TimingSAT: Decamouflaging Timing-based Logic Obfuscation

Meng Li[1], Kaveh Shamsi[2], Yier Jin[2], and David Z. Pan[1]

[1]ECE Department, University of Texas at Austin, Austin, TX, USA
[2]ECE Department, University of Florida, Gainesville, FL, USA

***Abstract*— In order to counter advanced reverse engineering techniques, various integrated circuit (IC) camouflaging methods are proposed to protect hardware intellectual property (IP) proactively. For example, a timing-based camouflaging strategy is developed recently representing a new class of parametric camouflaging strategies. Unlike traditional IC camouflaging techniques that directly hide the circuit functionality, the new parametric strategies obfuscate the circuit timing schemes, which in turn protects the circuit functionality and invalidates all the existing attacks. In this paper, we propose a SAT attack, named TimingSAT, to analyze the security of such timing-based camouflaging strategies. We demonstrate that with a proper transformation of the camouflaged netlist, traditional SAT attacks are still effective to decamouflage the new protection methods. The correctness of the resolved circuit functionality is formally proved. While a direct implementation of TimingSAT suffers from poor scalability, we propose a simplification procedure to significantly enhance the attack efficiency without sacrificing the correctness of the decamouflaged netlist. The efficiency and effectiveness of TimingSAT is validated with extensive experimental results.**

I. Introduction

Modern ICs are produced in a globalized supply chain, which significantly reduces the cost and design complexity for hardware IP vendors. However, emerging IP security and privacy violations due to reverse engineering come along as well [1]–[3]. By leveraging physical techniques, including chemical etching, high-resolution microscopy and so on, reverse engineering attackers can strip the ICs layer by layer and reconstruct the gate-level circuit netlist, thus compromising the IC confidentiality [3].

IC camouflaging is proposed to protect the hardware IP against reverse engineering proactively [4]–[7]. Traditional camouflaging schemes focus on obfuscating the functionality of the design directly [4]–[6]. By leveraging specific fabrication-level techniques [8], [9], camouflaging cells can be built to implement different Boolean logic functions, even though their layouts look the same to the attackers. The camouflaging cells are then used to synthesize the design to protect the circuit functionality.

However, all the traditional camouflaging schemes suffer from an intrinsic trade-off between the security against the decamouflaging attacks and the output error probability when the functionalities of the camouflaging cells are misinterpreted [5]. Such a trade-off is exploited by existing decamouflaging attacks. As shown in [10]–[14], with the SAT attack and its subsequent enhancements, nearly all the traditional camouflaging schemes can be attacked efficiently.

To remedy the problem of the traditional approaches, timing-based parametric camouflaging methods are proposed recently [15], [16]. Instead of directly hiding the circuit functionality, the parametric camouflaging methods obfuscate the timing behavior of the circuits. This is achieved by inserting buffers with tunable delay [15] or by introducing unconventional timing paths [16]. The obfuscated netlists consist of a combination of single-cycle paths and multi-cycle paths, which, if not resolved correctly, makes the reconstructed circuits malfunction. Therefore, the attackers are forced to determine the timing behavior for each path in the camouflaged netlists.

Timing-based approaches are promising and usually considered secure. Traditional testing-based methods that try to measure the path delay directly have been demonstrated impractical [16]. On one hand, they suffer from poor scalability as the number of timing paths increases exponentially with respect to the circuit size. On the other hand, camouflaging techniques are proposed to hide selected timing paths and hinder the generation of the test vectors [16]. Timing-based approaches are also assumed to be secure against existing query-based attacks, e.g., SAT attacks. This is because existing query-based attacks focus on discriminating different Boolean functions and cannot be applied to determine the circuit timing schemes directly [15]. However, such an assumption remains unverified.

In this paper, we formally study the resilience of the timing-based camouflaging strategies against query-based attacks. We propose the TimingSAT framework and demonstrate that after a proper transformation of the camouflaged netlist, existing SAT attacks are still effective to decamouflage the timing-based strategies. To enable the transformation, we first come up with a novel key-controlled transformation unit (TU) design that is able to convert a timing path, either a single-cycle path or a multi-cycle path, into a single-cycle path without changing its functionality. The TUs are then inserted into the camouflaged netlist to convert all the paths into single-cycle paths with a guarantee that the correct circuit functionality can be achieved when proper keys are inserted.

After the transformation procedure, the decamouflaging task is converted into determining the correct keys for the inserted TUs. Traditional SAT attacks cannot be directly

978-1-5386-8382-8/18/$31.00

applied due to the existence of uncontrollable flip flops in the inserted TUs. Therefore, we propose to unroll the transformed circuit for several time frames and apply the SAT attack to the unrolled netlist. The number of unrolling time frames is formally derived to guarantee the correctness of the resolved keys. While a direct implementation of TimingSAT can suffer from scalability issue for large benchmarks, we propose a simplification procedure to reduce the complexity of the unrolled netlist without compromising the correctness of the recovered keys. The performance of TimingSAT is validated with extensive experimental results. We summarize our contributions as follows:

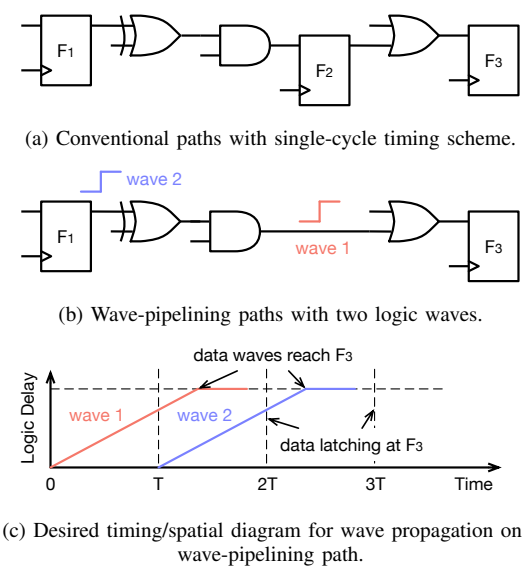
- We propose a novel TU design and a transformation procedure to enable SAT attack on the timing-based camouflaging approaches.
- We formally prove the functional equivalence between the netlist recovered by TimingSAT and the original netlist.
- We propose a simplification procedure to significantly enhance the efficiency of TimingSAT without impacting the functionality of the recovered netlist.

The rest of the paper is organized as follows. Section II provides an overview of existing camouflaging techniques with an emphasis on the timing-based camouflaging approaches. Section III describes the proposed TU design and uses a motivating example to illustrate how it can help to determine the path timing schemes. Section IV describes our TimingSAT algorithm, including the transformation and simplification techniques. Section V validates the performance of TimingSAT and investigates the runtime dependency of TimingSAT on different impacting factors. We conclude the paper in Section VI.

II. Preliminary

IC camouflaging targets at protecting hardware IP against reverse engineering. Different fabrication-level techniques are first proposed with a focus on hiding important circuit structures. For example, in [8], a doping-based technique is developed to create always-on and always-off transistors by changing the doping polarity for selected transistors. A dummy contact-based method is also proposed to obfuscate metal-to-metal or metal-to-diffusion connections. The fabrication-level techniques are then used to build different types of camouflaging cells that look the same to the attackers but implement different functions [4], [5].

