NiLoPher: Breaking a Modern SAT-Hardened Logic-Locking Scheme via Power Analysis Attack

Prithwish Basu Roy[1,2], Johann Knechtel[1], Akashdeep Saha[3], Saideep Sreekumar[1], Likhitha Mankali[1,2], Mohammed Nabeel[1,2], Debdeep Mukhopadhyay[3], Ramesh Karri[2] and Ozgur Sinanoglu[1]

[1] New York University Abu Dhabi, Abu Dhabi, UAE, {pb2718,johann,sds710,lm4434,mtn2,os22}@nyu.edu
[2] NYU Tandon School of Engineering, New York, USA, rkarri@nyu.edu
[3] Indian Institute of Technology Kharagpur, Kharagpur, West Bengal, India, {akashdeepsaha95,dmcseiitkgp}@gmail.com

Abstract. LoPher [SSC+20] brings, for the first time, cryptographic security promises to the field of logic locking in a bid to break the game of cat-and-mouse seen in logic locking. Toward this end, LoPher embeds the circuitry to lock within multiple rounds of a block cipher, by carefully configuring all the S-Boxes. To realize general Boolean functionalities and to support varying interconnect topologies, LoPher also introduces additional layers of MUXes between S-Boxes and the permutation operations. The authors of LoPher claim resilience against SAT-based attacks in particular.
Here, we show the first successful attack on LoPher. First, we uncover a significant limitation for LoPher's key-space configuration, resulting in large numbers of equivalent keys and, thus, a largely simplified search space for attackers in practice. Second, motivated by their well-proven working against ciphers, we employ a power side-channel attack against LoPher. We find that ISCAS-85 benchmarks locked with LoPher can all be broken in few thousands of traces. Finally, we also outline a simple and low-cost countermeasure to render LoPher more secure.

Keywords: Hardware security · IP piracy · Logic locking · Side-channel attack

1 Introduction

The increasing complexity of Integrated Circuits (ICs) and the rising expenses associated with establishing and maintaining semiconductor foundries have led to the global expansion and outsourcing of IC design and manufacturing processes. To streamline design efforts and meet tight time-to-market deadlines, design houses often acquire Intellectual Property (IP) cores from third-party vendors. Most design houses also opt for a fabless model, i.e., are outsourcing fabrication to offshore foundries offering the latest technology nodes.

In such a globalized IC supply chain, however, the risk of untrusted entities gaining access to valuable IP or physical ICs introduces security threats. These malicious actors may engage in activities such as IP piracy, illegal overproduction of ICs, tampering with ICs by inserting Hardware Trojans (HTs), or reverse engineering the netlist for other unauthorized means of use [KRRT10]. Furthermore, any sensitive data processing is threatened by attacks on the IC under operation, e.g., side-channel attacks, fault-injection attacks, data readout, et cetera [ZF05].

To protect against various threats, logic locking has gained much attention in recent years [SL22]. Locking means to obfuscate the IC by incorporating additional key-based logic into the original design. However, there is a continuous game of cat-and-mouse in the field of logic locking, with ever-advanced locking schemes versus ever-advanced attacks; see Section 2.3 for a detailed discussion. Independently, cryptographic ciphers are well-established for means of protecting sensitive data. While providing clear mathematical formulations and related security promises, hardware implementations of ciphers are known to be vulnerable to various attacks also outlined above. See Section 2.1 and 2.2 for more details on ciphers and related attacks.

LoPher [SSC+20] brings, for the first time, cryptographic security promises to the field of logic locking. LoPher is embedding the circuitry to lock within multiple rounds of a block cipher, by carefully configuring all the S-Boxes. To realize general Boolean functionalities and to support varying interconnect topologies, LoPher introduces layers of MUXes between S-Boxes and the permutation operations.

Here, we show the first successful attack on LoPher. We also outline simple and low-cost countermeasures to render LoPher more secure. The contributions of this work can be summarized as follows:

- A critical review of the implementation structure of LoPher, providing novel insights on a notable reduction in the key search space in practice.
- The first successful attack on LoPher, based on the well-known and effective principle of correlation power analysis.
- Proposals for different structural and functional countermeasures to make LoPher more resilient, with little to no additional overheads.

2 Background and Motivation

2.1 SPN-Based Block Ciphers

Block Ciphers: Block ciphers are a prominent and well-proven mechanism to protect sensitive data by means of cryptography.

Generally, block cipher algorithms involve the repetition of block-wise operations known as a *round transformation* [FS03]. These round transformations within block ciphers are created by combining two cryptographic sub-systems sharing the same plaintext and ciphertext space. Another way to define a block cipher is as a set of Boolean transformations operating on nb-bit vectors, referred to as blocks [Hey01]. This process converts plaintext blocks of a fixed length nb into ciphertext blocks of the same length under the influence of a cipher key k. The Boolean transformation in block ciphers typically consists of three steps: substitution, diffusion, and key mixing, which are all key-dependent. The security of such systems relies on the repetitive application of such round transformations.

During encryption, the input message is segmented into plaintext blocks, each matching the block size of the cipher. For example, in the Data Encryption Standard (DES), which precedes the Advanced Encryption Standard (AES), the block size is 64 bits. In the basic *Rijndael* ciphers, there are three variable block sizes: 128, 192, and 256 bits. In the case of AES, the block size is fixed at 128 bits. This iterative transformation and segmentation of the input contribute to the security of these block cipher systems [Pat00].

Substitution-Permutation Networks (SPNs): These networks are widely used architectures in the construction of block ciphers. The SPN architecture is characterized by alternating layers of *substitution* operations, through so-called S-Boxes, and *permutation* operations. More specifically, the substitution operations involve replacing blocks of input bits with different output bits using S-Boxes. The latter introduce non-linearities into the cipher operation, enhancing its resistance against linear and differential cryptanalysis. The permutation operations, on the other hand, are to re-arrange or shuffle the bits, to achieve diffusion. The alternation of substitution and permutation operations ensures that a change in one input bit propagates across the entire block cipher, providing a high degree of diffusion in the ciphertext.

In short, the SPN architecture enables two important features/properties for the block ciphers, namely confusion and diffusion, making the ciphers resistant to various cryptanalysis techniques [HFPM18].

The security of SPN-based ciphers relies on the properties of the S-Boxes, the number of rounds of joint substitution and permutation layers, and the key size. As computing capabilities advance, researchers have to further explore and design SPN-based ciphers with increased security and efficiency properties, ensuring their relevance in the ever-evolving landscape of secure data processing. As indicated, the substitution and permutation layers collectively contribute to achieving confusion and diffusion, making the cipher resistant to various attacks. The essence of confusion and diffusion is that the internal data processing is difficult to control in general and practically impossible to predict without knowing the secret key. The latter is important to remember for the motivation of this work, i.e., a first-of-its kind attack on LoPher, a logic locking scheme based on SPN ciphers.

PRESENT: This is an ultra-lightweight SPN-based block cipher. Crafted for compactness and efficiency in hardware design, PRESENT operates by default on 31 rounds, on 64-bit blocks, and accommodates keys of either 80 or 128 bits [BKL+07]. Its design caters to scenarios demanding low-power consumption and optimal chip efficiency, positioning it as a noteworthy choice for resource-constrained environments [BKL+07].

PRESENT operates as follows. In general, plaintexts pass through 31 rounds until the final ciphertexts are obtained. Each of the 31 rounds process the intermediate texts using a varying *round key* K_i which is used for a linear bitwise permutation and a non-linear substitution layer. The latter uses 16 S-Boxes of size 4 bits. More specifically, to obtain K_i, the register holding the user-provided master key, K is rotated by 61 bit positions to the left, the left-most four bits are passed through the current S-Box, and the round index is XORed with specific bits of K with the least significant bit of the round counter on the right. Then, the actual substitution and permutation operations follow. See [BKL+07] for more technical details.

2.2 Side-Channel Attacks

Side-channel attacks can infer sensitive information by observing and analyzing physical channels established by ICs' hardware during operation. That is, these hardware attacks exploit information leaked through physical implementations of cryptographic systems, such as power consumption or timing variations [ZF05]. These kind of information leakages occur due to the impact the environment has on the physical implementation of the system and the workings of the circuitry, due to the microarchitectural implementation of the IC, et cetera [ZF05, RD20].

A classical example of a side-channel attack is a power side-channel (PSC) attack on SPN-based block ciphers. This attack involves observing and analyzing variations in power consumption that occur during cryptographic operations. That is, the secret key of a SPN-based block cipher can be inferred by this analysis of power consumption, thus breaking the cipher's security promise [BCO04, SW12]. Different PSC attacks have been demonstrated, like correlation power analysis (CPA) [BCO04], differential power analysis (DPA) [MS16], or machine learning-based techniques [PHJ+17]. Furthermore, there are more generic, analytical approaches like test vector leakage assessment (TVLA) [SM15]. Note that PSC attacks apply to ciphers in general, not only to SPN-based block ciphers.

CPA: We focus on the correlation power analysis attack in this work, as this CPA approach is proven as quite effective against SPN-based block ciphers [BDG16]. During CPA, the attacker carefully observes and records the power consumption variations of the cryptographic device while it processes known plaintexts or ciphertexts. These so-called power traces are then correlated with the intermediate values or key-dependent operations, such as those involving the S-Boxes. The attacker aligns the power traces with the corresponding intermediate values, often obtained through the inverse S-Box operation. This alignment helps in identifying patterns or correlations between the power consumption and the internal states, which then helps to infer the secret keys in turn.

More specifically, statistical techniques like Pearson Correlation Coefficient (PCC) are applied to quantify the relationship between the observed power traces and all the intermediate values that could possibly arise for all possible key combinations. The goal is to find correlation peaks that indicate a high likelihood for the correct key for some specific intermediate value. The peaks indicate points where the power consumption correlates strongly with key-dependent operations, thereby revealing the correct key that is used under the hoods in the IC in operation.

