectiveness and efficiency. Experimental results showed that our approach significantly outperforms the state-of-the-art fault-resistance verification approaches. Our tool enables hardware designers to assess and verify the countermeasures in a systematic and automatic way, thus improving the development and security of cryptographic circuits.

For future research, it would be interesting to develop automated flaw repair techniques by leveraging the verification approaches from our approach. Moreover, the current work targets cryptographic circuits, but it would be interesting to extend our approach to general circuits, in particular, CPU designs, and furthermore, full systems composed of CPU design and software components.

References

[AA17] Mohammad Reza Azarbad and Bijan Alizadeh. Scalable smt-based equivalence checking of nested loop pipelining in behavioral synthesis. *ACM Trans. Design Autom. Electr. Syst.*, 22(2):22:1–22:22, 2017.

[ADN$^+$10] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. When clocks fail: On critical paths and clock faults. In *Proceedings of the 9th IFIP WG 8.8/11.2 International Conference (CARDIS)*, pages 182–193, 2010.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Comput. Networks*, 54(15):2787–2805, 2010.

[AMR$^+$20] Anita Aghaie, Amir Moradi, Shahram Rasoolzadeh, Aein Rezaei Shahmirzadi, Falk Schellenberg, and Tobias Schneider. Impeccable circuits. *IEEE Transactions on Computers*, 69:361–376, 2020.

[AS18] Gilles Audemard and Laurent Simon. On the glucose SAT solver. *International Journal on Artificial Intelligence Tools*, 27(1):1840001:1–1840001:25, 2018.

[AWMN20] Victor Arribas, Felix Wegener, Amir Moradi, and Svetla Nikova. Cryptographic fault diagnosis using verfi. In *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 229–240, 2020.

[Bak22] Anubhab Baksi. *Classical and Physical Security of Symmetric Key Cryptographic Algorithms*. Springer Singapore, 2022.

[BBK$^+$03] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. *IEEE Trans. Computers*, 52(4):492–505, 2003.

[BCC$^+$99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Masahiro Fujita, and Yunshan Zhu. Symbolic model checking using SAT procedures instead of bdds. In *Proceedings of the 36th Conference on Design Automation (DAC)*, pages 317–320, 1999.

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, pages 193–207, 1999.

[BCF$^+$07] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, Ziyad Hanna, Alexander Nadel, Amit Palti, and Roberto Sebastiani. A lazy and layered smt($\mathcal{BV}$) solver for hard industrial verification problems. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV)*, pages 547–560, 2007.

[BCL$^+$94] Jerry R. Burch, Edmund M. Clarke, David E. Long, Kenneth L. McMillan, and David L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 13(4):401–424, 1994.

[BCM$^+$90] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10ˆ20 states and beyond. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS)*, pages 428–439, 1990.

[BDF$^+$14] Gilles Barthe, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Jean-Christophe Zapalowicz. Synthesis of fault attacks on cryptographic implementations. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 1016–1027, 2014.

[BFFH20] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proceedings of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

[BGE$^+$17] Jan Burchard, Mael Gay, Ange-Salomé Messeng Ekossono, Jan Horácek, Bernd Becker, Tobias Schubert, Martin Kreuzer, and Ilia Polian. Autofault: Towards automatic construction of algebraic fault attacks. In *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 65–72, 2017.

[BHL17] Jakub Breier, Xiaolu Hou, and Yang Liu. Fault attacks made easy: Differential fault analysis automation on assembly code. Cryptology ePrint Archive, Report 2017/829, 2017.

[BM10] Robert K. Brayton and Alan Mishchenko. ABC: an academic industrial-strength verification tool. In *Proceedings of the 22nd International Conference on Computer Aided Verification (CAV)*, pages 24–40, 2010.

[Bra11] Aaron R. Bradley. Sat-based model checking without unrolling. In *Proceedings of the 12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pages 70–87, 2011.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Proceedings of the 17th Annual International Cryptology Conference Santa Barbara (CRYPTO)*, pages 513–525, 1997.

[CG18] Sagar Chaki and Arie Gurfinkel. Bdd-based symbolic model checking. *Handbook of Model Checking*, pages 219–245, 2018.

[Cla07a] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 181–194, 2007.

[Cla07b] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems*, 2007.

[CLFT14] Franck Courbon, Philippe Loubet-Moundi, Jacques J. A. Fournier, and Assia Tria. Adjusting laser injections for fully controlled faults. In *Proceedings of the 5th International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, pages 229–242, 2014.

[DBC+18] Jean-Max Dutertre, Vincent Beroulle, Philippe Candelier, Stephan De Castro, Louis-Barthelemy Faber, Marie-Lise Flottes, Philippe Gendrier, David Hély, Régis Leveugle, Paolo Maistri, Giorgio Di Natale, Athanasios Papadimitriou, and Bruno Rouzeyre. Laser fault injection at the CMOS 28 nm technology node: an analysis of the fault model. In *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 1–6, 2018.

[DDRT12] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. Electromagnetic transient faults injection on a hardware and a software implementations of AES. In *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 7–15, 2012.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical ineffective fault attacks on masked AES with fault countermeasures. In *Proceedings of the 24th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 315–342, 2018.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: Exploiting ineffective fault inductions on symmetric cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018:547–572, 2018.

[DLM19] Mathieu Dumont, Mathieu Lisart, and Philippe Maurine. Electromagnetic fault injection : How faults occur. In *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 9–16, 2019.

[DLM21] Mathieu Dumont, Mathieu Lisart, and Philippe Maurine. Modeling and simulating electromagnetic fault injection. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 40(4):680–693, 2021.

[dMB08] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

[Dut14] Bruno Dutertre. Yices 2.2. In Armin Biere and Roderick Bloem, editors, *Proceedings of the 26th International Conference on Computer Aided Verification*, volume 8559, pages 737–744, 2014.

[Dut20] Bruno Dutertre. An empirical evaluation of SAT solvers on bit-vector problems. In François Bobot and Tjark Weber, editors, *Proceedings of the 18th International Workshop on Satisfiability Modulo Theories*, volume 2854, pages 15–25, 2020.