Since camouflaging cells usually incur overheads on area, timing, or power, how to use these cells to synthesize the design to maximize the security under the overhead constraints becomes an important question that are intensively studied [4]–[7]. Different ad hoc approaches are first proposed [4] to maximize the output error probability when the functionalities of the camouflaging cells are mis-interpreted. However, due to a lack of formal security evaluation, these methods usually over-estimate the achieved security and can be decamouflaged easily by the SAT attacks proposed in [17], [18]. To enhance the resilience against SAT attack, [5] and



(a) Conventional paths with single-cycle timing scheme.

(b) Wave-pipelining paths with two logic waves.

(c) Desired timing/spatial diagram for wave propagation on a wave-pipelining path.

Fig. 1: Conversion from single-cycle paths to wave-pipelining paths and the required timing constraints.

[6] leverage the AND-tree structure to develop camouflaging schemes that can achieve an exponential increase of the attack complexity with a linear increase of the overhead. Nonetheless, these methods suffer from a fundamental trade-off, i.e., the output error probability reduces exponentially with the increase of the security level. Such weakness is exploited by approximate attacks which can quickly find a good approximation of the original circuit functionality that operates correctly with a very high probability [10], [11]. Although there are new approaches proposed against approximate attacks, which combine different camouflaging strategies or introduce combinational loops [5], [7], [19], the provided security is still implausible [13].

To enhance the security against reverse engineering, recently, timing-based parametric camouflaging approaches are proposed [15], [16]. Besides regular camouflaging cells, [15] also inserts gates with tunable delay to the circuits. The delay configurations of these gates are carefully chosen to satisfy the pre-defined timing constraints and force the attackers to enumerate all the timing paths. Though the idea is promising, [15] suffers from one important drawback. Given the camouflaged netlist, the attackers can resolve the hold time constraints by assigning each gate a larger delay. Then, by operating the circuit with a lower frequency, the setup time constraints can be satisfied as well. In this way, the attackers can either operate the circuit correctly at a lower frequency or conduct resynthesis to the resolved circuit to further optimize its performance.

The drawback is avoided in [16], which obfuscates the circuit timing by introducing multi-cycle paths. As shown in Figure 1(a), conventionally, all the paths in a combinational block operate within a single clock cycle. [16] deliberately removes some flip flops, e.g., F_2 in Figure 1(a), to convert

single-cycle paths into wave-pipelining paths. On a wave-pipelining path, e.g., the combinational path from F_1 to F_3 in Figure 1(b), there are more than one data waves propagating without a flip flop separating them. To retain the same functionalities as the original single-cycle circuit, the second wave cannot catch the first wave at any time during propagation as shown in Figure 1(c). Hence, the following timing constraints must be satisfied for all the wave-pipelining paths with two logic waves:

$$T_{clk} + t_h \leq d_p \leq 2T_{clk} - t_{su}, \tag{1}$$

where d_p is the delay of the wave-pipelining paths and T_{clk} represents the clock period time. t_h and t_{su} denote the hold and setup time for a flip flop.

By carefully removing the flip flops and sizing the gates, the functionality of the design remains unchanged with a mixture of single-cycle and wave-pipelining paths created. Due to the existence of wave-pipelining paths, the attackers cannot recover the correct circuit functionality by simply changing the clock period time. Therefore, they are forced to determine the timing scheme for each path to decide whether it is a single-cycle path or a wave-pipelining path.

To determine the path timing, physical reverse engineering alone is insufficient as it is usually difficult to measure the gate and interconnect delay accurately [16]. Assume the delay extraction techniques incur an inaccuracy factor τ ($0 < \tau < 1$). For a path with delay d_p, if d_p satisfies

$$(1-\tau)d_p \leq T_{clk} \leq (1+\tau)d_p, \tag{2}$$

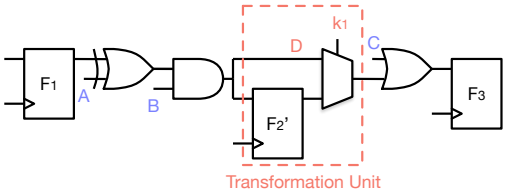
then, the attackers cannot decide whether the path is wave-pipelining or not by physical reverse engineering. Considering that the number of such paths can be enormous even for benchmarks of moderate size, and that it is usually expensive to determine the timing scheme for each path, the timing-based camouflaging strategy becomes very promising.

III. A Motivating Example

Before we formally describe our decamouflaging framework, in this section, we introduce our TU design and use a motivating example to illustrate how the TUs can be used to recover the path timing schemes.

We follow the most widely used attack model [4]–[7] and assume the attackers to have access to the camouflaged netlist and a functional circuit. The camouflaged netlist is acquired from physical reverse engineering. The timing for each path in the camouflaged netlist cannot be exactly determined with an inaccuracy factor τ. The functional circuit is obtained from the open market with its operating clock period, i.e., T_{clk}, known by the attackers. The attackers can query the functional circuit as a black box to determine the correct outputs for the selected input vectors. The goal of the attackers is to determine the path timing based on the input-output pairs and recover the correct circuit functionality.

Now we use the example in Figure 1 to introduce our TU design and illustrate our attack process. Consider the



(a) Circuit transformation based on the proposed TU design.

Cycle	F1	A	B	C	D	F2'	F3
0	1	0	1	x	1	x	x
T	1	0	0	0	0	1	?

Observation: at $2T_{clk}$, if ? = 1, k_1=1, otherwise, k_1=0

(b) Input queries to determine whether a path is single-cycle or wave-pipelining.

Fig. 2: An illustrative example to transform the camouflaged netlist with the proposed TU design and resolve the correct functionality based on input queries.

combinational path from F_1 to F_3 in the camouflaged netlist in Figure 1(b). Assume the path delay satisfies Eq. (2). To determine whether the path is a single-cycle path or a wave-pipelining path, we propose a novel TU design and insert it into the netlist as shown in Figure 2(a). Our TU design mainly consists of a flip flop and a multiplexer, which is controlled by a one-bit key. We denote the netlist after the insertion of TU as the transformed netlist. In the transformed netlist, we regard all the paths, including the combinational path from F_1 to F_3, the path from F_1 to F_2', the path from F_2' to F_3, etc, as single-cycle paths. Then, if the key bit is 0, the functionality of the transformed netlist is equivalent to that of the camouflaged netlist when the original path is a single-cycle path. Otherwise, the functionality of the transformed netlist is equivalent to that of the camouflaged netlist when the original path is a wave-pipelining path. Therefore, to determine whether the original path is single-cycle or wave-pipelining, it suffices to determine the key bit in the TU.