Countermeasures such as masking techniques are often suggested to make PSC attacks more challenging. However, such techniques often involve trade-offs between power, performance, and area of the IC hardware versus and the improved security [GMOP15, KGS+11, GM11].

2.3 Logic Locking

Logic locking has emerged as powerful defense mechanism against different threats in the IC supply chain [SL22]. The concept involves obfuscating the design by incorporating additional key-based logic into the original circuit. The secret key, essential for the locked circuit to function as intended, is securely stored on-chip in a tamper-proof memory.

SAT-Based Attacks: The advent of Boolean satisfiability (SAT)-based attacks has significantly impacted logic locking [SRM15]. This type of attack operates under an oracle-guided threat model, necessitating a locked netlist and an activated chip, also known as *oracle*. The attack involves transforming the locked netlist into conjunctive normal form (CNF) clauses and using a SAT solver to eliminate incorrect keys, namely by identifying distinguishing input patterns (DIPs). A DIP is a pattern that produces different outputs for different keys; for those patterns, the oracle then assists in identifying the key that confirms with the oracle's golden behaviour. This process iterates until all incorrect keys are eliminated, ultimately extracting the overall correct key.

Advanced Locking Schemes, Advanced Attacks: The advent of SAT-based attacks has prompted a shift in focus within the community towards developing so-called *provably secure* logic locking techniques. For example, *SAR-Lock* [YMRS16] and *Anti-SAT* [XS19, YMSR17a] offer a straightforward solution by controlling the key elimination capability of SAT-based attacks, forcing them to iterate one by one through the entire key space to find the correct key. A trivial observation in the aforementioned SAT-resilient locking techniques is that identifying and correcting the sole incorrect input-output pair for any given incorrect key renders the netlist functional. This insight is leveraged in creating the so-called *bypass attack* [XSTF17], which incorporates additional logic to rectify or bypass the lone incorrect input-output pair in such low-output-corruption locking schemes. The overhead associated with the bypass circuitry scales linearly with the number of incorrect input-output patterns generated. Another attack, utilizing structural traces and known as the *removal attack* [YMSR17b] has also gained prominence. This attack scrutinizes the locked netlist to identify the locations where the locking circuitry has been augmented with the original design. The removal attack obtains a functional netlist by eliminating this augmentation and introducing a binary constant at that location.

SFLL-HD [YTS19], a generalization of *SFLL* [YSN+17] and *TTLock* [YMRS17], is another provably secure logic locking technique. Here, a portion of the original circuit undergoes modification through a so-called "Functionality Stripper" component, subsequently being restored to its intended functionality with a "Functionality Restore Unit." Furthermore, SFLL-HD employs a comparator to generate more incorrect input-output patterns, thereby challenging the practicality of bypass attacks in terms of overheads. Note that the removal attack fails on SFLL-HD as it produces a non-functional netlist that is unable to recover the "Functionality Restore Unit." However, the comparator structure which SFLL-HD utilizes leaves behind some structural traces that can be exploited by the *structural and functional analysis attack on logic locking (FALL)* [SS19]. Following this, *CAS-Lock* [SXTF20] has been proposed to mitigate the shortcomings of SFLL-HD. Still, several novel attacks, including both oracle-guided and oracle-less threat models, have been proposed to nullify CAS-Lock's defense [SLS21].

Another genre for defending against SAT-based attacks in logic locking involves so-called SAT-hard structures into the circuit such that each iteration of the SAT solver requires exponential time to solve. In *FullLock* [KAHS19], a key-configurable logarithmic-based network (CLN) is implemented to obscure routes or a selected group of wires, which function as SAT-hard structures. Despite these efforts, machine learning-based, especially graph neural network-based attacks like [APK⁺21] have proven effective in circumventing such locking techniques.

2.4 Motivation

The perpetual cat-and-mouse game in logic locking emphasizes the continuous need to develop techniques with solid mathematical security guarantees; otherwise, the community will likely nullify them over time. The latter is the very motivation for LoPher [SSC+20], a modern, SAT-hard locking technique that proposes to utilize cryptographic SPN-based block ciphers to embed a circuit. In other words, the cipher fabric is utilized to realize logic locking. By inheriting the mathematical security of SPN-based block ciphers, LoPher aims to provide robust logic locking.

There are no known attacks on LoPher.[1] However, considering the well-known success of PSC attacks against ciphers, in this work, we are the first to thoroughly study the resilience (or rather lack thereof) of LoPher against such PSC attacks.

2.5 LoPher: A Modern, SAT-Hardened Scheme for Logic Locking Based on Circuit Embedding into Block Ciphers

As indicated, the main idea of LoPher is to utilize the security offered by SPN-based block ciphers to securely embed any design of choice into the cipher fabric, so that such embedding is obfuscated through the cipher's keys, i.e., the design of choice would only be operating as intended with the correct key being applied for the cipher. One could also think of the cipher fabric as some special case of programmable gate array, with the cipher keys serving to realize the programming of some circuitry of choice (i.e., the design to protect via means of LoPher logic locking) into the fabric.

By default and without loss of generality, LoPher utilizes PRESENT [SSC+20] as cipher of choice.

S-Boxes as Logic Gates: As outlined in Sec. 2.1, S-Boxes play a pivotal role in SPN-based block ciphers by introducing non-linearity. Now, LoPher suggests implementing logic gates through S-Boxes. Given that each output bit of an S-Box results is a non-linear mapping of its input bits, such mappings can be configured to achieve any desired Boolean function, i.e., any logic-gate mapping, namely by fixing specific input bits to binary constants.

For example, consider the input bits i_1, i_2, i_3, i_4 to a PRESENT S-Box. The mapping represented by the second output bit o_2 of the PRESENT S-Box can be expressed as:

$$o_2 = i_1 i_2 i_4 \oplus i_1 i_3 i_4 \oplus i_1 i_3 \oplus i_1 i_4 \oplus i_1 \oplus i_2 \oplus i_3 i_4 \oplus 1 \quad (1)$$

[1]A recent but orthogonal attack work has shown that an idea opposite idea to LoPher, namely logic locking of regular ciphers, can be easily broken by fault attacks [UGP24].

Table 1: Configuration examples for an S-Box acting as router or switch, i.e., to delegate selected inputs to specific outputs. Constant '0'/'1' for inputs values represent the specific K_s [i].K_c [i] assignments as needed. X denotes do-not-care assignments.

Input Bit Positions				**Output Bit Positions**			
i_1	i_2	i_3	i_4	o_1	o_2	o_3	o_4
I1	0	0	1	**I1**	X	X	X
1	**I2**	0	1	**I2**	X	**I2**	**I2**
0	1	**I3**	1	**I3**	**I3**	X	**I3**
1	1	0	**I4**	**I4**	**I4**	**I4**	X
I1	1	0	1	X	**I4**	**I4**	**I4**

By setting i_1 and i_2 to '0', the above equation simplifies to $o_2 = i_3 i_4 \oplus 1$ which is the NAND operation between i_3 and i_4.

Permutation Operations and S-Boxes for Routing: LoPher furthermore suggests leveraging bit-permutation layers of SPN-based block ciphers to establish interconnections between logic gates. Traditionally, these layers link S-Boxes from one cipher round to the next. Given that S-Boxes are configured as logic gates in LoPher, these bit-permutation layers facilitate the gates' connections based on the desired topology. This is achieved by configuring specific S-Boxes as buffer gates, which is explained next.

It is important to note that such buffer S-Boxes are crucial to maintain the general cipher structure/fabric while being able to support the embedding of various designs of choice. More specifically, these buffer S-Boxes help to route signals across cipher rounds without engaging in any computational processes. Depending on the circuit to embed, multiple rounds of buffer S-Boxes may have to be employed to route the output of some specific S-Box acting as logic gate to the input of another S-Box/gate in subsequent rounds – see Figure 1 for an example. Table 1 shows further examples for how an S-Box receiving values to its four input pins can be configured to route out a single value through an output pin of choice.

As indicated, embedding multi-level circuits with varying fan-ins and fan-outs necessitates connecting the configured S-Boxes within different cipher rounds. Recall that, in an SPN-based block cipher, the permutation layer determines which S-Box is linked to which S-Boxes in the next round. For instance, in one round of the PRESENT cipher, there are 16 S-Boxes, with the ith S-Box in round r connecting to the $q, q+4, q+8$, and $q+16$ S-Boxes of the succeeding round $r+1$, where $q = \lfloor (i/4) \rfloor$, respectively. Thus, LoPher utilizes S-Boxes as needed for routing functionality. That is, buffer S-Boxes are also used to replicate some signals to multiple output bits, thereby realizing multiple fan-outs signals. Figure 1 shows examples for this feature as well.

Additional MUX Layers for Configuration: As indicated, to configure S-Boxes as logic gates, buffers or routers, specific input bits of the S-Boxes must be fixed to specific binary values. To achieve this, LoPher introduces key-based 2 : 1 MUX layers between the S-Boxes and the permutation layers – see Figure 2. More details for this MUX layer are also described next.