[ELH$^+$15] Sho Endo, Yang Li, Naofumi Homma, Kazuo Sakiyama, Kazuo Ohta, Daisuke Fujimoto, Makoto Nagata, Toshihiro Katashita, Jean-Luc Danger, and Takafumi Aoki. A silicon-level countermeasure against fault sensitivity analysis and its evaluation. *IEEE Transactions on Very Large Scale Integration Systems*, 23(8):1429–1438, 2015.

[ES06] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.

[ESH$^+$11] Sho Endo, Takeshi Sugawara, Naofumi Homma, Takafumi Aoki, and Akashi Satoh. An on-chip glitchy-clock generator for testing fault injection attacks. *Journal of Cryptographic Engineering*, 1(4):265–270, 2011.

[EWW16] Hassan Eldib, Meng Wu, and Chao Wang. Synthesis of fault-attack countermeasures for cryptographic circuits. In *Proceedings of the 28th International Conference on Computer Aided Verification (CAV)*, pages 343–363, 2016.

[FJLT13] Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 108–118, 2013.

[FM14] Masahiro Fujita and Alan Mishchenko. Efficient sat-based ATPG techniques for all multiple stuck-at faults. In *Proceedings of the International Test Conference (ITC)*, pages 1–10, 2014.

[GAS14] Nahid Farhady Ghalaty, Aydin Aysu, and Patrick Schaumont. Analyzing and eliminating the causes of fault sensitivity analysis. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition DATE)*, pages 1–6, 2014.

[GD07] Vijay Ganesh and David L. Dill. A decision procedure for bit-vectors and arrays. In *Proceedings of the 19th International Conference on Computer Aided Verification*, pages 519–531, 2007.

[GPB01] Evguenii I. Goldberg, Mukul R. Prasad, and Robert K. Brayton. Using SAT for combinational equivalence checking. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 114–121, 2001.

[JKSC08] Himanshu Jain, Daniel Kroening, Natasha Sharygina, and Edmund M. Clarke. Word-level predicate-abstraction and refinement techniques for verifying RTL verilog. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 27(2):366–379, 2008.

[KH03] Zurab Khasidashvili and Ziyad Hanna. Sat-based methods for sequential hardware equivalence verification without synchronization. In *Proceedings of the 1st International Workshop on Bounded Model Checking*, pages 593–607, 2003.

[KK97] Andreas Kuehlmann and Florian Krohm. Equivalence checking using cuts and heaps. In *Proceedings of the 34st Conference on Design Automation (DAC)*, pages 263–268, 1997.

[KKG03] Ramesh Karri, Grigori Kuznetsov, and Michael Gössel. Parity-based concurrent error detection of substitution-permutation network block ciphers. In *Proceedings of the 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 113–124, 2003.

[KRH17] Punit Khanna, Chester Rebeiro, and Aritra Hazra. Xfc: a framework for exploitable fault characterization in block ciphers. In *Proceedings of the 54th Annual Design Automation Conference (DAC)*, pages 1–6, 2017.

[KSHK07] Daher Kaiss, Marcelo Skaba, Ziyad Hanna, and Zurab Khasidashvili. Industrial strength sat-based alignability algorithm for hardware equivalence verification. In *Proceedings of the 7th International Conference on Formal Methods in Computer-Aided Design*, pages 20–26, 2007.

[KvE02] Andreas Kuehlmann and Cornelis AJ van Eijk. Combinational and sequential equivalence checking. *Logic synthesis and Verification*, pages 343–372, 2002.

[LMZY22] Jiaxin Liang, Feifei Ma, Junping Zhou, and Minghao Yin. AllSATCC: Boosting allsat solving with efficient component analysis. In Luc De Raedt, editor, *Proceedings of the 31st International Joint Conference on Artificial Intelligence*, pages 1866–1872, 2022.

[LS14] Suho Lee and Karem A. Sakallah. Unbounded scalable verification based on approximate property-directed reachability and datapath abstraction. In *Proceedings of the 26th International Conference on Computer Aided Verification (CAV)*, pages 849–865, 2014.

[MCBE06] Alan Mishchenko, Satrajit Chatterjee, Robert K. Brayton, and Niklas Eén. Improvements to combinational equivalence checking. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 836–843, 2006.

[MSKM16] Rajdeep Mukherjee, Peter Schrammel, Daniel Kroening, and Tom Melham. Unbounded safety verification for hardware using software analyzers. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1152–1155, 2016.

[MSY06] Tal Malkin, François-Xavier Standaert, and Moti Yung. A comparative cost/security analysis of fault attack countermeasures. In *Proceedings of the 3rd International Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 159–172, 2006.

[NIS22] NIST. Projects/programs: Lightweight cryptography. https://www.nist.gov/programs-projects/lightweight-cryptography, 2022.

[NOV+22] Pascal Nasahl, Miguel Osorio, Pirmin Vogel, Michael Schaffner, Timothy Trippel, Dominic Rizzo, and Stefan Mangard. SYNFI: pre-silicon fault analysis of an open-source secure element. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):56–87, 2022.

[NP23] Aina Niemetz and Mathias Preiner. Bitwuzla. In Constantin Enea and Akash Lal, editors, *Proceedings of the 35th International Conference on Computer Aided Verification*, pages 3–17, 2023.

[Pix92] Carl Pixley. A theory and implementation of sequential hardware equivalence. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 11(12):1469–1478, 1992.

[RBSG23] Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. Revisiting fault adversary models - hardware faults in theory and practice. *IEEE Transactions on Computers*, 72:572–585, 2023.

[RBSS+21] Jan Richter-Brockmann, Aein Rezaei Shahmirzadi, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. Fiver - robust verification of countermeasures against fault injections. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021:447–473, 2021.

[RRHB20] Indrani Roy, Chester Rebeiro, Aritra Hazra, and Swarup Bhunia. SAFARI: automatic synthesis of fault-attack resistant block cipher implementations. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 39(4):752–765, 2020.

[RSDT13] Cyril Roscian, Alexandre Sarafianos, Jean-Max Dutertre, and Assia Tria. Fault model analysis of laser-induced faults in SRAM memory cells. In *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 89–98, 2013.

[SA03] Sergei P Skorobogatov and Ross J Anderson. Optical fault induction attacks. In *Proceedings of the 4th International Workshop Redwood Shores on Cryptographic Hardware and Embedded Systems (CHES)*, pages 2–12, 2003.