To determine the key bit, we need to query the black-box circuit. Since we assume full access to the scan chain, the logic values of F_1 and F_3 can be controlled and observed. For F_2', as it does not exist in the real circuit, we cannot control/observe its logic value. Consider the input queries shown in Figure 2(b). At 0th cycle, the inputs A, B, C, and F_1 are set to 0, 1, x, 1, respectively, where x represents don't care. Hence, F_2' latches 1 at the clock rising edge at time T_{clk}. Then, we change the logic value for A, B, C, and F_1 to be 0, 0, 0, and 1, respectively. Now, the logic value of D becomes 0, which is different from that of F_2'. Depending on the logic value latched by F_3 at time $2T_{clk}$, we can determine the key bit. More specifically, if F_3 latches 0, then, the key bit equals to 0; otherwise, the key bit equals to 1. Thereby, we can determine whether the path is wave-pipelining or not.

From the example described above, we see that it is indeed possible to determine whether a path is wave-pipelining or not

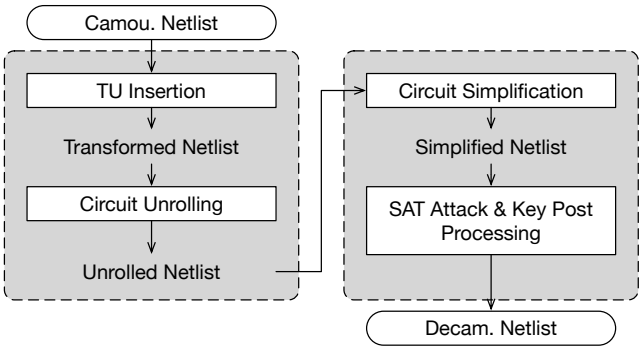


Fig. 3: Flow chart of TimingSAT.

Algorithm 1 TU Insertion

1: $V_t \leftarrow \texttt{TopologicalSort}(C)$;
2: **for** $i = 1$ to $|V_t|$ **do**
3: $\quad u_i.\,\text{dtopi} \leftarrow d_g + \min\{u_j.\,\text{dtopi}\,|u_j \in u_i.\,\text{fanin}\}$;
4: **end for**
5: **for** $i = |V_t|$ to 1 **do**
6: $\quad u_i.\,\text{dtopo} \leftarrow d_g + \min\{u_j.\,\text{dtopo}\,|u_j \in u_i.\,\text{fanout}\}$;
7: **end for**
8: **for** $i = 1$ to $|V_t|$ **do**
9: $\quad$ **if** $u_i.\,\text{dtopi} + u_i.\,\text{dtopo} \geq T/(1+\tau)$ **then**
10: $\quad\quad \texttt{InsertTU}(u_i)$;
11: $\quad$ **end if**
12: **end for**

by querying the black-box circuit and observing the outputs. To apply the attack procedure to a general circuit, there are following important questions we need to answer:

- How to insert the TUs to convert the camouflaged netlist to its single-cycle counterpart and guarantee the correct functionality can be achieved when proper keys are inserted;
- How to determine the input queries to prune incorrect keys in the presence of uncontrollable flip flops in the inserted TUs and prove the correctness of the resolved key bits;
- How to enhance the attack efficiency.

In the next section, we will describe our decamouflaging framework with a focus on answering the questions above.

IV. TimingSAT Framework

In this section, we formally describe the proposed TimingSAT framework. As shown in Figure 3, TimingSAT consists of four steps. First, TUs are inserted into the camouflaged netlist to convert all the paths into single-cycle paths. Then, we unroll the transformed netlist and change all the uncontrollable flip flops into the circuit internal nodes. Although the SAT attack can be applied to the unrolled netlist directly, we propose a simplification procedure to reduce the size of the unrolled netlist and then, exploit the SAT attack to prune the incorrect key assignments. Finally, a circuit post-processing step is conducted to reconstruct the decamouflaged netlist.

A. TU Insertion

Let $\mathcal{C}$ represent the black-box functional circuit and C denote the camouflaged netlist extracted from physical reverse engineering. Since the flip flops in C are controllable and observable through scan chain [4]–[7], we regard the inputs and outputs of the flip flops the same as the primary outputs (POs) and primary inputs (PIs), respectively. Let m be the total number of PIs with input space $\mathcal{I} \subseteq \{0,1\}^m$ and n be the total number of POs with output space $\mathcal{O} \subseteq \{0,1\}^n$.

The first step of TimingSAT is to insert the TUs into C. In C, the wave-pipelining paths are generated by removing a subset of flip flops. Let $\mathcal{D}$ denote the subset of nodes in C where flip flops are removed. To convert all the paths into single-cycle paths and guarantee that the correct functionality can be recovered when a proper set of keys are inserted, at least one TU needs to be inserted to each node in $\mathcal{D}$. As $\mathcal{D}$ is unknown during the decamouflaging process, a straightforward strategy is to insert a TU to each circuit node in C. However, such strategy can result in a huge amount of key bits, and thus, a large number of possible functionalities for the camouflaged netlist, which can significantly increase the attack complexity [5].

We propose to leverage the estimated path delay to reduce the number of TUs inserted to C. Recall in C, there are two types of paths, i.e., single-cycle paths and wave-pipelining paths. For a single-cycle path in C, when a flip flop is inserted to the path, its functionality gets changed. This indicates for a TU inserted along a single-cycle path, its controlling key bit must be 0. Hence, all the TUs inserted along the single-cycle paths can be avoided. Considering the inaccuracy factor τ for the estimated path delay and the cycle period T_{clk}, we now have the following lemma

Lemma 1. *Consider a circuit node u in C. If there exists a path p through u with delay d_p that satisfies*

$$(1+\tau)d_p < T_{clk},$$

then, p must be a single-cycle path and no TUs need to be inserted at u.

Based on Lemma 1, for each node in C, to decide whether a TU should be inserted, the key step is to determine the smallest delay for all the paths through the node. This can be finished in linear time with respect to the size of the camouflaged netlist. As shown in Algorithm 1, after the topological sort of the circuit nodes (line 1), we first traverse the circuit in a topological order and keep a record of the minimum delay from the PIs to each node (line 2 – 4). Then, we traverse the circuit in a reverse-topological order and determine the minimum delay from each node to the POs (line 5 – 7). The TUs can then be inserted based on the acquired delay and Lemma 1 (line 8 – 12).

B. Determine Input Query through Unrolling

Let S denote the transformed netlist with l TUs inserted. Then, the number of key bits in S is l with key space $\mathcal{K} \subseteq \{0,1\}^l$. For $k \in \mathcal{K}$, let $S(k)$ denote the completed circuit by applying k to S. All the paths in $S(k)$ are regarded as single-cycle paths. Consider a sequence of input vectors $I = \{i^0, \ldots, i^{p-1}\}$ of length p with $i^j \in \mathcal{I}, \forall 0 \leq j \leq p-1$. Let $\mathcal{C}(I, s^0)$ denote the sequence of outputs produced by applying I to $\mathcal{C}$ starting from an initial state s^0. We use $\mathcal{C}^j(I, s^0)$ to represent the output at jth cycle. Let $S(I, s^0; k)$ denote the sequence of outputs produced by $S(k)$ with the input sequence I and the initial state s^0.