LoPher Implementation: In the default configuration of LoPher, the PRESENT cipher works on an 80-bit key and a 64-bit input state. For each round, a 64-bit key is generated by a key scheduler. Note that, when implementing a logic gate or a router in LoPher, it is necessary to specify which input bits to the S-Box will represent the inputs to the gate or the router being implemented. The remaining input bits need to be configured accordingly. Since there is no direct way of configuring an S-Box in the middle of two rounds, the vanilla PRESENT cipher is modified for LoPher (Figure 2): after the S-Box

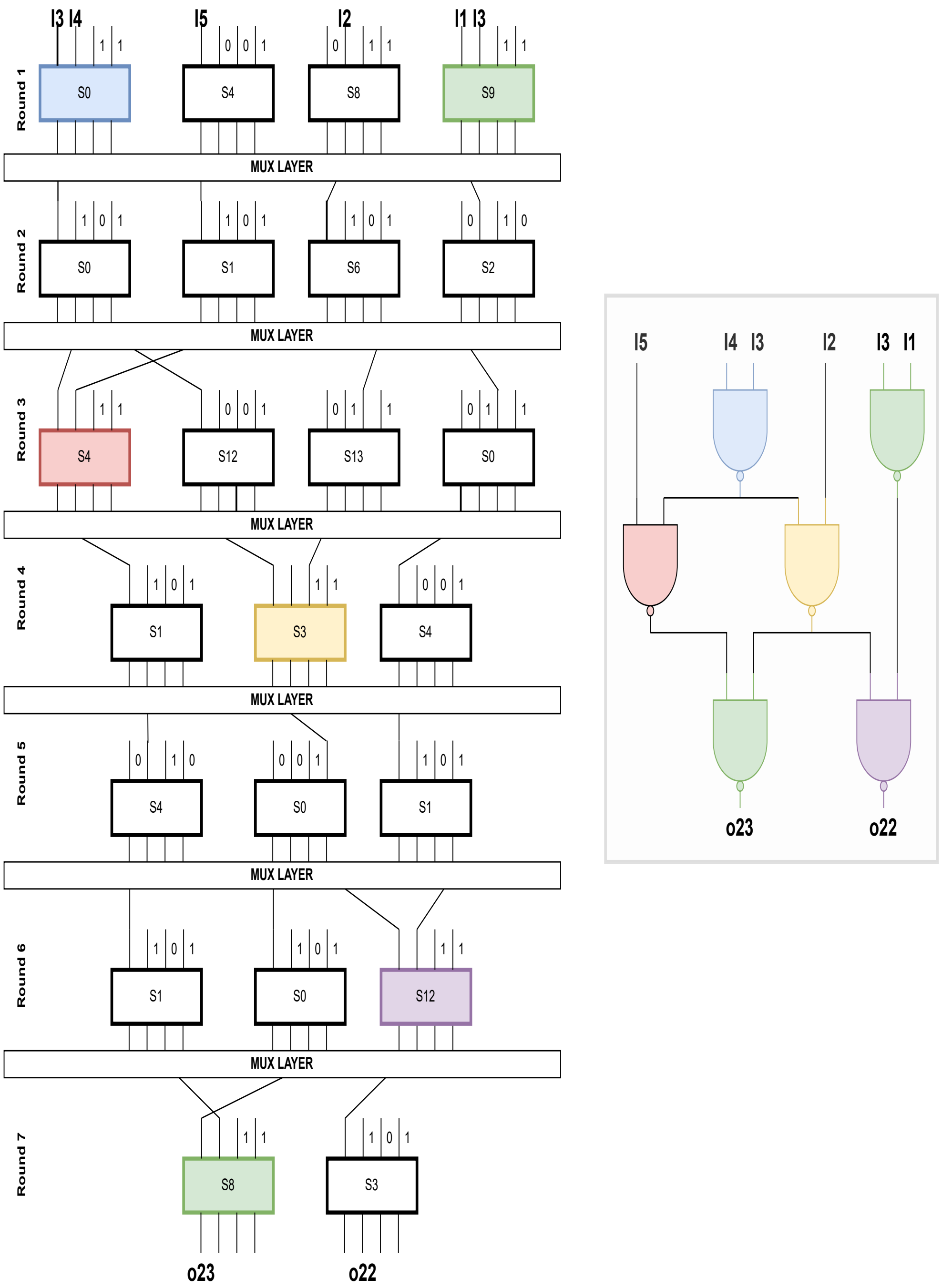


Figure 1: Embedding a simple circuit *c17* of logic level 3 into 7 rounds of LoPher. Gates in the original circuit and the corresponding S-Boxes have the same color. White S-Boxes act as routers that pass selected inputs to selected outputs, based on their configuration keys K_c.

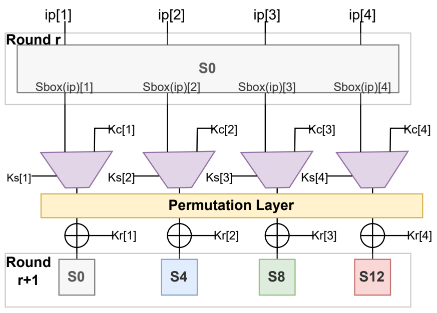


Figure 2: LoPher round structure. The outputs of S-Box $S0$ of round r is fed into 4 MUXes. The other inputs of each of the MUXes are a configuration bit K_c, which determines how the next S-Boxes should act, and a select bit K_s, respectively, which selects whether the input from the previous S-Box or the configuration bit should be passed on.

layer, a MUX layer is introduced. Note that the round-key layer of the cipher also remains present in LoPher, thereby introducing another set of keys K_r. All these layers and keys are described more formally in the following.

The MUX layer comprises, for each of the S-Box, four 2:1 MUXes. Each MUX takes two bits as input. One bit is the output bit from the S-Box, while the other bit is a configuration bit, represented as K_c. The select line of the MUX is another bit, K_s. To determine whether the next S-Box will act as a logic gate or as router, we need to configure K_c and K_s accordingly. Equation 2 represents the round equation for one output bit $op[i]$ of an S-Box j, while Equation 3 represents the round equation for one output bit with the MUX layer added to it. Here, K_r[i] is the round key bit corresponding to the output bit of the S-Box, respectively. So, in LoPher, each round has a 64-bit round key K_r, a 64-bit configuration key K_c, and a 64-bit MUX select key K_s.

$$op[i] = SBox_j(ip)[i] \oplus K_r[i] \tag{2}$$

$$op[i] = (SBox_j(ip)[i].\overline{K_s[i]} + K_s[i].K_c[i]) \oplus K_r[i] \tag{3}$$

Full Example: Figure 1 shows an example for a small circuit, namely the ISCAS-85 benchmark *c17*, embedded into the LoPher fabric L. Here, *c17* has five primary inputs and two primary outputs and has a depth of three logic levels.

Starting from the first level of *c17*, from the first NAND gate on the left, we configure the S-Box S0, i.e., the first of all 16 in the PRESENT round structure, such that its first and the second input pins are connected to the *c17*'s primary inputs I3 and I4.

Note that, albeit not shown in figure, there is another MUX layer just above the S-Boxes in the first round that helps in configuring the S-Boxes via the respective K_c and K_s keys. More specifically, for S0, $K_s = 0110$ and $K_c = 0011$, respectively. This makes S-Box S0 act as an NAND gate for inputs I3 and I4; refer to Table 2 for this configuration. The output of the NAND gate will be available from the first output pin of S0. Note that all four outputs of S0 are routed within the permutation layer to the next round, namely to S-Boxes S0, S4, S8, and S12, respectively. This is done to maintain a generic structure that by itself is not revealing hints to attackers, also following the generic structure of PRESENT. However, since we are only concerned about the data of the first output, we will set K_s for the corresponding MUX between rounds 1 and 2 to '0'. Next, we embed the second NAND gate of level 1 into the S-Box S9, in a similar way as we did for the first NAND gate. It is important to note that the selection of which S-Box to pick for each gate, as well as the configuration of subsequent S-Boxes, all depends on the overall structure of *c17*. For example, the first NAND gate has a fan-out of two, but when we configure

Table 2: Configuration for an S-Box as logic gates, as shown in LoPher [SSC+20]. Constant '0'/'1' for inputs represent the specific K_s [i].K_c [i] assignments needed to realize the logic. *op* refers to the output of the logic.

Gate	S-Box Input Bits				S-Box Output Bits			
	i_1	i_2	i_3	i_4	o_1	o_2	o_3	o_4
XOR	A	0	B	1			*op*	
	A	1	B	1		*op*		*op*
	1	A	B	0		*op*		
	A	B	0	0				*op*
	A	B	1	0	*op*			
AND	0	1	A	B		*op*		
	A	1	0	B			*op*	
OR	1	0	A	B	*op*			
	A	0	1	B	*op*			
NOR	1	A	1	B		*op*		
	A	1	0	B	*op*			
NAND	A	B	1	1	*op*			
	0	0	A	B		*op*		
NOT	0	A	1	0		*op*		
	A	0	0	0	*op*			

an S-Box to act as NAND gate, we have only one output available (from the first S-Box output pin). To deal with this scenario, we need to introduce one "buffer round" where we will take the input from S0 of round 1 and duplicate the output into two signals to be used in the actual round 2 (round 3 in the figure, as the buffer round becomes round 2).

To implement this buffer round, S-Box S0 in round 2, which is connected to S0's first output, has to act as a router that receives an input from its first input and relays it to both its second and fourth outputs. This can be achieved by configuring $K_s = 0111$ and $K_c = 0101$, respectively. This will route the outputs of S0 to the S-Boxes S0 and S4 of round 3 (originally round 2); refer Table 1 for routing configuration. Since round 2 is a buffer round now, we also have to ensure that the output from S9 of round 1 passes through this round unchanged. Thus, we configure S2 of round 2 in a way that passes the second input bit value to the first output bit, which is connected again to S0 of round 3. S-boxes S4 of round 3 also have to receive an input of I5, this input is sent to it by configuring S-Box S1 of round 2 as a router with $K_c = 0101$ and $K_s = 0111$ respectively. S1 of round 2 is connected to S4 of round 1, which has the configuration of $K_c = 0001$ and $K_s = 0111$ respectively. For implementing the second gate from left in the second level of *c17*, we had to find a common successor of S0 of round 1 and S8 of round 1 (which is the entry point of I2) with minimum depth. It can be observed that S0 reaches S3 of round 4 via S12 of round 2. S8 of round 1 also is connected to S3 of round 4 via S6, and S13 of round 2 and 3 respectively. Similarly, we find common successors of the already embedded gates in LoPher to embed the remaining gates from *c17*. The remaining two gates are embedded in the S8 and S3 of round 7, from where the outputs($o23, o22$) are collected from the first and second output pins of S8 and S3. The S-Boxes that were not used in the embedding process were assigned the $K_c = 0000$ and $K_s = 0000$, respectively.