[SFG+16] Falk Schellenberg, Markus Finkeldey, Nils Gerhardt, Martin Hofmann, Amir Moradi, and Christof Paar. Large laser spots and fault sensitivity analysis. In *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 203–208, 2016.

[SGD08] Nidhal Selmane, Sylvain Guilley, and Jean-Luc Danger. Practical setup time violation attacks on AES. In *Proceedings of the 7th European Dependable Computing Conference (EDCC)*, pages 91–96, 2008.

[SHO19] Bodo Selmke, Florian Hauschild, and Johannes Obermaier. Peak clock: Fault injection into pll-based systems via clock manipulation. In *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop (ASHES@CCS)*, pages 85–94, 2019.

[SKK13] Aleksandar Simevski, Rolf Kraemer, and Milos Krstic. Automated integration of fault injection into the ASIC design flow. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 255–260, 2013.

[SMD18] Sayandeep Saha, Debdeep Mukhopadhyay, and Pallab Dasgupta. Expfault: An automated framework for exploitable fault characterization in block ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(2):242–276, 2018.

[SRM20] Aein Rezaei Shahmirzadi, Shahram Rasoolzadeh, and Amir Moradi. Impeccable circuits ii. *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[SSR+20] Milind Srivastava, Patanjali Slpsk, Indrani Roy, Chester Rebeiro, Aritra Hazra, and Swarup Bhunia. Solomon: An automated framework for detecting fault attack vulnerabilities in hardware. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 310–313, 2020.

[TAC+23] Simon Tollec, Mihail Asavoae, Damien Couroussé, Karine Heydemann, and Mathieu Jan. μarchifi: Formal modeling and verification strategies for microarchitectural fault injections. In Alexander Nadel and Kristin Yvonne Rozier, editors, *Proceedings of the Formal Methods in Computer-Aided Design*, pages 101–109, 2023.

[TS21] Amit Kumar Tyagi and N. Sreenath. Cyber physical systems: Analyses, challenges and possible solutions. *Internet of Things and Cyber-Physical Systems*, 1:22–33, 2021.

[vE98] C. A. J. van Eijk. Sequential equivalence checking without state space traversal. In *Proceedings of Design, Automation and Test in Europe (DATE)*, pages 618–623, 1998.

[WGF19] Peikun Wang, Amir Masoud Gharehbaghi, and Masahiro Fujita. Automatic test pattern generation for double stuck-at faults based on test patterns of single faults. In *Proceedings of the 20th International Symposium on Quality Electronic Design (ISQED)*, pages 284–290, 2019.

[WGF20] Peikun Wang, Amir Masoud Gharehbaghi, and Masahiro Fujita. An automatic test pattern generation method for multiple stuck-at faults by incrementally extending the test patterns. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 39(10):2990–2999, 2020.

[WLR$^+$21] Huanyu Wang, Henian Li, Fahim Rahman, Mark M Tehranipoor, and Farimah Farahmandi. SoFI: Security property-driven vulnerability assessments of ICs against fault-injection attacks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(3):452–465, 2021.

[WMGF18] Peikun Wang, Conrad J. Moore, Amir Masoud Gharehbaghi, and Masahiro Fujita. An ATPG method for double stuck-at faults by analyzing propagation paths of single faults. *IEEE Trans. Circuits Syst. I Regul. Pap.*, 65-I(3):1063–1074, 2018.

[Wyn18] Ed Wynn. A comparison of encodings for cardinality constraints in a SAT solver. *CoRR*, abs/1810.12975, 2018.

[ZBL$^+$15] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. *SCIENCE CHINA Information Sciences*, 58(12):1–15, 2015.

[ZDCT13] Loic Zussa, Jean-Max Dutertre, Jessy Clediere, and Assia Tria. Power supply glitch induced faults on fpga: An in-depth analysis of the injection mechanism. In *Proceedings of the IEEE 19th International On-Line Testing Symposium (IOLTS)*, pages 110–115, 2013.

A Detailed Results of the Number of Threads of Section 5

The detailed experimental results of AES-R1-b1-D, AES-R1-b2-D, and CRAFT-R4-b3-D by varying the number of threads from 1 to 12 are reported in Table 6.

Table 6: Results of verifying fault-resistance of AES-R1-b1-D, AES-R1-b2-D, and CRAFT-R4-b3-D using FIRMER (SAT) by varying the number of threads.

	FIRMER (**SAT**)							**SMT**
#Thread	**1**	**2**	**4**	**6**	**8**	**10**	**12**	
AES-R1-b1-$\mathtt{n}_e$1	1234.31	662.06	385.52	258.38	195.24	179.58	**146.25**	4856.58
AES-R1-b1-$\mathtt{n}_e$2	**2.34**	2.39	2.44	2.50	2.57	2.65	2.70	3650.83
AES-R1-b2-$\mathtt{n}_e$2	11028.35	6218.07	3317.16	2329.37	1850.07	1536.98	**1364.48**	23542.57
AES-R1-b2-$\mathtt{n}_e$3	5.16	**5.00**	5.05	5.21	5.28	5.41	5.46	3386.38
CRAFT-R4-b3-$\mathtt{n}_e$3-$\mathtt{n}_c$1	453.57	264.05	136.66	94.12	62.07	55.14	**45.52**	1188.03
CRAFT-R4-b3-$\mathtt{n}_e$4-$\mathtt{n}_c$1	0.73	**0.71**	0.74	0.76	0.77	0.80	0.81	8.25
CRAFT-R4-b3-$\mathtt{n}_e$3-$\mathtt{n}_c$2	998.18	524.15	295.33	206.91	155.75	136.38	**127.18**	978.78
CRAFT-R4-b3-$\mathtt{n}_e$3-$\mathtt{n}_c$3	577.90	290.44	150.73	102.40	82.08	66.91	**55.21**	900.28
CRAFT-R4-b3-$\mathtt{n}_e$3-$\mathtt{n}_c$4	398.52	217.15	120.24	76.40	71.15	54.40	**45.34**	916.13
CRAFT-R4-b3-$\mathtt{n}_e$3-$\mathtt{n}_c$5	396.56	208.42	103.70	75.11	55.16	46.08	**37.08**	922.49

B Fault-resistance Verification with Fault Type τ_{bf} Only

In this section, in additional to the previous section, we evaluate FIRMER for verifying fault-resistance with the fault type τ_{bf} solely.