After the first step of TimingSAT, we can guarantee a TU is inserted to each node in $\mathcal{D}$. Therefore, there must exist a key assignment k^* such that $S(k^*)$ has the correct functionality. More formally, we have the following lemma.

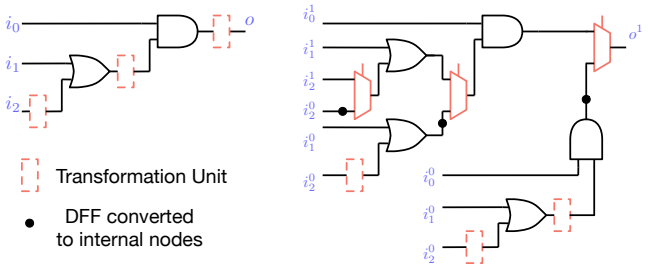
Lemma 2. *$\exists k^* \in \mathcal{K}$ such that $S(I, s^0; k^*) = \mathcal{C}(I, s^0)$, for any input sequence $I = \{i^0, \ldots, i^{p-1}\}$ of length $p > 0$ with $i^j \in \mathcal{I}$, $\forall 0 \leq j \leq p-1$, and initial state s^0.*

To determine k^*, [17], [18] propose the SAT attack to search for the input vectors that can be used to prune incorrect key assignments iteratively. However, these methods are mainly designed for combinational circuits[1]. In S, as the flip flops in the inserted TUs cannot be directly controlled or observed, existing SAT attack strategies cannot be applied directly.

To leverage the SAT attack, we propose to unroll S to create a combinational logic that is equivalent to S up to the unrolled time frame. In the unrolling process, we remove the uncontrollable flip flops in the TUs and connect their output signals to their input signals in the previous cycle. Meanwhile, different from the traditional circuit unrolling scheme [21], we only keep the POs in the last time frame. In Figure 4, we show an example of the transformed netlist and the netlist generated by unrolling the transformed netlist for 1 time frame. As we can see, the unrolled netlist in Figure 4(b) consists of the PIs for 2 time frames, e.g., i_0^0 and i_0^1, and the POs for the last time frame, i.e., o^1.

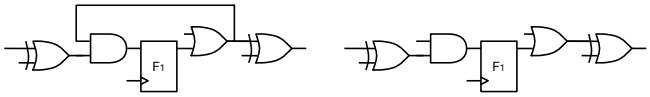
In Figure 4(b), as we unroll the netlist for 1 time frame, not all the flip flops in the inserted TUs are converted into circuit internal nodes. This indicates the outputs of the unrolled circuit still depend on the uncontrollable flip flops. In fact, for general sequential circuits, despite the number of unrolled time frames, the outputs of the generated netlist can always depend on the uncontrollable flip flops. This is because of the existence of the feedback arcs in the fan-in cone of the uncontrollable flip flops. One example is shown in Figure 5(a). If F_1 is an uncontrollable flip flop, then, due to the feedback arc, the outputs of the unrolled netlist always depend on F_1.

[1]Although SAT attack for sequential circuits has been proposed recently [20], it requires unbounded model checking to formally guarantee the correctness of the resolved netlist, which can be intractable.



(a) Transformed netlist. (b) Unrolled netlist.

Fig. 4: Example of circuit unrolling: (a) the transformed netlist; (b) the unrolled netlist generated by unrolling the transformed netlist for 1 time frame (i_0^1 represents the first input signal in the second time frame).



(a) With feedback arcs. (b) Without feedback arcs.

Fig. 5: Example of circuits (a) with and (b) without feedback arcs. For (a), unrolling is insufficient to convert the flip flops into internal nodes.

However, in the transformed netlist S, there are no such feedback arcs. The reasons come from two folds. On one hand, after we regard the outputs and inputs of controllable flip flops as PIs and POs, there are no feedback arcs in C[2]. On the other hand, the insertion of TUs does not introduce any feedback connections either. Therefore, let d denote the maximum number of TUs along a path in S, we have the following lemma regarding the unrolled netlist.

Lemma 3. *After unrolling S for d time frames, all the uncontrollable flip flops in S can be converted into circuit internal nodes and the outputs of the unrolled netlist become independent of the uncontrollable flip flops.*

Let R denote the netlist generated by unrolling S for d time frames. Then, the inputs of R are composed of the PIs of S for $d+1$ cycles. For example, for the transformed netlist in Figure 4(a), $d = 3$. The netlist generated by unrolling the transformed netlist for 3 cycles is shown in Figure 7(a). As we can see, all the flip flops are converted into circuit internal nodes.

As the functionality of the circuit is preserved during unrolling, for any input sequence $I = \{i^0, \ldots, i^d\}$ of length $d+1$ with $i^j \in \mathcal{I}, \forall 0 \leq j \leq d$, we have

$$S^d(I, s^0; k) = R(I; k). \tag{3}$$

As the output of R is independent of initial state s^0, Eq. (3) holds regardless of the initial state s^0. Hence, we rewrite Eq. (3) as $S^d(I; k) = R(I; k)$.

[2]We assume there are no combinational loops in the original netlist, which holds for most of the circuits

After the circuit unrolling, as all the paths in R operates within a single clock cycle and the outputs of R are completely determined by the controllable inputs, we can now leverage the SAT attack proposed in [18] to search for $k^+ \in \mathcal{K}$ such that for any input sequence $I = \{i^0, \ldots, i^d\}$ of length $d+1$ with $i^j \in \mathcal{I}, \forall 0 \leq j \leq d$,

$$R(I; k^+) = R(I; k^*). \tag{4}$$

Regarding k^+, we have the following theorem.

Theorem 1. *$\forall k^+ \in \mathcal{K}$ that satisfies Eq. (4), given an initial state s^0 and input sequence $\{i^0, \ldots, i^p\}$ of length $p+1$, where $p \geq d$, with $i^j \in \mathcal{I}, \forall 0 \leq j \leq p$, we have*

$$S^j(i^0, \ldots, i^p, s^0; k^+) = \mathcal{C}^j(i^0, \ldots, i^p, s^0), \forall j \geq d.$$

Proof. For any $j \geq d$, based on Eq. (3), Eq. (4), and Lemma 2, we have

$$\begin{aligned} S^j(i^0, \ldots, i^p, s^0; k^+) &= S^j(i^{j-d}, \ldots, i^j; k^+) \\ &= R(i^{j-d}, \ldots, i^j; k^+) \\ &= R(i^{j-d}, \ldots, i^j; k^*) \\ &= S^j(i^{j-d}, \ldots, i^j; k^*) \\ &= S^j(i^0, \ldots, i^p, s^0; k^*) \\ &= \mathcal{C}^j(i^0, \ldots, i^p, s^0). \end{aligned}$$

Hence proved. □

Theorem 1 indicates that starting from a random initial state, after feeding the decamouflaged netlist with $d+1$ input vectors, we can always guarantee the outputs generated by the downstream execution of the circuit to be correct. Hence, we formally prove the correctness of the resolved circuit functionality.