3 Our Observations for LoPher: Reduced Key Space

Claims in LoPher: According to the LoPher authors, the correct output of the embedded circuit can only be obtained for a unique set of round keys, with the latter being a unique combination of a 64-bit K_c value, a 64-bit K_s value, and a 64-bit K_r value. Note that the key K_r can be derived from a PRESENT 80-bit master key. Accordingly, the LoPher authors claim that breaking LoPher would require to break $(r \times 128) + 80$ bits for r rounds of LoPher, which is computationally intractable for classical computing technology.

Next, we will prove otherwise—we show how the LoPher key space is smaller in practice.

Table 3: Truth table of one S-Box input and output in LoPher along with its MUX.

S-Box *ip*	K_s	K_c	S-Box *op*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

It is important to note that these observations and the actual attack, presented in Section 4, are orthogonal and separate contributions of this work.

Dependence of K_c on K_s: As explained, LoPher comprises a modified version of PRESENT cipher by adding MUX layers. These layer determine whether the following S-Box should act as a logic gate or router (i.e., reroute some values to other S-Boxes in the next round but without computation). The latter feature can be turned off by setting all K_s bits to '1's: this would make LoPher pass all 4 bits of an S-Box to the next round as is and maintain the basic operation of the PRESENT cipher, including its properties of diffusion and confusion. Now also recall how Equation 3 differs from Equation 2, considering the additional component of $K_c[i].K_s[i]$ describing the MUX layer and routing feature of LoPher. When $K_s[i]$ is '0', LoPher sends the output of S-Box $S0$ to the next S-Box, say $S0'$, which is the S-Box S0 of the next round. When $K_s[i]$ is '1', however, the configuration bit $K_c[i]$'s value gets forwarded.

In Table 3, we show the truth table for the Equation 3. It can be seen here that, when $K_s[i] = 0$, the value of $K_c[i]$ does not influence the output. This means that, when $K_s[i] = 0$, $K_c[i]$ can take either '0' or '1', but the S-Box output *op* will not be impacted.

Multiple Equivalent Keys: Recall that PRESENT uses 16 S-Boxes per round, each with 4-bit inputs and 4-bit outputs. For LoPher, these bits are passed on to four 2:1 MUXes with four configuration bits ($K_{c3} - -K_{c0}$). Also recall that the selection of a particular S-Box output bit or configuration bit is controlled by the corresponding K_s bit.

Table 4 shows nibbles of an S-Box output, labelled as (O_3–O_0), along with different values for K_c and K_s nibbles. Now, the input bit at position I_i is assigned X (i.e., do-not-care) when Ks_i is set to '1', whereas K_{ci} is assigned X when K_{si} is set to '0'. These do-not-care assignments will not affect the output of the MUX.

For security analysis, since I_i can vary, we are mainly concerned with the key K_c, which should be kept secret and fixed for a given round of LoPher. When K_s is set to all '1's, only one possible K_c value will give the correct output. However, if K_s is set to all '0's, then all sixteen possible K_c nibble values will give the correct output, which is actually totally independent of K_c. It can be inferred that the number of possible K_c nibble values for a given K_s nibble is determined by the number of bits set to '1' in the K_s nibble. The important security implication of that insight is the following. For a fixed 64-bit K_r and a 64-bit K_s key, we need only be concerned about the positions where the bit is set to '1', i.e., to pass on the configuration bits. For all the other remaining positions that are set to '0', the K_c bits in those positions can take either '1' or '0' without impacting the outputs and behaviour of the LoPher circuitry, i.e., they are do-not-care.

In short, *if K_s has k bits set to '1', then only 2^k possible combinations of K_c bits are to be considered by an attacker, not all 2^{64}.*

Table 4: For a given K_s nibble, there can be multiple equivalent K_c nibbles that provide the same output for the round. The values at bit position marked as X is do-not-care, i.e., does not get transferred as such to the output.

$K_{s3}K_{s2}K_{s1}K_{s0}$	$I_3I_2I_1I_0$	$K_{c3}K_{c2}K_{c1}K_{c0}$	# Equiv. K_c	$O_3O_2O_1O_0$
0000	$I_3I_2I_1I_0$	$XXXX$	16	$I_3I_2I_1I_0$
1111	$XXXX$	$K_{c3}K_{c2}K_{c1}K_{c0}$	1	$K_{c3}K_{c2}K_{c1}K_{c0}$
0101	I_3XI_1X	$XK_{c2}XK_{c0}$	4	$I_3K_{c2}I_1K_{c0}$
1010	XI_2XI_0	$K_{c3}XK_{c1}X$	4	$K_{c3}I_2K_{c1}I_0$
0001	$I_3I_2I_1X$	$XXXK_{c0}$	8	$I_3I_2I_1K_{c0}$
1000	$XI_2I_1I_0$	$K_{c3}XXX$	8	$K_{c3}I_2I_1I_0$
0111	I_3XXX	$XK_{c2}K_{c1}K_{c0}$	2	$I_3K_{c2}K_{c1}K_{c0}$
0011	I_3I_2XX	$XXK_{c1}K_{c0}$	4	$I_3I_2K_{c1}K_{c0}$

4 Power Analysis Attack on LoPher

In this work, for the first time in the literature, we show the vulnerability of LoPher to traditional PSC attacks. Such attacks are promising since, at its heart, LoPher is a customized implementation of PRESENT, which itself is known to be vulnerable to such attacks [LBC18, BDG16].

Attack Assumptions: We follow some traditional assumptions for PSC attacks: the attacker has access to an *oracle*, i.e., a chip that secretly holds the correct key and can be monitored for its power consumption and its functional behaviour. Further, we follow a chosen-plaintexts attack model where we also observe the ciphertexts. This is valid because the LoPher circuitry embeds/obfuscates some circuitry that is unknown to the attacker, but the primary inputs and outputs remain accessible as such, since the circuitry obfuscated by LoPher is still to be used as is.

Recall that, unlike a vanilla PRESENT, aside from the round keys K_r, each round of LoPher involves an additional MUX layer with a 64-bit configuration key K_c and another 64-bit selection key K_s. We assume that the values for all keys are fixed in the oracle. We also assume that the attacker can separately identify each round of LoPher from the captured power traces.

Attack Implementation: Without loss of generality, we utilize the correlation power analysis (CPA) attack in this work. Unlike for PSC attacks on a regular PRESENT implementation, here we need to attack each round separately, to infer the 128-bit K_c and K_s values which are independent for each round. The specific CPA workflow to attack LoPher is outlined in Algorithm 1, and important aspects are also outlined next.

We devise a Python implementation of LoPher that mimics the IC oracle. We implement a pipelined version of the LoPher hardware for that purpose; this is a practical assumption for IC design in general. Besides, to emulate real-world power traces for that IC oracle, we implement a Hamming distance (HD)-based power model that can be subjected to variable Gaussian noise. In other words, given some inputs to the LoPher IC oracle, we can observe real-world power traces that cover the toggling activities of sensitive flip-flops via HD-based and noisy power traces. Note that, in another instance without Gaussian noise turned on, this Python implementation also serves as model for generating all hypothetical power values for all possible key candidates. This instance is typically also referred to the actual power model for PSC attacks.

For the main body of the CPA, on the high level, we follow a classical approach: we attack each S-Box separately, while exploring all related, possible key combinations for K_c and K_s. However, we also have to follow certain customized steps as required to attack LoPher. For an important example, we have to attack each round separately, since there is no dependency of K_c and K_s keys across the different rounds in LoPher. Few more details

Algorithm 1: CPA on LoPher

Input: **M**: Model of the LoPher circuitry under attack: **O**: Oracle, i.e., actual LoPher circuitry under attack; number of S-Boxes per round; R: number of rounds used in **M**; N: number of input patterns; I: list of N input patterns; m: number of input bits for each S-Box; HD_O: list of noisy HD power-model values, obtained from actual *oracle*, for N input patterns.
Output: A list of recovered equivalent keys Kc_{rec}, Ks_{rec} for all R rounds.

```
1  $r = 0$ /* Index of current round under attack; reset */
2  $Kc_{rec}[r] = Ks_{rec}[r] = \phi$ /* Set for all recovered equivalent keys for each round $r$; reset */
3  while $r < R$ do
4      $s = 0$ /* Index of current S-Box under attack; reset */
5      while $s < S$ do
6          $i = 0$ /* Data value of current $Ks_{guess}$ nibble for current S-Box $s$; reset */
7          $L = \phi$ /* Pearson correlation values for Kc, Ks pairs for current S-Box $s$; reset */
8          while $i < 2^m$ do
9              $Ks_{guess}[s] = i * 0x1000000000000000 >> 4 * s$ /* Shift bits by 4*s to obtain bit-level $Ks_{guess}$ for current S-Box $s$ */
10             $j = 0$ /* Data value of current $Kc_{guess}$ nibble for current S-Box $s$; reset */
11             while $j < 2^m$ do
12                 $Kc_{guess}[s] = j * 0x1000000000000000 >> 4 * s$ /* Shift bits by 4*s to obtain bit-level $Kc_{guess}$ for current S-Box $s$ */
13                 $HD_M = calculateHammingDistance(\mathbf{M}, r, I, Kc_{guess}[s], Ks_{guess}[s])$ /* Calculate HD power model for current S-Box $s$, considering the current pair $Kc_{guess}$, $Ks_{guess}$ */
14                 $L = L \cup PCC(HD_M, HD_O)$ /* Calculate and store Pearson correlation for calculated HD power model versus HD power observed from oracle */
15                 $j = j + 1$ /* Increment for next candidate of $Kc_{guess}$ */
16             $i = i + 1$ /* Increment for next candidates of $Ks_{guess}$ */
17         $Kc_{pred}[s], Ks_{pred}[s] = getKcKsWithMaxCorrelation(L)$ /* Kc, Ks pairs with max correlation value are tracked for S-Box $s$ */
18         $s = s + 1$ /* Increment for next S-Box */
19     $s = 0$ /* New round; reset S-Box index */
20     while $s < S$ do
           /* Store recovered keys $Kc_{rec}, Ks_{rec}$ for all S-Boxes for current round */
21         $Kc_{rec}[r] = Kc_{rec}[r] \cup allCombination(Kc_{pred}[s])$
           $Ks_{rec}[r] = Ks_{rec}[r] \cup allCombination(Ks_{pred}[s])$
22     $r = r + 1$ /* Increment r for next round */
23 return $Kc_{rec}, Ks_{rec}$ /* Final list of all guessed keys for all rounds */
```

are also outlined next.