B.1 Efficiency of FIRMER for Fault-resistance Verification

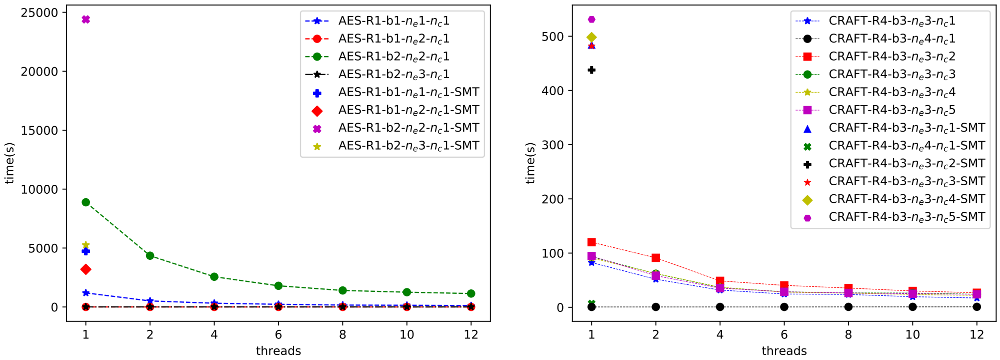
We compare FIRMER with (1) the state-of-the-art verifier FIVER and (2) an SMT-based approach which directly checks the constraints generated by our encoding method without translating to Boolean formulas.

The results are reported in Table 7, where both the SAT solver Glucose and the BDD-based verifier FIVER are run with 8 threads while the SMT solver Bitwuzla is run with a single thread. Columns (2CNF) and (Solving) give the execution time of building and solving Boolean formulas in seconds, respectively. Columns (Total) and (Time) give the total execution time in seconds. Mark ✓ (resp. ✗) indicates that the protected circuit $\mathcal{S}'$ is fault-resistant (resp. not fault-resistant) w.r.t. the given fault-resistance model in column “Model".

Overall, FIRMER (i.e., SAT-based approach) solved all the verification tasks, the SMT-based approach ran out of time on one verification task, while FIVER ran out of time on 30 verification tasks. The SAT/SMT-based approach becomes more and more efficient than FIVER with the increase of round numbers (i.e., Ri) and the maximal number of protected faulty bits (i.e., bj). FIRMER is still significantly more efficient than the SMT-based approach on relatively larger benchmarks (e.g., AES-R1-b1, AES-R1-b2, AES-R2-b1, AES-R2-b2, CRAFT-R1-b2-C, CRAFT-R2-b2-C, CRAFT-R3-b3-D, CRAFT-R4-b3-D, LED-R2-b1-D, LED-R2-b2-D, LED-R3-b1-D, LED-R3-b2-D, etc.) while they are comparable on smaller benchmarks.

Interestingly, we found that (i) implementations with correction-based countermeasures are more difficult to prove than those with detection-based countermeasures (e.g., CRAFT-Ri-bj-C vs. CRAFT-Ri-bj-D, for $i = 1, 2$ and $j = 1, 2$), because implementing correction-based countermeasures require more gates; (ii) FIRMER is more efficient at disproving fault-resistance than proving fault-resistance, because UNSAT instances are often more difficult to prove than SAT instances in CDCL SAT solvers. (iii) FIRMER often scales very well with an increase of the round numbers (i.e., Ri for $i = 1, 2, 3, 4$), the maximal number of protected faulty bits (i.e., bj for $j = 1, 2, 3$), the maximum number of fault events per clock cycle (i.e., $\mathtt{n}_e$) and the maximum number of clock cycles in which fault events can occur (i.e., $\mathtt{n}_c$), but FIVER has very limited scalability.

To understand the effect of the number of threads, we evaluate FIRMER on the



(a) AES-R1-b1-D and AES-R1-b2-D (b) CRAFT-R4-b3-D

Figure 9: Results with different number of threads and only the fault type τ_{bf}

Table 7: Verification results with only the fault type τ_{bf}.

