HardCODE: Hardware-based Circuit Obfuscation using Data Encryption

Akashdeep Saha[1], Sayani Sinha[2], Chandan Kumar[2], Animesh Singh[2], Siddhartha Chowdhury[2], Sikhar Patranabis[3], and Debdeep Mukhopadhyay[2]

[1]New York University Abu Dhabi
[2]Indian Institute of Technology Kharagpur
[3]IBM Research India

October 23, 2025

Abstract

Indistinguishability Obfuscation ($i\mathcal{O}$) renders indistinguishable any pair of same-sized and functionally identical circuits. Since its introduction by Barak et al. (Crypto '01), $i\mathcal{O}$ has become the holy grail of modern cryptography due to the myriad of applications it enables. Despite several theoretical constructions (including recent breakthrough results from well-studied assumptions), practically efficient frameworks for $i\mathcal{O}$ have remained out of reach. In this paper, we formally introduce the notion of Hardware-based indistinguishability obfuscation ($Hi\mathcal{O}$) with an aim to make $i\mathcal{O}$ practical. Further, we present a novel framework called Hardware-based Circuit Obfuscation from Data Encryption (HardCODE) for realizing highly efficient $Hi\mathcal{O}$.

Given a simple tamper-proof hardware to store a fixed size secret key, our proposed HardCODE framework yields a provably secure hardware module that achieves $i\mathcal{O}$ for all poly-sized circuits, while additionally relying on *any* one-way function and certain (heuristic but widely studied) assumptions on substitution-permutation networks (used extensively in the design of secure block ciphers). We practically instantiate HardCODE from widely deployed realizations of a simple HSM (Hardware Security Module), and the substitution-permutation network underlying the PRESENT block cipher (an ISO standard for lightweight cryptography). We then use this instance of HardCODE to obfuscate, without loss of generality, the C-17 circuit of the ISCAS-85 benchmark suite, while incurring very small practical overheads. To the best of our knowledge, this constitutes the first practical demonstration of $i\mathcal{O}$.

Contents

1 Introduction **3**

1.1 Our Contributions 3

1.2 Technical Overview 6

2 Hardware-based Indistinguishability Obfuscation **10**

3 HardCODE: An Instantiation of HiO **12**

3.1 A Detailed Overview of HardCODE 12

3.2 Formal Description of HardCODE 18

3.2.1 Proof of Correctness. 22

3.3 Proof of Security 23

4 Hardness of Key Recovery from $\mathcal{E}'$ **28**

5 Application and Implementation **29**

A Some Related Definitions **38**

B An overview of PRESENT **39**

C A Note on Collision of Signals **39**

D Role of PRG Round Keys in Embedding Circuits **39**

E Encoding $\mathcal{E}'$ as SAT problem **41**

E.1 Key recovery from $\mathcal{E}'$. 43

F Logical Cryptanalysis of Augmented Block Cipher $\mathcal{E}'$ - An Example with PRESENT **44**

1 Introduction

Increasing globalization of the Integrated Circuit (IC) supply chain and recent advancements in reverse engineering have intensified security concerns surrounding Intellectual Property (IP) cores. This motivates techniques for safeguarding hardware IPs against potential attacks causing high financial losses. One such technique for IP protection is program/circuit obfuscation.

Circuit Obfuscation. Circuit obfuscation – the task of making code unintelligible while preserving its functionality – was first rigorously studied in [BGI+01]. The authors of [BGI+01] introduced two notions of obfuscation – virtual black-box (VBB) obfuscation, and indistinguishability obfuscation ($i\mathcal{O}$). VBB is a somewhat *ideal* notion of obfuscation, requiring that access to the obfuscated program gives no more power than access to an impenetrable black box with the same input-output functionality, and is impossible to realize in general. Most of the works on VBB obfuscators have been realized for only restricted (and mostly simple) classes of circuits [Can97, CD08, CRV10, BR14].