For each S-Box in each round, we keep track of the key candidates with the highest PCC value. If there are multiple (K_c and K_s) pairs that give the same highest correlation value, all have to be stored as possible key values in Kc_{rec} and Ks_{rec}, respectively. Then, all possible combinations of key values have to be explored across all rounds, to finally obtain some recovered keys. It is important to keep the latter in mind for the discussion of the complexity faced by the attackers; recall Section 3 and also see Section 6. Finally, all possible recovered keys are verified against the functional behaviour of the oracle, and the ones that comply are considered as correct keys.

5 Experimental Investigation

5.1 Setup

Code: All the codes required for this study, including plot generation, were devised in Python. The implementation of the LoPher-tailored CPA framework, which is the main asset for this study, is outlined in Section 4.

We will release all codes post peer-review.

Computation Platform: All codes were implemented and executed on a desktop workstation with Intel Xeon Processor W-2245 (8C, 3.9GHz 4.7GHz Turbo HT 16.5MB) and with 128GB RAM.

Dataset: Experiments are conducted on *ISCAS-85* benchmarks. Selected benchmarks locked using LoPher have been provided to us as courtesy by the LoPher authors.

5.2 Detailed Case Study for *c17*

In this section, we describe in detail the proposed, custom CPA-based attack on LoPher for the case of the *c17* benchmark. We will also cover the role of noise on the success of the attack, as well as the dependence of the PCC on the number of power traces.

***c17* in LoPher:** Recall the embedding of *c17* into LoPher from Section 2.5. The first round has two S-Boxes, S0 and S9, that acted as the two NAND gates residing in the first logic level of *c17*. Then there were two other S-Boxes, S4 and S8, that acted as routers that send the primary inputs $I5$ and $I2$ to the subsequent rounds. For all the S-Boxes that were not used, neither as logic gate nor as switch, default K_c and K_s values of '0' were assigned. Further recall that we have a software implementation of the LoPher hardware, called the IC oracle, which is used to obtain noisy power traces for the LoPher-locked design operating within the PRESENT cipher fabric (Section 4). We also have another simplified instance of the LoPher hardware that models (without noise) the toggling activities for all possible key combinations while attacking individual S-Boxes in each round of LoPher. The latter is also called the hypothetical power model for PSC attacks.

Attack Operation and General Observations: First, we create a set of input patterns of size N. This set will be provided to the hypothetical power model as well as to the IC oracle. As for any CPA, the power traces from the IC oracle are correlated against all the values obtained from the hypothetical power model. However, for attacking LoPher, we have to tackle each round separately, unlike for other classical ciphers. Again, this is because the K_c, K_s keys for LoPher are independent for each round.

More details for the attack on *c17* are discussed next. For the first S-Box S0, we consider the K_c, K_s value as a pair of two 4-bit keys. Then we calculate the HD for that S-Box for each of the N input patterns and the S-Box output, considering all possible key assignments. Once this list of HD values is obtained from the hypothetical power model, the PCC is calculated against the noisy power traces obtained from the IC oracle. For each (K_c, K_s) combination considered for the S-Box S0, the value of the (K_s, K_c) pair for which the maximum value of PCC is observed is recorded. In Figure 3a, we show the PCC values for all possible (K_c, K_s) guesses for an input set of size $N = 1500$. It can be observed that the (K_s, K_c) pair of (3, 3) has the highest correlation – and that indeed is the actual (K_s, K_c) value for the S-Box S0 for *c17*. Similarly, we tackle all other S-Boxes in that first round of LoPher. For the S-Box S9, which embeds the second NAND gate of the first logic level of *c17*, we successfully retrieve the correct (K_s, K_c) pair as (3, 3) again. For the other two relevant S-Boxes, S4 and S8, that were acting as a router for the primary inputs $I5$ and $I2$, respectively, the highest PCC value revealed the correct (K_s, K_c) pairs to be (7, 1) (Figure 3c) and (11, 3) (Figure 3d). Thus, have we retrieved all the (K_s, K_c) pairs of the configured S-Boxes successfully.

Next, we tried to retrieve the (K_s, K_c) pair for the S-Box S1, which is an unused S-Box. It is important to note that, in the real world, attackers would not be able to differentiate unused S-Boxes from others; hence, we cannot and do not rely on such information, but only want to emphasize the different role of S-Boxes here. Now, the actual (K_s, K_c) value for the S-Box S1 is (0, 0). While computing the PCC values against the noisy power traces from the IC oracle, we observed something interesting: there were 16 different (K_s, K_c) pairs, all of whom had shown the same, highest PCC values. This is visualized in

Figure 3b, where multiple peaks with the same maximum PCC value can be observed. This behavior is due to the fact, for the S-Box S1, $K_s = 0000$. This means that K_c is independent of K_s here. In other words, the output of the S-Box S1 – and thus also its corresponding HD value that is utilized for the PCC computation – does not depend on the value of K_c. In short, there are multiple (K_s, K_c) pairs or equivalent keys that work correctly, and for all those keys the value of K_c is of do-not-care assignment. Recall that this shortcoming of LoPher was already discussed in Section 3.

Impact of Noise: Power traces obtained in the real world can exhibit varying levels of noise. Recall that, to emulate the effects of such noisy traces, we implemented the capability to consider different Gaussian noise profiles for the IC oracle. More specifically, noise profiles are superimposed onto the HD values observed from each S-Box toggling.

For the following experiments to study the impact of noise, we consider the relevant S-Boxes S0, S4, S8, and S9 of round 1. We also consider the S-Box S1, which is neither used as gate nor as router. We perform the first set of experiments by adding relatively low Gaussian noises, with means of the profiles ranging from 0 to 3 and standard deviations ranging from 0 to 3. Note that the only positive mean values can be thought of as being introduced by additional switching activities from other S-Boxes or from other, non-state-related flip-flops.

We observe that S-Box S8, which acts as router, experiences the largest impact by noise (Figure 4a). This is because $K_s = 1101$ for S8, which means that, apart from $K_c = 0011$, all other combinations of K_c like $K_c = \{7, 11, 15\}$ will exhibit very similar baseline correlation values. Similarly, the (K_c, K_s) combinations for which $K_c \& K_s = 0001$ will also exhibit high PCC values. After addition of noise, any one of these (K_c, K_s) combinations can be easily mis-predicted as an incorrect (K_s, K_c) pair.

In contrast, for the S-Boxes S9 (Figure 3e) and S0, $K_c = 0011, K_s = 0011$. Since there are limited possible choices for which $K_c \& K_s = 0011$, the probability of finding the correct (K_c, K_s) remains high even under noise. For the unused S-Box 1, $K_s = 0000$, which means that the output is practically independent of the K_c assignment. This is manifested as distinct peaks having the same PCC values (7 distinct peaks are shown in Figure. 4c). Even in the presence of noise, these peaks stand out in the traces, thus making it evident that K_s has been set as ‘0,’ thereby leaking crucial information to the attacker. Due to such information leakage, we also refer to an assignment of $K_s = 0000$ as *weak assignment*.

Impact of Noise under Varying Number of Traces: Next, we study how the number of traces impacts the correct inference of the (K_c, K_s) pairs for different S-Boxes. In Figure 5a, we can see that the impact of low noise on the PCC values of various (K_c, K_s) combinations is large when the number of traces is small, i.e., 100 here. For such small numbers of traces, even low noise impacts the PCC values quite significantly, which can easily lead to a mis-prediction of the (K_s, K_c) pair. In Figure 5c, the same noise profile is added, but the number of traces is 2,500 now. We can see that the increased number of traces smooths out the imbalances or variations introduced by the noise significantly. Similar observations can be made for Figure 5b versus Figure 5d, where larger noises are added, namely with means = {5,7,9}, and standard deviations ranging from 0 to 3.

Hence, as expected, we can conclude that larger number of traces are beneficial to distinctly identify the maximum PCC values of different (K_c, K_s) combinations. Throughout our various experiments, we observe that 2,500 traces is a practical choice value for making correct predictions for all types of S-Boxes.

5.3 Further Results on ISCAS-85 Benchmarks

For the *c432* benchmark (160 gates, 36 primary inputs, and 7 primary outputs), a sub-circuit comprising 4 nodes was embedded into LoPher within 3 rounds by following the LoPher algorithm for gate selection and locking. Our attack successfully retrieved all the exact K_s and K_c keys by just considering the top values of (K_s, K_c) pairs for each S-Box.