Name	Model	FIRMER (SAT)					SMT	FIVER	R	DR
		#Var	# Cls	2CNF	Solving	Total	Time	Time		
AES-R1-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	34822	246273	1.87	159.44	161.31	4728.67	**24.50**	✓	✓
AES-R1-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	46570	181812	2.33	0.41	**2.74**	3192.99	10154.53	✗	✗
AES-R1-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	56196	243722	4.50	1397.42	**1401.92**	24397.81	timeout	✓	✓
AES-R1-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	60291	268197	4.44	0.66	**5.10**	5254.88	timeout	✗	✗
AES-R2-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	74678	528469	33.18	1132.13	**1165.31**	47809.92	timeout	✓	✓
AES-R2-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	92900	449460	33.25	3.13	**36.38**	20206.54	timeout	✗	✗
AES-R2-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	124574	565820	78.85	5905.02	**5983.87**	timeout	timeout	✓	✓
AES-R2-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	133659	632641	79.25	6.27	**85.52**	88.75	timeout	✗	✗
CRAFT-R1-b1-C	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	4326	26659	0.09	0.64	0.73	5.87	**0.10**	✓	✓
CRAFT-R1-b1-C	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	5167	24760	0.09	0.06	**0.15**	0.16	1.03	✗	✗
CRAFT-R1-b2-C	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	35997	188158	2.11	142.04	**144.15**	2586.82	175.84	✓	✓
CRAFT-R1-b2-C	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	38195	212732	2.24	0.56	**2.80**	9.79	958.40	✗	✗
CRAFT-R1-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	1347	7890	0.02	0.16	0.18	0.36	**0.03**	✓	✓
CRAFT-R1-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	1659	7471	0.03	0.02	0.05	**0.01**	0.06	✗	
CRAFT-R1-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	2165	10140	0.03	0.31	0.34	0.61	**0.17**	✓	✓
CRAFT-R1-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	2347	11418	0.04	0.04	0.08	**0.01**	8.64	✗	✗
CRAFT-R1-b3-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	2572	14530	0.04	0.64	**0.68**	5.12	310.40	✓	✓
CRAFT-R1-b3-D	$\zeta(4, 1, \tau_{bf}, \mathtt{lm})$	2730	15916	0.04	0.05	**0.09**	0.58	1724.35	✗	✗
CRAFT-R2-b1-C	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	8008	49988	0.18	13.44	13.62	91.76	**2.09**	✓	✓
CRAFT-R2-b1-C	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	9579	46410	0.20	0.13	**0.33**	0.42	41.27	✗	✗
CRAFT-R2-b2-C	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	70834	361687	5.50	1891.50	1897.00	12779.36	**1770.58**	✓	✓
CRAFT-R2-b2-C	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	75079	407564	5.60	2.20	**7.80**	2464.15	3317.16	✗	✗
CRAFT-R2-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	2595	15155	0.04	0.91	0.95	1.89	**0.64**	✓	✓
CRAFT-R2-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	3195	14214	0.03	0.04	0.07	**0.04**	1.05	✗	✗
CRAFT-R2-b1-D	$\zeta(1, 2, \tau_{bf}, \mathtt{lm})$	2595	15155	0.05	0.85	**0.90**	3.63	124.40	✓	✓
CRAFT-R2-b1-D	$\zeta(1, 3, \tau_{bf}, \mathtt{lm})$	2594	15153	0.06	0.62	**0.68**	3.53	25815.91	✓	✓
CRAFT-R2-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	4015	19410	0.09	0.94	**1.03**	6.03	6.50	✓	✓
CRAFT-R2-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	4324	21975	0.05	0.06	0.11	**0.05**	89.51	✗	✗
CRAFT-R2-b2-D	$\zeta(2, 2, \tau_{bf}, \mathtt{lm})$	4015	19410	0.14	1.02	**1.16**	8.63	timeout	✓	✓
CRAFT-R3-b3-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	7066	38313	0.17	13.11	**13.28**	241.14	timeout	✓	✓
CRAFT-R3-b3-D	$\zeta(4, 1, \tau_{bf}, \mathtt{lm})$	7462	41709	0.17	0.11	**0.28**	3.19	timeout	✗	✗
CRAFT-R3-b3-D	$\zeta(3, 2, \tau_{bf}, \mathtt{lm})$	7068	38332	0.17	15.08	**15.25**	243.42	timeout	✓	✓
CRAFT-R3-b3-D	$\zeta(3, 3, \tau_{bf}, \mathtt{lm})$	7066	38313	0.18	16.36	**16.54**	242.72	timeout	✓	✓
CRAFT-R3-b3-D	$\zeta(3, 4, \tau_{bf}, \mathtt{lm})$	7065	38309	0.15	14.32	**14.47**	241.82	timeout	✓	✓
CRAFT-R4-b3-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	9209	50064	0.60	23.10	**23.70**	717.94	timeout	✓	✓
CRAFT-R4-b3-D	$\zeta(4, 1, \tau_{bf}, \mathtt{lm})$	9732	54521	0.62	0.16	**0.78**	6.87	timeout	✗	✗
CRAFT-R4-b3-D	$\zeta(3, 2, \tau_{bf}, \mathtt{lm})$	9211	50095	0.59	34.93	**35.52**	667.81	timeout	✓	✓
CRAFT-R4-b3-D	$\zeta(3, 3, \tau_{bf}, \mathtt{lm})$	9211	50095	0.56	26.07	**26.63**	671.85	timeout	✓	✓
CRAFT-R4-b3-D	$\zeta(3, 4, \tau_{bf}, \mathtt{lm})$	9209	50064	0.58	25.43	**26.01**	688.15	timeout	✓	✓
CRAFT-R4-b3-D	$\zeta(3, 5, \tau_{bf}, \mathtt{lm})$	9208	50057	0.57	25.95	**26.52**	691.08	timeout	✓	✓
LED-R1-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	1866	10623	0.06	0.46	0.52	11.63	**0.07**	✓	✓
LED-R1-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	2231	9401	0.05	0.02	0.07	**0.02**	0.74	✗	✗
LED-R1-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	2628	12473	0.09	1.54	1.63	12.01	**1.40**	✓	✓
LED-R1-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	2755	13268	0.07	0.03	0.10	**0.03**	32.71	✗	✗

Continued on next page

Table 7 – continued from previous page

Name	Model	FIRMER (SAT)					SMT	FIVER	R	DR
		#Var	# Cls	2CNF	Solving	Total	Time	Time		
LED-R2-b1-D	$\zeta(1, 1, \tau_{bf}, \texttt{lm})$	3594	21569	0.12	1.68	**1.80**	54.22	timeout	✓	✓
LED-R2-b1-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	4162	20409	0.09	0.07	**0.16**	1.44	timeout	✗	✗
LED-R2-b2-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	5116	26420	0.12	3.21	**3.33**	266.12	timeout	✓	✓
LED-R2-b2-D	$\zeta(3, 1, \tau_{bf}, \texttt{lm})$	5370	29218	0.14	0.08	**0.22**	2.21	timeout	✗	✗
LED-R3-b1-D	$\zeta(1, 1, \tau_{bf}, \texttt{lm})$	5249	31588	0.13	4.28	**4.41**	263.93	timeout	✓	✓
LED-R3-b1-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	6103	29921	0.13	0.11	**0.24**	2.96	timeout	✗	✗
LED-R3-b2-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	7735	38318	0.12	7.79	**7.91**	394.05	timeout	✓	✓
LED-R3-b2-D	$\zeta(3, 1, \tau_{bf}, \texttt{lm})$	8116	41939	0.22	0.15	**0.37**	3.06	timeout	✗	✗
GIFT-R1-b1-D	$\zeta(1, 1, \tau_{bf}, \texttt{lm})$	1214	7028	0.07	0.08	0.15	2.30	**0.02**	✓	✓
GIFT-R1-b1-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	1500	6522	0.07	0.03	0.10	1.50	**0.05**	✗	✗
GIFT-R1-b2-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	2058	10370	0.12	0.13	0.25	5.31	**0.19**	✓	✓
GIFT-R1-b2-D	$\zeta(3, 1, \tau_{bf}, \texttt{lm})$	2184	11760	0.12	0.03	**0.15**	3.45	4.01	✗	✗
GIFT-R2-b1-D	$\zeta(1, 1, \tau_{bf}, \texttt{lm})$	2323	13610	0.13	0.18	0.31	6.82	**0.10**	✓	✓
GIFT-R2-b1-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	2908	12597	0.14	0.04	**0.18**	3.73	0.44	✗	✗
GIFT-R2-b2-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	3817	19365	0.26	0.43	**0.69**	20.48	2.71	✓	✓
GIFT-R2-b2-D	$\zeta(3, 1, \tau_{bf}, \texttt{lm})$	4070	21962	0.27	0.05	0.32	**0.23**	120.08	✗	✗
PRESENT-R1-b1-D	$\zeta(1, 1, \tau_{bf}, \texttt{lm})$	1576	8719	0.08	0.08	0.16	2.61	**0.02**	✓	✓
PRESENT-R1-b1-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	1871	8342	0.08	0.03	0.11	1.67	**0.06**	✗	✗
PRESENT-R1-b2-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	2665	13245	0.14	0.13	0.27	9.25	**0.24**	✓	✓
PRESENT-R1-b2-D	$\zeta(3, 1, \tau_{bf}, \texttt{lm})$	2855	15211	0.15	0.04	**0.19**	9.71	4.19	✗	✗