C. Netlist Simplification for Decamouflaging Acceleration

Based on the unrolling scheme proposed above, we are able to leverage the existing SAT attack and find k^+. However, as the size of the unrolled netlist increases in proportional to d, the performance of the SAT attack can be significantly degraded for a circuit with large d. From our empirical study, which will be shown in the experimental results, for small benchmarks, when the number of unrolling cycles increases from 2 to 10, the runtime of the SAT attack on average increases by $45\times$. For large benchmark circuits, as shown in Figure 6, d can be as large as 82. Simply unrolling the circuit for 82 cycles will result in a much larger netlist compared to the original netlist, which makes the SAT attack prohibitively expensive and unpractical.

To enhance the efficiency of TimingSAT, we hope to reduce the number of unrolling time frames. According to [16], in order to satisfy the delay constraints in Eq. (1), for each wave-pipelining path generated by removing a flip flop, the flip flops at the beginning and the end of the path must remain unchanged. Therefore, to recover the original circuit, for each path in C, at most one flip flop needs to be inserted. Let $\mathcal{P}_R$ represent the set of paths in R and $\mathcal{K}_p$ represent the set of key bits along a path $p \in \mathcal{P}_R$, then, we have the following lemma.

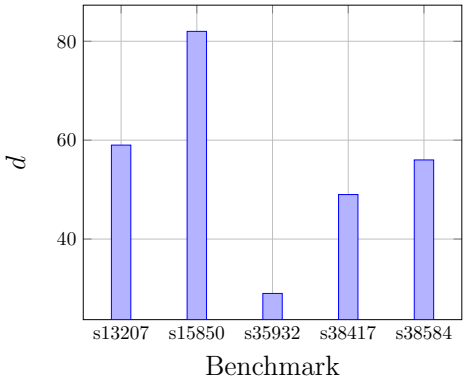
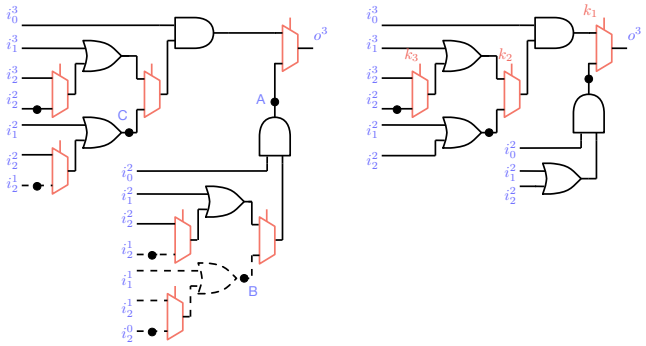


Fig. 6: The maximum number of TUs inserted along a path for large benchmark circuits.



(a) Unrolled netlist. (b) Simplified netlist.

Fig. 7: Example of (a) the unrolled netlist with only controllable PIs as circuit inputs for the camouflaged netlist in Figure 4(a) and (b) the simplified netlist generated by removing the redundant paths.

Lemma 4. $\sum_{j \in \mathcal{K}_p} k_j^* \leq 1, \forall p \in \mathcal{P}_R$.

Based on Lemma 4, we can identify the redundant paths in R and try to simplify R. Again consider the camouflaged netlist in Figure 4(a) and the unrolled netlist in Figure 7(a). In Figure 7(a), the paths marked with dashed lines are redundant as they all have more than one non-zero key bits along the paths. Therefore, we can remove the redundant paths and get the simplified netlist as shown in Figure 7(b).

The simplification procedure can be accomplished in linear time with respect to the circuit size by traversing the unrolled circuit R in a reverse topological order. Starting from the POs, for each node converted from a flip flop in the unrolling process, we check whether there are any other nodes converted from flip flops in its fan-out cone. If there are no such nodes in the fan-out cone, this node is kept; otherwise, we simply remove the node as well as the circuits in its fan-in cone. For example in Figure 7(a), node A, B, and C are converted from flip flops in the unrolling process. For B, because there is A

Algorithm 2 Post processing to determine the key values

1: $V_t \leftarrow$ `TopologicalSort`(R);
2: **for** $i = |V_t|$ to 1 **do**
3: **if** u_i.gate = TU and $k_i^+ = 1$ **then**
4: `SetKeyInFanin`(u_i, R)
5: **end if**
6: **end for**

in its fan-out cone, we know it must be redundant and thus, we remove it and its fan-in cone. For C, because there is no such nodes in its fan-out cone, we keep it in the simplified netlist.

After the simplification procedure, the inputs of the simplified netlist are composed of PIs of S for just 2 time frames, which remains constant for different benchmarks and is only determined by the camouflaging strategy. Meanwhile, the simplified circuit preserves all the properties of R. By applying the SAT attack to the simplified circuit, the correct key k^+ can be resolved.

D. Key Post-processing

After resolving k^+, we carry out post processing to the resolved key bits to construct the decamouflaged netlist. The importance of the post-processing stage can be illustrated with the example in Figure 7. Consider the simplified netlist in Figure 7(b). If $k_1^+ = 1$, then, k_2^+ and k_3^+ become don't cares and do not impact the functionality of the simplified circuit. To satisfy Lemma 4, we need set all these don't-care key bits to 0.

The pseudo code to determine the key values is shown in Algorithm 2. In R, given k^+, we first do topological sorting of the circuit nodes (line 1). Then, we traverse the circuit in a reverse topological order (line 2 – 6). For each TU, if the corresponding key is 1 (line 3), all the key bits in its fan-in cone must be don't cares. Hence, we traverse its fan-in cone and set all the key bits within the cone to 0. The process is continued until the key values of all the TUs are determined.

After the key values are determined, we insert the keys back to the transformed netlist. For each TUs inserted in the transformed netlist, if its control key bit is 0, we just remove the TU and connect the input of the TU directly to its output. Otherwise, we remove the TU and insert a flip flop to the circuit node. Thereby, we can construct a sequential circuit which operates within a single clock period and has the correct functionality.

E. Discussion

In TimingSAT, we rely on Lemma 4 to simplify the unrolled circuit and enhance the efficiency of TimingSAT significantly. As we have discussed, Lemma 4 holds since the existing camouflaged netlists only consist of wave-pipelining paths with at most two logic waves. However, it is indeed possible to remove multiple flip flops and generate wave-pipelining paths with more than two logic waves while keeping the circuit functionality unchanged [22]. Such changes can be easily accommodated in our current framework. First, from physical reverse engineering, though accurate path delay cannot be determined due to the uncertainty, the upper bound on the path delay can still be estimated to determine the maximum number of logic waves along a path. Then, in the simplification process, for each node converted from a flip flop, in its fan-out cone, we can determine the maximum number of nodes converted from flip flops along a certain path. If the number is larger than the maximum number of logic waves, we know the path must be redundant. Hence, we can remove the node as well as its fan-in cone. One example of the flow is shown in Figure 8. Assume there are at most 3 logic waves for a wave-pipelining paths. Given the camouflaged netlist in Figure 8(a), we first determine whether a TU should be inserted to each node, i.e., A, B, C, D, E. Assume node D and E satisfy Lemma 1, then, we only insert TUs to node A, B and C and get the transformed netlist shown in Figure 8(b). After we unroll and simplify the transformed netlist, we get the simplified netlist in Figure 8(c). Finally, we apply the SAT attack to the simplified netlist and carry out post processing to determine the correct key bits.