In other words, the step for exploration of all possible key combinations was trivially solved in one iteration. For *c499* (202 gates, 41 primary inputs, and 32 primary outputs), a sub-circuit with 7 nodes was embedded into LoPher within 4 rounds. For this locked circuit, apart from round 3, all the K_c and K_s assignments were again successfully determined from the most probable (K_s, K_c pairs). For the third round, the actual K_s value was among the top 5 predictions based on PCC. Thus, the exploration of key combinations still succeeded with only few iterations. For *c1908* (880 gates, 33 primary inputs, and 25 primary outputs), a sub-circuit of 7 nodes was embedded within 4 rounds of LoPher. For the fourth round, the guessed K_s value's most likely nibble differs from the actual one. Still, the K_c &K_s computation are the same for the actual and the recovered key, making the latter an equivalent key. The actual K_s value was also found in the top 3 key choices considering the PCC values. For *c6288* (2,406 gates, 32 primary inputs, and 32 primary outputs), 10 rounds of LoPher were utilized to embed 5 circuit nodes. Across all circuits, the impact of noise was highest for this case. More specifically, for noise profiles with mean = 0, standard deviation = 3.0, and for 1,500 input patterns, we see mis-predictions in rounds 5, 6, 7, and 8, i.e., when we just consider the (K_s, K_c) pairs having the highest PCC. Still, once considering the top-5 candidates for each S-Box, we can still infer all the correct key values. For *c17*, recall that this circuit was completely embedded within 7 rounds of LoPher. Our attack recovered exact or equivalent (K_c, K_s) pairs for all rounds except for rounds 4 and 5. For the S-Boxes S0, S2, S3 in round 4, the 1st candidate of (K_s, K_c) were not the same or equivalent to those in the oracle, but the correct values were still found in the top-10 PCC values for each of the S-Boxes.

Overall, our proposed attack was able to recover actual/exact or equivalent keys for all circuits within few iterations of exploring the possible key combinations defined by the top-5/top-10 PCC-based key candidates for each S-Box.

6 Countermeasure: Varying K_s for Unused S-Boxes

In LoPher, there are three configurations for S-Boxes: logic gates, routers, and not in use. When an S-Box is not in use, recall that the ease of detecting the (K_c, K_s) pair depends on the value of K_s. That is, the K_c values for a given K_s are dependent on the number of bits in K_s that are set to '1'. When considered together, K_c and K_s have a key space of 2^8 for each S-Box. Further recall that, when all the K_s bits are set to '0', any K_c value will work, which significantly reduces the (K_c, K_s) key space, namely from 2^8 to 2^4.

To further understand this limitation of LoPher, we performed an analysis where we varied the number of unused S-Boxes in a given round of LoPher. For the best case, we assumed that one out of the sixteen S-Boxes, whereas in the worst case, only four S-Boxes remain unused (i.e., twelve S-Boxes are used). We simulated various scenarios where the different K_s nibbles of the unused S-Boxes are configured as follows:

1. Each K_s nibble is configured the same, with $k \in \{0, 1, 2, 3, 4\}$ bits set to '1' (Figure 6a).
2. Each K_s nibbles have random numbers of bits set to '1' (Figure 6b).
3. Some fractions of the S-Boxes have all K_s bits set to '1', while the remaining S-Boxes have random K_s values assigned (Figure 6c,6d).
4. Some arrangements of S-Boxes which all have equal random probability for having their K_s values with four, three, or two bits set to '1' (Figure 6e,6f).

For Scenario 1, we observed that, as the number of bits set to '1' decreases, the key space increases exponentially for a given number of unused S-Boxes. The best scenario, for the defender, occurs when all the S-Boxes are assigned the value of $0xF$, which eliminates all equivalent (K_c, K_s) key pairs. For the cases where K_s had three, two, one, and zero bits set to '1', the number of equivalent (K_c, K_s) key pairs doubled from 2^{10} to 2^{40}. For Scenario 2, we saw that, in the worst-case configuration, where all the 14 K_s values are randomly assigned, the equivalent key space went up to 2^{40}, while in the best-case configuration, it was around 2^{10}. For Scenario 3, we studied two specific sub-cases. In the first case (Figure 6c), we kept 25% of the K_s nibbles set as $0xF$, while the remaining ones were randomly assigned. In the second case (Figure 6d), 50% of the K_s nibbles were set to $0xF$, while the remaining values were assigned again randomly. For the first case, we observed a possible worst-case scenario of 2^{30} equivalent (K_c, K_s) pairs, while for the second case, the worst-case scenario was limited to 2^{20} equivalent key pairs. Finally, for Scenario 4, it can be seen that a uniform arrangement of four and three bits set to '1' (Figure 6e) performs better – as in the numbers of equivalent keys falls in the range of 2^7 – when compared to arrangements with four, three, and two bits equi-probably set to '1' – the number of equivalent keys are in the range of 2^{14} here.

From all the above observations, it can be said that, for an effective countermeasure that avoids redundant key assignments, all the unused S-Boxes should have their K_s values either a) all set to all '1's or b) configured for three and four bits having equi-probably been set to '1'. Note that the choice a), while the best-case scenario in principle, it is easy to explore by attackers as well. Hence, the choice b) is the more suitable. Other choices, especially weak ones with all bits set to '0' or only a single bit set to '1' should be avoided.

It is important to note that this outlined countermeasure incurs zero overheads/cost, as it only requires to re-configure the already existing LoPher circuitry. In future work, we will also explore this countermeasure in more detail and end-to-end, i.e., in terms of how many more traces are needed for PSC attacks with that countermeasure applied.

7 Conclusions and Future Work

In this work, we have provided a critical review of, and a successful attack on, a supposedly secure logic-locking technique called LoPher [SSC+20]. LoPher brings the security promises of cryptography to the domain of logic locking, and its strength lies in the idea of embedding (selected parts of) a design IP within multiple rounds of a block cipher, taking advantage of the inherent resistance of block ciphers against SAT-based attacks on locking.

To fully support different Boolean gates and varying interconnect topologies found in any regular design to lock, however, LoPher extends the block-cipher architecture to a) re-configure the S-Boxes and b) include an additional MUX layer between S-Boxes and permutation operations in every cipher round. We have critically analyzed these architectural changes and noted that adding such layers comes with considerable impact on the claimed security: while the main part of the key space of LoPher is claimed to be 2^{128} (for 64-bit K_c and 64-bit K_s configuration keys), our analysis shows otherwise. We show that the value of K_c is closely related to the value of K_s, which means there are multiple K_c and K_s assignments that can act as equivalent keys. This reduces the effective key space drastically.

We have exploited this fact in the second major contribution of this work: a tailored correlation power analysis attack. We have attacked a set of ISCAS-85 circuits (locked and provided as courtesy by the LoPher authors), where we successfully retrieved all the K_s and K_c key combinations, either the exact ones or equivalent ones, in only few thousands traces, even in the presence of considerable noise for the power traces. Our attack clearly demonstrates that LoPher is vulnerable to power side-channel analysis.

We have also outlined a simple countermeasure to render LoPher more resilient against such attacks. Toward this end, we first explored the nature of weak key assignments which are easy to detect by correlation analysis and thus reduce the effective key space. Second, based on that analysis, we also propose the use of strong and randomized key assignments, in particular for the K_s nibble of all unused S-Boxes. Note that this countermeasure requires no additional circuitry and, thus, incurs no cost or overheads to LoPher.

In future work, we will extend our experimental study to an FPGA implementation of LoPher including the proposed countermeasure. Recall that we already model a real-world scenario of noisy power traces, but using an FPGA and actual power measurement is still relevant. We also aim to study another countermeasure: given the minimum number of traces required to break LoPher under varying noise, an additional system-level circuitry may control and revise the embedding of the design IP at runtime. This is feasible since circuits can be embedded in many different ways into the LoPher fabric, given the flexibility of assignment of gates and routing topologies to different S-Boxes and a varying number of cipher rounds to use for LoPher. However, given the presence of equivalent keys, such a countermeasure would require a careful investigation for its true benefits.

References

[APK+21] Lilas Alrahis, Satwik Patnaik, Faiq Khalid, Muhammad Abdullah Hanif, Hani Saleh, Muhammad Shafique, and Ozgur Sinanoglu. Gnnunlock: Graph neural networks-based oracle-less unlocking scheme for provably secure logic locking. In *Proc. Des. Autom. Test Europe*, pages 780–785, 2021.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Proc. Cryptogr. Hardw. Embed. Sys.*, 2004.

[BDG16] Alex Biryukov, Daniel Dinu, and Johann Großschädl. Correlation power analysis of lightweight block ciphers: From theory to practice. In *Applied Cryptography and Network Security*, pages 537–557, 2016.

[BKL+07] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In *Proc. Cryptogr. Hardw. Embed. Sys.*, pages 450–466, 2007.

[FS03] Niels Ferguson and Bruce Schneier. *Practical Cryptography.* John Wiley & Sons, Inc., USA, 1 edition, 2003.

[GM11] Tim Güneysu and Amir Moradi. Generic side-channel countermeasures for reconfigurable devices. In *Proc. Cryptogr. Hardw. Embed. Sys.*, pages 33–48, 2011.

[GMOP15] Andreas Gornik, Amir Moradi, Jürgen Oehm, and Christof Paar. A hardware-based countermeasure to reduce side-channel leakage: Design, implementation, and evaluation. *Trans. Comp.-Aided Des. Integ. Circ. Sys.*, 34(8):1308–1319, 2015.

[Hey01] Howard Heys. A tutorial on linear and differential cryptanalysis. *Cryptologia*, 26, 2001.

[HFPM18] George Hatzivasilis, Konstantinos Fysarakis, Ioannis Papaefstathiou, and Harry Manifavas. A review of lightweight block ciphers. *J. Cryptogr. Eng.*, 8:1–44, 06 2018.

[KAHS19] Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, and Avesta Sasan. Full-lock: Hard distributions of SAT instances for obfuscating circuits using fully configurable logic and routing blocks. In *Proc. Des. Autom. Conf.*, pages 89:1–89:6, 2019.

[KGS+11] Armin Krieg, Johannes Grinschgl, Christian Steger, Reinhold Weiss, and Josef Haid. A side channel attack countermeasure using system-on-chip power profile scrambling. In *Proc. Int. On-Line Test Symp.*, pages 222–227, 2011.

[KRRT10] Ramesh Karri, Jeyavijayan Rajendran, Kurt Rosenfeld, and Mohammad Tehranipoor. Trustworthy hardware: Identifying and classifying hardware trojans. *Computer*, 43(10):39–46, 2010.

[LBC18] Owen Lo, William J. Buchanan, and Douglas Carson. Correlation power analysis on the present block cipher on an embedded device. In *Proc. Int. Conf. Avail., Rel. Sec.*, ARES '18, 2018.