Continued on next page

Table 8: Results of verifying fault-resistance of AES-R1-b1-D, AES-R1-b2-D, and CRAFT-R4-b3-D using FIRMER (SAT) with only the fault type τ_{bf} by varying the number of threads.

	FIRMER (**SAT**)							SMT
#Thread	1	2	4	6	8	10	12	
AES-R1-b1-$\mathtt{n}_e$1	1192.68	513.15	317.66	226.61	161.31	145.69	**119.36**	4728.67
AES-R1-b1-$\mathtt{n}_e$2	2.71	**2.70**	2.74	2.84	2.74	2.91	3.02	3192.99
AES-R1-b2-$\mathtt{n}_e$2	8882.96	4349.25	2564.76	1796.85	1401.92	1253.49	**1139.42**	24397.81
AES-R1-b2-$\mathtt{n}_e$3	**6.59**	6.78	6.67	6.77	6.67	6.99	7.08	5254.88
CRAFT-R4-b3-$\mathtt{n}_e$3-$\mathtt{n}_c$1	82.61	51.79	31.79	24.56	23.70	19.56	**17.15**	484.17
CRAFT-R4-b3-$\mathtt{n}_e$4-$\mathtt{n}_c$1	**0.67**	**0.67**	0.69	**0.67**	0.78	0.81	0.86	6.87
CRAFT-R4-b3-$\mathtt{n}_e$3-$\mathtt{n}_c$2	120.11	91.31	48.79	40.44	35.52	30.15	**26.83**	437.81
CRAFT-R4-b3-$\mathtt{n}_e$3-$\mathtt{n}_c$3	93.24	62.00	36.55	28.61	26.63	26.40	**24.06**	481.85
CRAFT-R4-b3-$\mathtt{n}_e$3-$\mathtt{n}_c$4	90.88	62.49	37.16	27.81	26.01	23.66	**21.79**	498.15
CRAFT-R4-b3-$\mathtt{n}_e$3-$\mathtt{n}_c$5	94.53	58.37	35.38	28.61	26.52	25.07	**24.04**	531.08

Table 7 – continued from previous page

Name	Model	FIRMER (**SAT**)					SMT	FIVER	R	DR
		#Var	# Cls	2CNF	Solving	Total	Time	Time		
PRESENT-R2-b1-D	$\zeta(1,1,\tau_{bf},\mathtt{lm})$	3074	17622	0.15	0.25	0.40	8.90	**0.13**	✓	✓
PRESENT-R2-b1-D	$\zeta(2,1,\tau_{bf},\mathtt{lm})$	3652	16721	0.16	0.04	**0.20**	1.31	0.77	✗	✗
PRESENT-R2-b2-D	$\zeta(2,1,\tau_{bf},\mathtt{lm})$	5418	25447	0.31	0.43	**0.74**	22.87	3.16	✓	✓
PRESENT-R2-b2-D	$\zeta(3,1,\tau_{bf},\mathtt{lm})$	5751	28524	0.30	0.06	0.36	**0.34**	113.94	✗	✗
SIMON-R1-b1-D	$\zeta(1,1,\tau_{bf},\mathtt{lm})$	781	4788	0.04	0.06	0.10	1.04	**0.01**	✓	✓
SIMON-R1-b1-D	$\zeta(2,1,\tau_{bf},\mathtt{lm})$	991	4519	0.04	0.02	0.06	0.88	**0.04**	✗	✗
SIMON-R1-b2-D	$\zeta(2,1,\tau_{bf},\mathtt{lm})$	1348	7121	0.08	0.11	0.19	2.89	**0.14**	✓	✓
SIMON-R1-b2-D	$\zeta(3,1,\tau_{bf},\mathtt{lm})$	1458	8223	0.08	0.03	**0.11**	1.88	6.21	✗	✗
SIMON-R2-b1-D	$\zeta(1,1,\tau_{bf},\mathtt{lm})$	1312	8555	0.06	0.08	**0.14**	3.46	timeout	✓	✓
SIMON-R2-b1-D	$\zeta(2,1,\tau_{bf},\mathtt{lm})$	1646	8156	0.06	0.03	**0.09**	2.07	timeout	✗	✗
SIMON-R2-b2-D	$\zeta(2,1,\tau_{bf},\mathtt{lm})$	2485	12623	0.12	0.20	**0.32**	16.91	timeout	✓	✓
SIMON-R2-b2-D	$\zeta(3,1,\tau_{bf},\mathtt{lm})$	2690	14420	0.12	0.04	**0.16**	6.97	timeout	✗	✗

benchmarks AES-R1-b1-D, AES-R1-b2-D, and CRAFT-R4-b3-D by varying the number of threads from 1 to 12. The results are depicted in Fig. 9(a) and Fig. 9(b), respectively, where $\mathtt{n}_e i$ and $\mathtt{n}_c j$ denote the fault-resistance mode $\zeta(i, j, \tau_{bf}, \mathtt{lm})$. Detailed results are reported in Table 8. We can observe that FIRMER always outperforms the SMT-based approach. On the fault-resistant benchmarks (i.e., curves with bj-$\mathtt{n}_e k$ such that $j \geq k$), FIRMER becomes more and more efficient while the improvement becomes less and less, with the increase of the number of threads. On the non-fault-resistant benchmarks (i.e., curves with bj-$\mathtt{n}_e k$ such that $j < k$), multi-threading does not improve performance and instead slightly worsens performance, because they are easy to disprove and thread scheduling causes overhead.