V. Experimental Results

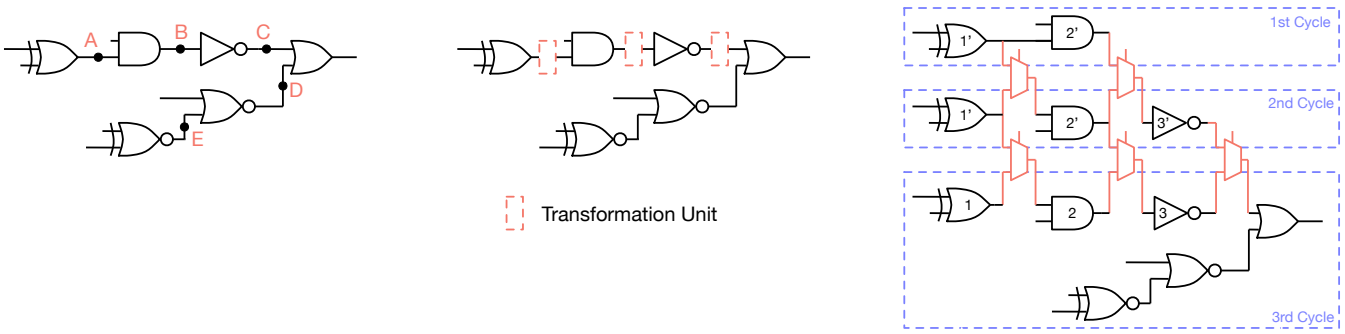
In this section, we report on our extensive experiments to demonstrate the effectiveness of the proposed TimingSAT framework. The transformation and simplification procedure in TimingSAT is implemented in C++ and the SAT attack engine is adopted from [18]. The timing-based camouflaging strategies are implemented following [16] and the functional camouflaging strategies are implemented following [4]. The benchmarks are chosen from ISCAS'89 benchmark suite [23], the detailed statistics of which are shown in Table I. We run all the experiments on an eight-core 3.40 GHz Linux server with 32 GB RAM. We set the runtime limit of TimingSAT to 5×10^4 seconds.

TABLE I: Benchmark statistics of ISCAS'98.

bench	PIs	POs	FFs	Nodes
s953	16	23	29	418
s1196	14	14	18	530
s1238	14	14	18	509
s5378	35	49	179	2779
s9234	36	39	211	5597
s13207	62	152	638	8027
s15850	77	150	534	9786
s35932	35	320	1728	16353
s38417	28	106	1636	22397
s38584	38	304	1426	19407

A. Efficiency of TimingSAT

We first report the efficiency of TimingSAT on the benchmark circuits and demonstrate the importance of the simplification procedure. To generate the camouflaged netlist, 10 flip flops are removed following [16]. We also assume largest uncertainty in the reverse engineering process, i.e., we assume the delay of each path in the camouflaged netlist satisifies Eq. (2) so that a TU is inserted to each circuit node in the transformation process. In Table II, we show



(a) (b) (c)

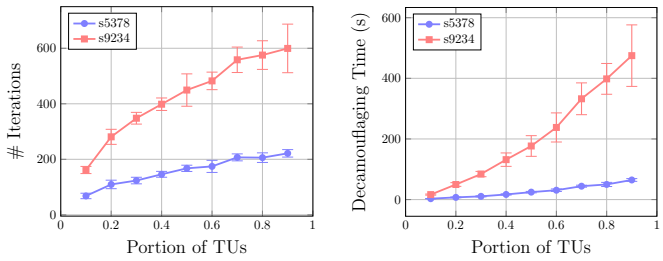
Fig. 8: Example of the transformation flow to enable SAT attack when there exists at most three waves for a wave-pipelining path: (a) camouflaged netlist; (b) insertion of TUs; and (c) simplified unrolled circuit.

the statistics of the unrolled netlists without simplification and the simplified netlists. As we can see, the simplified netlists are significantly smaller compared with the unrolled netlist. Then we conduct SAT attacks on the unrolled netlist and the simplified netlist. As shown in Table II, for the first three small benchmarks, TimingSAT with the simplification procedure achieves on average 87× speedup. Meanwhile, for the large benchmarks, when we directly apply the SAT attacks to the unrolled netlists, the SAT attacks cannot be finished within the pre-defined time limit. However, with the simplification procedure, all the camouflaged netlist can be decamouflaged efficiently within 10^4 seconds, which demonstrates the importance of the simplification procedures and the efficiency of our TimingSAT framework.

B. Runtime Dependency of TimingSAT

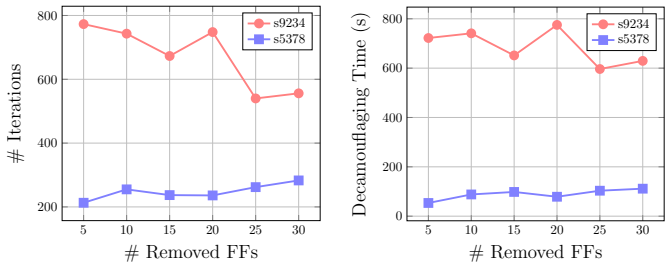
Now, we want to evaluate the runtime dependency of TimingSAT on the number of flip flops removed in the camouflaging process and the delay uncertainty in the reverse engineering process. We use benchmark circuit s5378 and s9234 as examples. To evaluate the impact of delay uncertainty, we fix the number of removed flip flops to 10 in the camouflaging process. Then, we gradually increase the number of TUs inserted in the transformation process and run the decamouflaging attack. For each configuration, we run 10 different experiments and insert the TUs to different subsets of nodes for each experiment. We show the decamouflaging iterations and time in Figure 9. As we can see, with the increase of the number of inserted TUs, a significant increase of the decamouflaging iterations and time can be observed. Specifically, when TUs are inserted to 90% of circuit nodes, the decamouflaging time is almost 30X compared to the case when TUs are inserted to 10% of circuit nodes.

To evaluate the impact of the number of flip flops removed in the camouflaging process, we select the largest uncertainty and insert a TU to each node in the camouflaged netlist. Then, we gradually change the number of removed flip flops and run the decamouflaging attack. We run 10 different experiments and remove different flip flops for each experiments. The



(a) Number of iterations. (b) Decamouflaging time.

Fig. 9: Impact of delay uncertainty on the decamouflaging complexity.



(a) Number of iterations. (b) Decamouflaging time.

Fig. 10: Impact of the number of removed flip flops on the decamouflaging complexity.