[MS16] Amir Moradi and François-Xavier Standaert. Moments-correlating DPA. In *Proc. Theory of Implementation Security*, pages 5–15, 2016.

[Pat00] Dhiren Patel. The AES winner. 2000.

[PHJ+17] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. Side-channel analysis and machine learning: A practical perspective. In *Proc. IJCNN*, pages 4095–4102, 2017.

[RD20] Mark Randolph and William Diehl. Power side-channel attack analysis: A review of 20 years of study for the layman. *Cryptography*, 4(2), 2020.

[SL22] Dominik Sisejkovic and Rainer Leupers. *Logic Locking: A Practical Approach to Secure Hardware.* Springer Nature, 2022.

[SLS21] Abhrajit Sengupta, Nimisha Limaye, and Ozgur Sinanoglu. Breaking CAS-Lock and its variants by exploiting structural traces. Cryptology ePrint Archive, Paper 2021/581, 2021.

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - a clear roadmap for side-channel evaluations. In *IACR Crypt. ePrint Arch.*, 2015.

[SRM15] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *Proc. Int. Symp. Hardw.-Orient. Sec. Trust*, pages 137–143, 2015.

[SS19] Deepak Sirone and Pramod Subramanyan. Functional analysis attacks on logic locking. In *Proc. Des. Autom. Test Europe*, 2019.

[SSC+20] Akashdeep Saha, Sayandeep Saha, Siddhartha Chowdhury, Debdeep Mukhopadhyay, and Bhargab B Bhattacharya. Lopher: Sat-hardened logic embedding on block ciphers. In *Proc. Des. Autom. Conf.*, pages 1–6, 2020.

[SW12] Sergei Skorobogatov and Christopher Woods. In the blink of an eye: There goes your AES key. In *IACR Crypt. ePrint Arch.*, number 296, 2012.

[SXTF20] Bicky Shakya, Xiaolin Xu, Mark Tehranipoor, and Domenic Forte. CAS-Lock: A security-corruptibility trade-off resilient logic locking scheme. *Trans. Cryptogr. Hardw. Embed. Sys.*, 2020(1):175–202, Nov. 2020.

[UGP24] Devanshi Upadhyaya, Maël Gay, and Ilia Polian. Locking-enabled security analysis of cryptographic circuits. *Cryptography*, 8(1), 2024.

[XS19] Yang Xie and Ankur Srivastava. Anti-SAT: Mitigating sat attack on logic locking. *Trans. Comp.-Aided Des. Integ. Circ. Sys.*, 38(2):199–207, 2019.

[XSTF17] Xiaolin Xu, Bicky Shakya, Mark M. Tehranipoor, and Domenic Forte. Novel bypass attack and BDD-based tradeoff analysis against all known logic locking attacks. In *Proc. Cryptogr. Hardw. Embed. Sys.*, 2017.

[YMRS16] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan J V Rajendran, and Ozgur Sinanoglu. SARLock: SAT attack resistant logic locking. In *Proc. Int. Symp. Hardw.-Orient. Sec. Trust*, pages 236–241, 2016.

[YMRS17] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan J V Rajendran, and Ozgur Sinanoglu. Ttlock: Tenacious and traceless logic locking. In *Proc. Int. Symp. Hardw.-Orient. Sec. Trust*, pages 166–166, 2017.

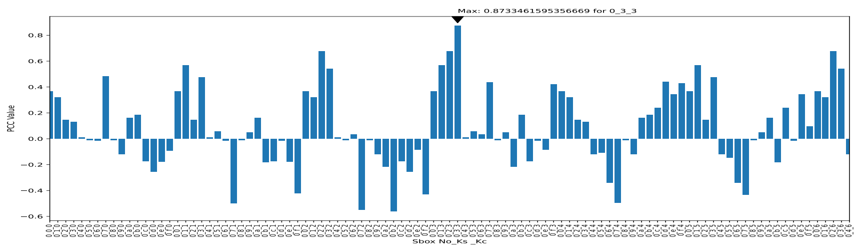
[YMSR17a] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Security analysis of Anti-SAT. In *Proc. Asia South Pac. Des. Autom. Conf.*, pages 342–347, 2017.

[YMSR17b] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan (JV) Rajendran. Removal attacks on logic locking and camouflaging techniques. *Trans. Emerg. Top. Comp.*, PP(99), 2017.

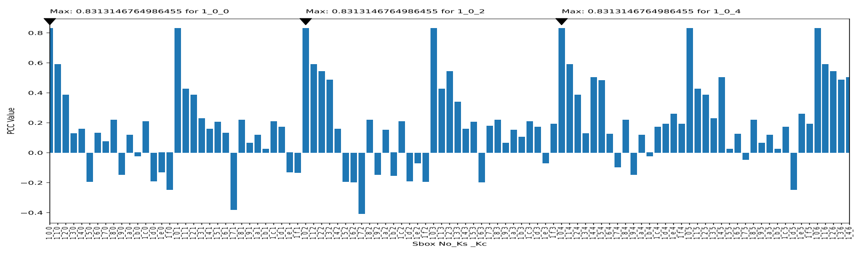
[YSN$^+$17] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan (JV) Rajendran, and Ozgur Sinanoglu. Provably-secure logic locking: From theory to practice. In *Proc. Comp. Comm. Sec.*, pages 1601–1618, 2017.

[YTS19] Fangfei Yang, Ming Tang, and Ozgur Sinanoglu. Stripped functionality logic locking with hamming distance based restore unit (SFLL-hd) – unlocked. *Trans. Inf. Forens. Sec.*, 2019.

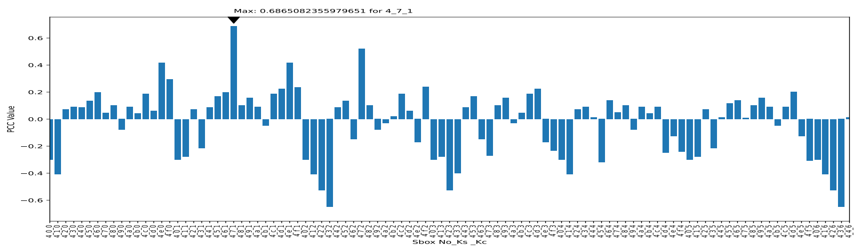
[ZF05] YongBin Zhou and DengGuo Feng. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. In *IACR Crypt. ePrint Arch.*, number 388, 2005.



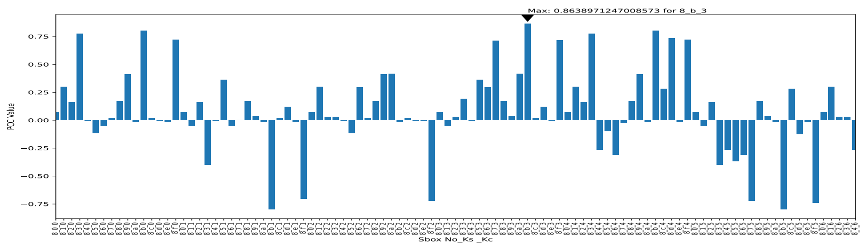
(a) (K_s, K_c) pair (3,3) with the highest PCC value is correctly identified for S-Box S0.



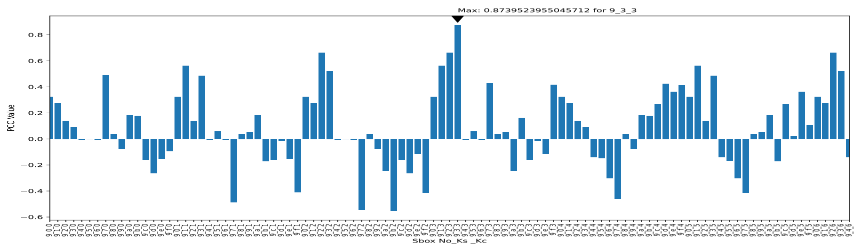
(b) (K_s, K_c) pairs with the highest PCC value are correctly identified for S-Box S1. Note that multiple peaks with the same PCC value are observed, as K_s is set to ‘0’.



(c) (K_s, K_c) pair (7,1) with the highest PCC value is correctly identified for S-Box S4.

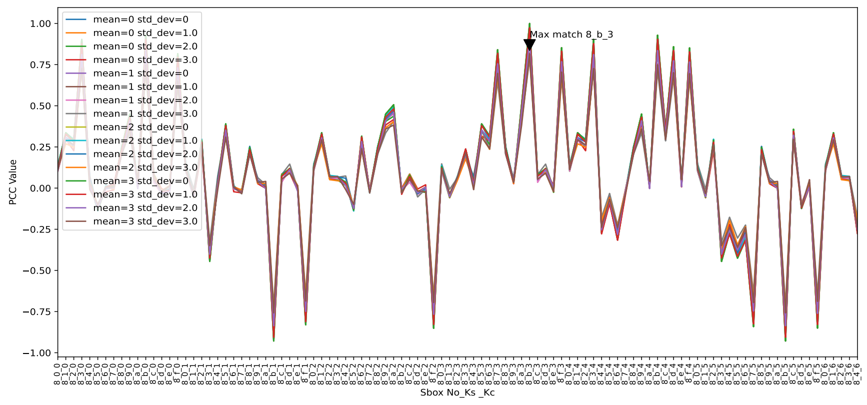


(d) (K_s, K_c) pair (b,3) with the highest PCC value is correctly identified for S-Box S8.