B.2 Effectiveness of the Vulnerable Gate Reduction

We evaluate FIRMER with/without the vulnerable gate reduction using 8 threads for SAT solving, considering only the fault type τ_{bf}.

The results are reported in Table 9, where columns (#Gate) give the number of vulnerable gates that should be considered when verifying fault-resistance. We can observe that our vulnerable gate reduction is able to significantly reduce the number of vulnerable gates that should be considered when verifying fault-resistance, 70% reduction rate on average, consequently, significantly reducing the size of the resulting Boolean formulas and accelerating fault-resistance verification, no matter the adopted countermeasure, fault-resistance model, fault-resistance and size of the benchmarks.

Table 9: Results of FIRMER with/without vulnerable gate reduction using only the fault type τ_{bf}.

Name	Model	Without reduction				With reduction				R	DR
		#Var	#Cls	#Gate	Time	#Var	#Cls	#Gate	Time		
AES-R1-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	77675	577979	24432	972.45	34822	246273	**7920**	**161.31**	✓	✓
AES-R1-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	100592	417708	24432	3.52	46570	181812	**7920**	**2.74**	✗	✗
AES-R1-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	149413	466954	33104	timeout	56196	243722	**10640**	**1401.92**	✓	✓
AES-R1-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	165796	517451	33104	6.09	60291	268197	**10640**	**5.10**	✗	✗
AES-R2-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	172147	1277143	49152	4749.92	74678	528469	**15872**	**1165.31**	✓	✓
AES-R2-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	234594	982452	49152	40.11	92900	449460	**15872**	**36.38**	✗	✗
AES-R2-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	303910	1319943	66592	timeout	124574	565820	**21408**	**5983.87**	✓	✓
AES-R2-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	328643	1578156	66592	90.39	133659	632641	**21408**	**85.52**	✗	✗
CRAFT-R1-b1-C	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	9773	70431	2948	3.71	4326	26659	**760**	**0.73**	✓	✓
CRAFT-R1-b1-C	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	12422	61020	2948	0.28	5167	24760	**760**	**0.15**	✗	✗
CRAFT-R1-b2-C	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	88296	415886	19636	2263.07	35997	188158	**5672**	**144.15**	✓	✓
CRAFT-R1-b2-C	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	96746	500084	19636	4.10	38195	212732	**5672**	**2.80**	✗	✗
CRAFT-R1-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	2657	17579	766	0.26	1347	7890	**274**	**0.18**	✓	✓
CRAFT-R1-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	3493	15509	766	0.08	1659	7471	**274**	**0.05**	✗	✗
CRAFT-R1-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	4716	22755	1139	1.32	2165	10140	**376**	**0.34**	✓	✓
CRAFT-R1-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	5214	27519	1139	0.12	2347	11418	**376**	**0.08**	✗	✗
CRAFT-R1-b3-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	5990	31598	1296	10.24	2572	14530	**448**	**0.68**	✓	✓
CRAFT-R1-b3-D	$\zeta(4, 1, \tau_{bf}, \mathtt{lm})$	6468	35656	1296	0.18	2730	15916	**448**	**0.09**	✗	✗
CRAFT-R2-b1-C	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	18430	134017	5656	71.00	8008	49988	**1440**	**13.62**	✓	✓
CRAFT-R2-b1-C	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	23948	115187	5656	0.76	9579	46410	**1440**	**0.33**	✗	✗
CRAFT-R2-b2-C	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	172699	818270	38872	timeout	70834	361687	**11200**	**1897.00**	✓	✓
CRAFT-R2-b2-C	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	189340	988323	38872	10.88	75079	407564	**11200**	**7.80**	✗	✗
CRAFT-R2-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	5169	34088	1480	3.14	2595	15155	**508**	**0.95**	✓	✓
CRAFT-R2-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	7058	29782	1480	0.16	3195	14214	**508**	**0.07**	✗	✗
CRAFT-R2-b1-D	$\zeta(1, 2, \tau_{bf}, \mathtt{lm})$	5169	34088	1480	4.32	2595	15155	**508**	**0.90**	✓	✓
CRAFT-R2-b1-D	$\zeta(1, 3, \tau_{bf}, \mathtt{lm})$	5168	34086	1480	3.59	2594	15153	**508**	**0.68**	✓	✓
CRAFT-R2-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	9294	43390	2180	58.66	4015	19410	**672**	**1.03**	✓	✓
CRAFT-R2-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	10303	51899	2180	0.25	4324	21975	**672**	**0.11**	✗	✗
CRAFT-R2-b2-D	$\zeta(2, 2, \tau_{bf}, \mathtt{lm})$	9294	43390	2180	33.45	4015	19410	**672**	**1.16**	✓	✓
CRAFT-R3-b3-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	17284	92557	3776	1274.24	7066	38313	**1104**	**13.28**	✓	✓
CRAFT-R3-b3-D	$\zeta(4, 1, \tau_{bf}, \mathtt{lm})$	18656	104353	3776	0.43	7462	41709	**1104**	**0.28**	✗	✗
CRAFT-R3-b3-D	$\zeta(3, 2, \tau_{bf}, \mathtt{lm})$	17286	92576	3776	2323.82	7068	38332	**1104**	**15.25**	✓	✓
CRAFT-R3-b3-D	$\zeta(3, 3, \tau_{bf}, \mathtt{lm})$	17284	92557	3776	1615.21	7066	38313	**1104**	**16.54**	✓	✓
CRAFT-R3-b3-D	$\zeta(3, 4, \tau_{bf}, \mathtt{lm})$	17283	92553	3776	1461.67	7065	38309	**1104**	**14.47**	✓	✓
CRAFT-R4-b3-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	22681	122758	4992	2152.98	9209	50064	**1440**	**23.70**	✓	✓
CRAFT-R4-b3-D	$\zeta(4, 1, \tau_{bf}, \mathtt{lm})$	24500	138447	4992	1.08	9732	54521	**1440**	**0.78**	✗	✗
CRAFT-R4-b3-D	$\zeta(3, 2, \tau_{bf}, \mathtt{lm})$	22683	122789	4992	8174.81	9211	50095	**1440**	**35.52**	✓	✓
CRAFT-R4-b3-D	$\zeta(3, 3, \tau_{bf}, \mathtt{lm})$	22683	122789	4992	4473.37	9211	50095	**1440**	**26.63**	✓	✓
CRAFT-R4-b3-D	$\zeta(3, 4, \tau_{bf}, \mathtt{lm})$	22681	122758	4992	3864.15	9209	50064	**1440**	**26.01**	✓	✓
CRAFT-R4-b3-D	$\zeta(3, 5, \tau_{bf}, \mathtt{lm})$	22680	122751	4992	3626.20	9208	50057	**1440**	**26.52**	✓	✓
LED-R1-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	3886	29433	1312	3.87	1866	10623	**240**	**0.52**	✓	✓
LED-R1-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	5329	21857	1312	0.10	2231	9401	**240**	**0.07**	✗	✗
LED-R1-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	8128	27465	1888	27.49	2628	12473	**336**	**1.63**	✓	✓
LED-R1-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	8991	30212	1888	0.20	2755	13268	**336**	**0.10**	✗	✗
LED-R2-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	8139	61143	2496	26.85	3594	21569	**480**	**1.80**	✓	✓
LED-R2-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	10202	53817	2496	0.29	4162	20409	**480**	**0.16**	✗	✗
LED-R2-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	15630	69468	3536	279.56	5116	26420	**672**	**3.33**	✓	✓
LED-R2-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	17116	80426	3536	0.57	5370	29218	**672**	**0.22**	✗	✗
LED-R3-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	11963	90062	3680	63.25	5249	31588	**720**	**4.41**	✓	✓
LED-R3-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	15464	77778	3680	0.45	6103	29921	**720**	**0.24**	✗	✗
LED-R3-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	22448	103306	5184	453.41	7735	38318	**1008**	**7.91**	✓	✓
LED-R3-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	24493	121295	5184	0.69	8116	41939	**1008**	**0.37**	✗	✗
GIFT-R1-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	2959	20017	912	0.30	1214	7028	**224**	**0.15**	✓	✓
GIFT-R1-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	4316	17002	912	0.13	1500	6522	**224**	**0.10**	✗	✗
GIFT-R1-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	5835	28978	1472	1.04	2058	10370	**336**	**0.25**	✓	✓
GIFT-R1-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	6425	35552	1472	0.25	2184	11760	**336**	**0.15**	✗	✗
GIFT-R2-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	5756	39119	1776	0.90	2323	13610	**432**	**0.31**	✓	✓
GIFT-R2-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	8298	33221	1776	0.25	2908	12597	**432**	**0.18**	✗	✗
GIFT-R2-b2-D	$\zeta(2, 1, \mathcal{T}, \mathtt{lm})$	11673	55595	2848	4.41	3817	19365	**624**	**0.69**	✓	✓
GIFT-R2-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	12774	67536	2848	0.53	4070	21962	**624**	**0.32**	✗	✗
PRESENT-R1-b1-D	$\zeta(1, 1, \tau_{bf}, \mathtt{lm})$	4063	27837	1224	0.48	1576	8719	**260**	**0.16**	✓	✓
PRESENT-R1-b1-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	5291	23503	1224	0.17	1871	8342	**260**	**0.11**	✗	✗
PRESENT-R1-b2-D	$\zeta(2, 1, \tau_{bf}, \mathtt{lm})$	8967	36377	1992	1.55	2665	13245	**448**	**0.27**	✓	✓
PRESENT-R1-b2-D	$\zeta(3, 1, \tau_{bf}, \mathtt{lm})$	9805	42551	1992	0.28	2855	15211	**448**	**0.19**	✗	✗