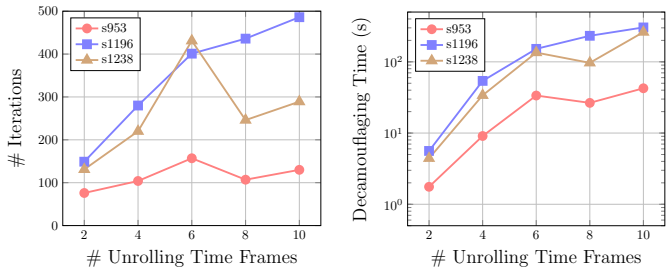
change of decamouflaging iterations and time is shown in Figure 10. As can be observed, while the decamouflaging time for s5378 increases slightly with the increase of the number of removed flip flops, the decamouflaging time for s9234 even reduces. This indicates when the number of unrolling time frames and the number of inserted TUs remain unchanged, the decamouflaging time is not directly dependent on the number of removed flip flops in the camouflaging process.

C. Impact of Unrolling Time Frames

As have described in Section IV-C, Lemma 4 is the key basis that enables the simplification process without impacting the correctness of the resolved keys. Meanwhile, Lemma

TABLE II: Performance of TimingSAT with and without simplification on the benchmark circuits.

bench	TimingSAT w/o simplification						TimingSAT w/ simplification					
	PIs	POs	Nodes	Keys	# Iter	Rt (s)	PIs	POs	Nodes	Keys	# Iter	Rt (s)
s953	5.6×10^2	42	6.6×10^3	1.8×10^2	1.2×10^2	36.5	70	42	1.0×10^3	1.8×10^2	72	1.7
s1196	5.0×10^2	22	1.2×10^4	5.1×10^2	5.4×10^2	8.6×10^2	44	22	1.6×10^3	5.1×10^2	1.0×10^2	4.5
s1238	4.6×10^3	22	1.1×10^4	4.9×10^2	2.7×10^2	2.6×10^2	44	22	1.5×10^3	4.9×10^2	1.2×10^2	5.4
s5378	5.2×10^3	2.0×10^2	7.4×10^4	2.5×10^3	N/A	N/A	3.8×10^2	2.0×10^2	8.4×10^3	2.5×10^3	2.5×10^2	82.1
s9234	1.3×10^4	2.4×10^2	3.1×10^5	5.3×10^3	N/A	N/A	4.7×10^2	2.4×10^2	1.6×10^4	5.3×10^3	1.3×10^3	1.6×10^3
s13207	4.0×10^4	7.8×10^2	4.3×10^5	7.2×10^3	N/A	N/A	1.3×10^3	7.8×10^2	2.3×10^4	7.2×10^3	6.3×10^2	4.7×10^2
s15850	4.8×10^4	6.7×10^2	7.5×10^5	1.0×10^4	N/A	N/A	1.2×10^3	6.7×10^2	2.9×10^4	1.0×10^4	1.1×10^3	1.3×10^3
s35932	4.9×10^4	2.0×10^3	4.2×10^5	1.8×10^4	N/A	N/A	3.5×10^3	2.0×10^3	4.8×10^4	1.8×10^4	1.2×10^2	4.4×10^2
s38417	7.9×10^4	1.7×10^3	1.0×10^6	2.0×10^4	N/A	N/A	3.3×10^3	1.7×10^3	6.7×10^4	2.0×10^4	1.6×10^3	6.0×10^3
s38584	7.9×10^4	1.7×10^3	1.0×10^6	1.7×10^4	N/A	N/A	2.9×10^3	1.7×10^3	5.7×10^4	1.7×10^4	5.5×10^2	2.5×10^3



(a) Number of iterations. (b) Decamouflaging time.

Fig. 11: Relation between decamouflaging complexity with the number of unrolling time frames.

4 holds as the existing camouflaged netlists only consist of wave-pipelining paths with at most two logic waves. Now, we want to evaluate the impact when more than two logic waves exist in the wave-pipelining paths. We select three small benchmark circuits, including s953, s1196, and s1238, and two large benchmark circuits, i.e., s5378 and s13207, as examples and demonstrate the relation between the number of possible logic waves along a wave-pipelining paths and the decamouflaging iterations and time. We still assume the largest uncertainty in the reverse engineering process and remove 10 flip flops following [16]. As shown in Figure 11 and Figure 12, both the decamouflaging time and iterations increase significantly with the number of logic waves. Specifically, for the three small benchmarks, increasing the number of logic waves from 2 to 10 can on average increase the decamouflaging time by 45.8X. For the two large benchmarks, when the number of logic waves becomes larger than 4, TimingSAT cannot finish within the pre-defined time limit. The experimental results indicate that increasing the number of logic waves for the wave-pipelining paths can indeed enhance the security against SAT attacks.

However, since we assume the largest uncertainty here, the result is rather conservative. We further investigate the change of decamouflaging iterations and time when both the number of unrolling time frames and the number of inserted TUs are changed. The experimental results are shown in Figure 12. As we can see, when we insert TUs to 50% or 20% of circuit nodes, significant reduction of decamouflaging time can be observed. Even when the number of unrolling time frames becomes as large as 10, TimingSAT can still be finished within the pre-defined time limit.

D. Decamouflaging of Combination of Timing-based and Traditional Strategy

Now, we evaluate the situation when the timing-based camouflaging strategy is combined with high-entropy functional camouflaging strategies. We leverage the fault analysis based camouflaging strategy proposed in [24] and combine it with the timing-based strategy. We use benchmark s5378 and s9234 as examples. The number of removed flip flops is set to 10 and the largest delay uncertainty is assumed. We gradually change the number of camouflaging gates inserted into the circuit and compare the decamouflaging time of the combined strategy with the camouflaging strategy in [24]. We run 10 experiments for each configuration and insert the camouflaging cells into different circuit nodes. The experimental results are shown in Figure 13. As we can see, by combining different camouflaging strategies, much larger decamouflaging time can be achieved compared with leveraging the strategy in [24] alone. However, even after 50 camouflaging gates are inserted following [24], the netlist can still be decamouflaged within 10^3 seconds.

VI. Conclusion

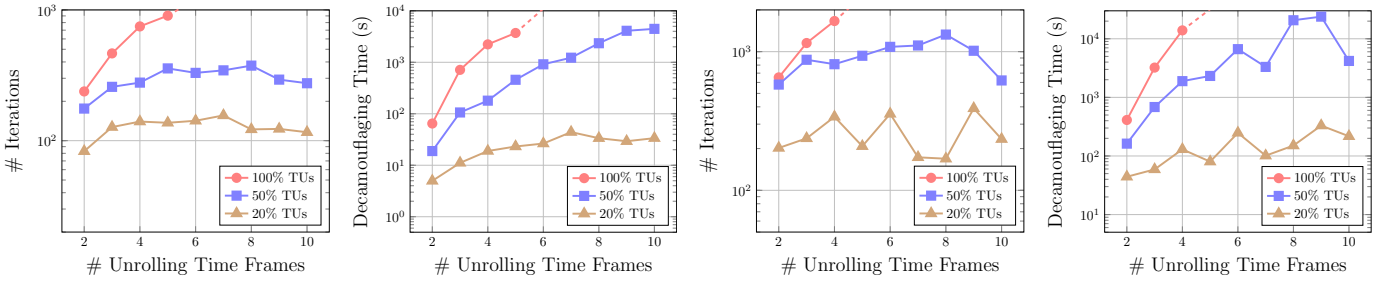
In this paper, we propose TimingSAT, a SAT-based framework for security evaluation of the timing-based parametric camouflaging strategies. TimingSAT leverages a novel transformation strategy to convert all the paths in the camouflaged netlists into single-cycle paths while preserving the correct circuit functionalities. An unrolling procedure is then proposed in TimingSAT to enable SAT attacks. The functional correctness of the resolved netlist is formally proved. To accelerate TimingSAT, we also propose a simplification procedure to significantly reduce the complexity of the transformed netlist. The efficiency and effectiveness of TimingSAT are demonstrated with extensive experimental results.