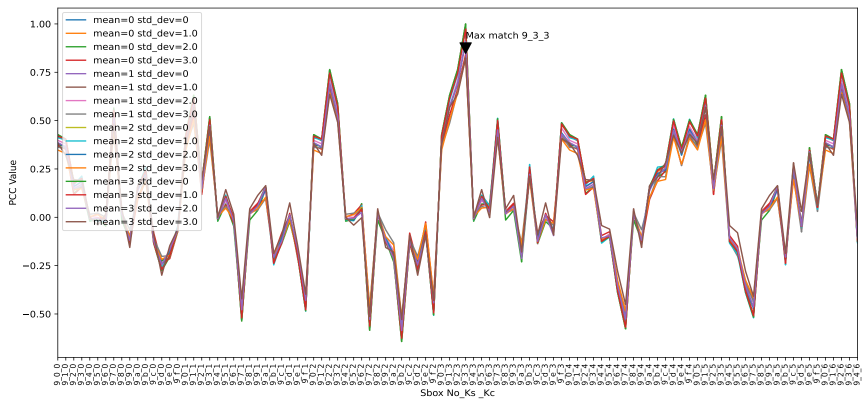


(e) (K_s, K_c) pair (3,3) with the highest PCC value is correctly identified for S-Box S9.

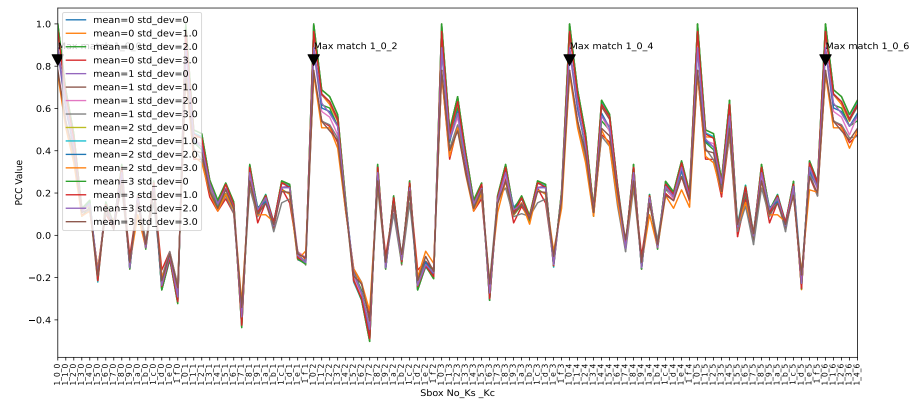
Figure 3: Attack on *c17* embedded in LoPher. We successfully recover the (K_s, K_c) pairs for the four relevant S-Boxes S0, S1, S4, S8 and S9.



(a) Variation of PCC under noise, S-Box S8 which is configured as a router.

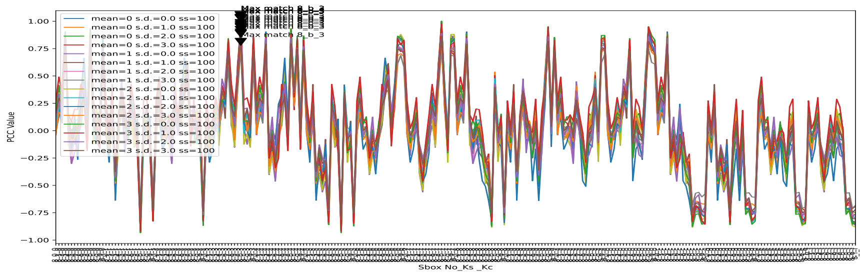


(b) Variation of PCC under noise, S-Box S9 which is configured as NAND gate.

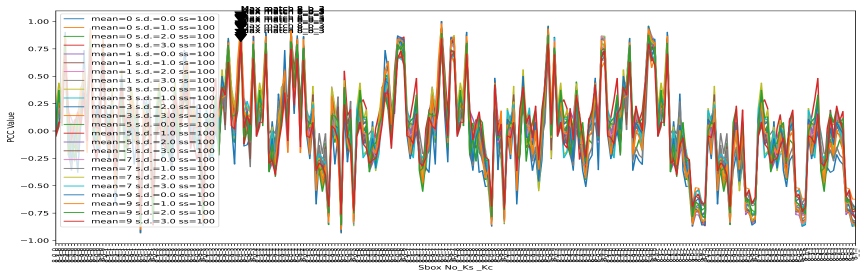


(c) Variation of PCC under noise, S-Box S1 which is not used.

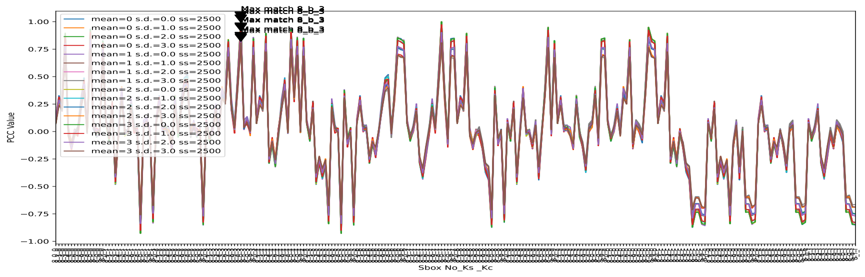
Figure 4: Impact of low-noise profiles on different types of S-Boxes in *c17*.



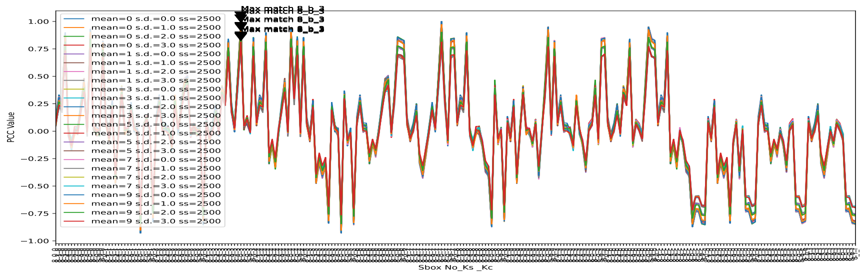
(a) Variation of PCC with the addition of noise for S8, low noise, # traces = 100



(b) Variation of PCC with the addition of noise for S8, high noise, # traces = 100

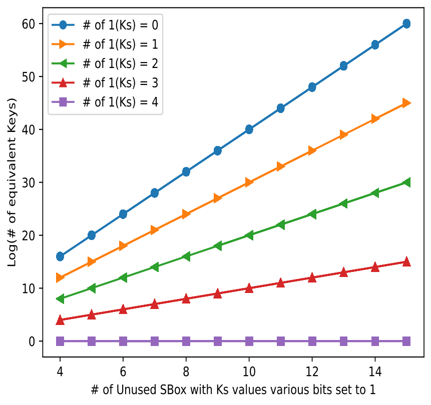


(c) Variation of PCC with the addition of noise for S8, low noise, # traces = 2,500

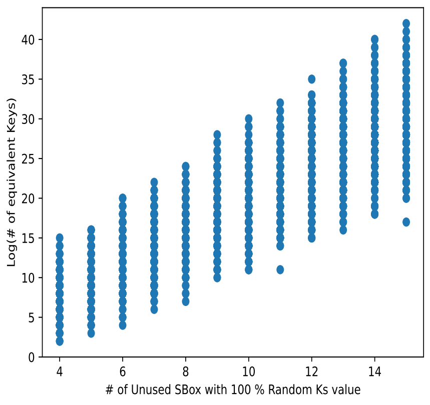


(d) Variation of PCC with the addition of noise for S8, high noise, # traces = 2,500

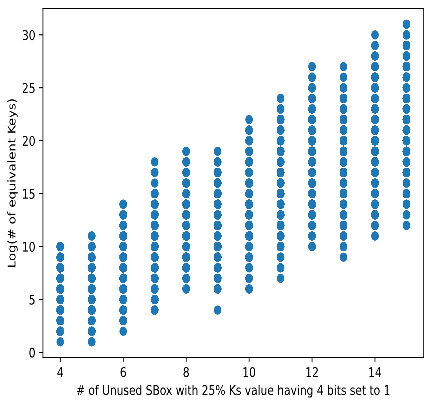
Figure 5: Impact of noise and number of traces on PCC values, for *c17*.



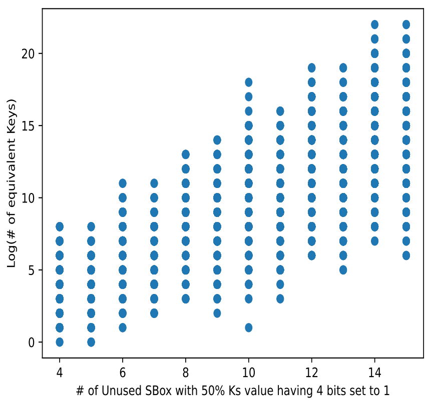
(a)



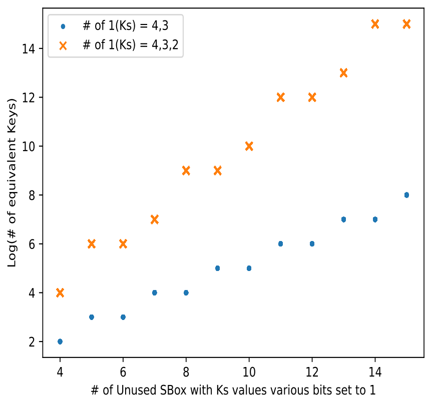
(b)



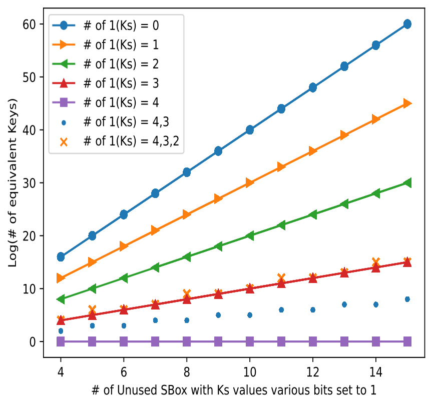
(c)



(d)



(e)



(f)

Figure 6: Analyzing different possible assignments of K_s nibbles for unused S-Boxes and their role for the number of equivalent (K_c, K_s) pairs. This analysis is done on 10,000 different, randomized logic and routing embeddings in a single round of LoPher.