Continued on next page

Table 9 – continued from previous page

Name	Model	Without reduction				With reduction				R	DR
		#Var	#Cls	#Gate	Time	#Var	#Cls	#Gate	Time		
PRESENT-R2-b1-D	$\zeta(1, 1, \tau_{bf}, \texttt{lm})$	7976	55262	2400	1.76	3074	17622	**504**	**0.40**	✓	✓
PRESENT-R2-b1-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	10263	47100	2400	0.33	3652	16721	**504**	**0.20**	✗	✗
PRESENT-R2-b2-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	17371	71719	3888	6.10	5418	25447	**848**	**0.74**	✓	✓
PRESENT-R2-b2-D	$\zeta(3, 1, \tau_{bf}, \texttt{lm})$	19080	84140	3888	0.58	5751	28524	**848**	**0.36**	✗	✗
SIMON-R1-b1-D	$\zeta(1, 1, \tau_{bf}, \texttt{lm})$	1087	7301	320	0.12	781	4788	**176**	**0.10**	✓	✓
SIMON-R1-b1-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	1528	6516	320	0.07	991	4519	**176**	**0.06**	✗	✗
SIMON-R1-b2-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	2182	10860	520	0.30	1348	7121	**272**	**0.19**	✓	✓
SIMON-R1-b2-D	$\zeta(3, 1, \tau_{bf}, \texttt{lm})$	2388	12906	520	0.12	1458	8223	**272**	**0.11**	✗	✗
SIMON-R2-b1-D	$\zeta(1, 1, \tau_{bf}, \texttt{lm})$	1941	13625	592	0.27	1312	8555	**336**	**0.14**	✓	✓
SIMON-R2-b1-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	2654	12111	592	0.11	1646	8156	**336**	**0.09**	✗	✗
SIMON-R2-b2-D	$\zeta(2, 1, \tau_{bf}, \texttt{lm})$	4070	19437	944	0.81	2485	12623	**496**	**0.32**	✓	✓
SIMON-R2-b2-D	$\zeta(3, 1, \tau_{bf}, \texttt{lm})$	4443	22738	944	0.20	2690	14420	**496**	**0.16**	✗	✗