Acknowledgement

This work was partially supported by DARPA (FA8650-18-1-7822).

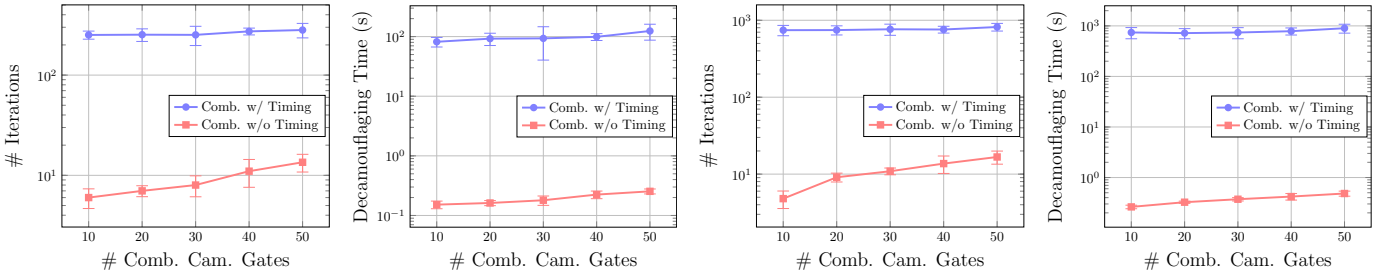
References

[1] P. Subramanyan, N. Tsiskaridze, W. Li, A. Gascon, W. Y. Tan, A. Tiwari, N. Shankar, S. Seshia, and S. Malik, "Reverse engineering digital circuits using structural and functional analyses," *IEEE Trans. on Emerging Topics in Computing*, vol. 2, no. 1, pp. 63–80, 2014.



(a) Number of iterations (s5378) (b) Decamouflaging time (s5378) (c) Number of iterations (s13207) (d) Decamouflaging time (s13207)

Fig. 12: Dependency of the decamouflaging complexity on the number of unrolling time frames and the inserted TUs.



(a) Number of iterations (s5378). (b) Decamouflaging time (s5378). (c) Number of iterations (s9234). (d) Decamouflaging time (s9234).

Fig. 13: Comparison on the decamouflaging complexity between random camouflaging with/without timing camouflaging.

[2] S. E. Quadir, J. Chen, D. Forte, N. Asadizanjani, S. Shahbazmohamadi, L. Wang, J. Chandy, and M. Tehranipoor, "A survey on chip to system reverse engineering," *ACM J. on Emerging Technologies in Computing Systems*, vol. 13, no. 1, pp. 6:1–6:34, 2016.

[3] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," in *Proc. IEEE/ACM Design Automation Conf.*, 2011, pp. 333–338.

[4] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proc. ACM Conf. on Computer & Communications Security*, 2013, pp. 709–720.

[5] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Z. Pan, "Provably secure camouflaging strategy for IC protection," in *Proc. Int. Conf. on Computer Aided Design*, 2016, pp. 28:1–28:8.

[6] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "CamoPerturb: Secure IC camouflaging for minterm protection," in *Proc. Int. Conf. on Computer Aided Design*, 2016, pp. 29:1–29:8.

[7] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic obfuscation for creating sat-unresolvable circuits," in *Proc. IEEE Great Lakes Symp. on VLSI*, 2017.

[8] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware trojans: extended version," *J. of Cryptographic Engineering*, vol. 4, no. 1, pp. 19–31, 2014.

[9] A. Vijayakumar, V. C. Patil, D. E. Holcomb, C. Paar, and S. Kundu, "Physical design obfuscation of hardware: A comprehensive investigation of device and logic-level techniques," *IEEE Trans. on Information Forensics and Security*, vol. 12, no. 1, pp. 64–77, 2017.

[10] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, 2017.

[11] Y. Shen and H. Zhou, "Double DIP: Re-evaluating security of logic encryption algorithms," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM, 2017, pp. 179–184.

[12] C. Yu, X. Zhang, D. Liu, M. Ciesielski, and D. Holcomb, "Incremental SAT-based reverse engineering of camouflaged logic circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2017.

[13] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based attack on cyclic logic encryptions," in *Proc. Int. Conf. on Computer Aided Design*. IEEE, 2017, pp. 49–56.

[14] X. Wang, Q. Zhou, Y. Cai, and G. Qu, "A conflict-free approach for parallelizing sat-based de-camouflaging attacks," in *Proc. Asia and South Pacific Design Automation Conf.* IEEE, 2018.

[15] Y. Xie and A. Srivastava, "Delay Locking: Security enhancement of logic locking against ic counterfeiting and overproduction," in *Proc. IEEE/ACM Design Automation Conf.*, 2017.

[16] L. Zhang, B. Li, B. Yu, D. Z. Pan, and U. Schlichtmann, "TimingCamouflage: Improving circuit security against counterfeiting by unconventional timing," in *Proc. Design, Automation and Test in Europe*, 2018.

[17] M. El Massad, S. Garg, and M. V. Tripunitara, "Integrated circuit (IC) decamouflaging: Reverse engineering camouflaged ICs within minutes." in *Proc. Network and Distributed System Security Symp.*, 2015.

[18] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, 2015, pp. 137–143.

[19] S. Patnaik, M. Ashra, J. Knechtel, and O. Sinanoglu, "Obfuscating the interconnects: Low-cost and resilient full-chip layout camouflaging," in *Proc. Int. Conf. on Computer Aided Design*. IEEE, 2017, pp. 41–48.

[20] M. El Massad, S. Garg, and M. Tripunitara, "Reverse engineering camouflaged sequential circuits without scan access," in *Proc. Int. Conf. on Computer Aided Design*. IEEE, 2017, pp. 33–40.

[21] N. Miskov-Zivanov and D. Marculescu, "Soft error rate analysis for sequential circuits," in *Proc. Design, Automation and Test in Europe*. IEEE, 2007.

[22] L. Zhang, B. Li, M. Hashimoto, and U. Schlichtmann, "Virtualsync: Timing optimization by synchronizing logic waves with sequential and combinational components as delay units," in *Proc. IEEE/ACM Design Automation Conf.* IEEE, 2018.

[23] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. IEEE Int. Symp. on Circuits and Systems*, 1989, pp. 1929–1934.

[24] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proc. ACM Conf. on Computer & Communications Security*, 2013, pp. 709–720.