Indistinguishability Obfuscation. $i\mathcal{O}$ is weaker in that it only guarantees that two same-sized and functionally identical circuits ckt_0 and ckt_1 are rendered indistinguishable once obfuscated. This weakening allows $i\mathcal{O}$ to bypass the impossibility results for VBB obfuscation [BGI+01, GR07]. There exists a long line of works on realizing $i\mathcal{O}$ for all poly-sized circuits [GGH13, CLT13, PST14, GGH15, GGH+16, Lin16, LM18, AJS18, GJK18, Agr19, BDGM23], including recent breakthrough results from well-understood assumptions [JLS21, BCG+18, GGH+16, GP21, AJ15, BKP23] Today, $i\mathcal{O}$ is widely considered to be the holy grail of modern cryptography due to the myriad of applications it enables [SW14, BZ15, GGH+16, CHN+16, BZ17, BCG+18].

Despite its rich history of fascinating theoretical constructions, practically efficient frameworks for $i\mathcal{O}$ (for all poly-sized circuits) have remained out of reach. Indeed, existing lower bounds [GMM17] rule out provably secure black-box constructions of $i\mathcal{O}$ from even powerful cryptographic objects such as witness encryption [GGH13], predicate encryption [BW07, GVW15], and fully homomorphic encryption (FHE) [Gen09, BGV14] (assuming that one-way functions exist). While some recent efforts [SB25] attempt more efficient $i\mathcal{O}$ construction, they rely on non-standard assumptions. While there are known/folklore approaches for "augmenting" VBB obfuscators for specialized circuits into full-fledged $i\mathcal{O}$, such approaches also rely on complex cryptographic objects, such as (bootstrappable) FHE, and are thus costly in practice. This presents a massive hindrance towards deploying $i\mathcal{O}$ in practical applications.

$i\mathcal{O}$ in Practice. An alternative line of works [YSN+17, SXTF20, YMRS16, SSC+20] have proposed hardware/software-based obfuscation solutions that are tailored towards specific circuits/applications. While such solutions are significantly more efficient in practice, their lack of formal/provable security guarantees leaves them prone to cryptanalyis and structural attacks [SS19, SCMC22, SRM15]. In this paper, we focus on designing a practically efficient framework for iO that is generic (in particular, circuit-independent), equipped with formal security definitions, and amenable to provable guarantees based on standard cryptographic assumptions.

1.1 Our Contributions

Towards our goal of achieving provably secure yet practically efficient $i\mathcal{O}$, we propose a formal notion of hardware-based $i\mathcal{O}$. We then introduce, rigorously analyze, and concretely instantiate a novel framework called "Hardware-based Circuit Obfuscation from Data Encryption" (HardCODE)

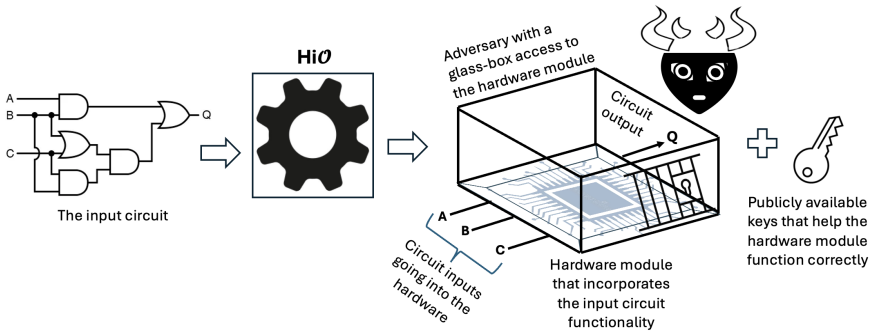


Figure 1: Overview of the proposed $Hi\mathcal{O}$.

that realizes a provably secure notion of $i\mathcal{O}$ *practically for the first time*, while relying on lightweight and efficiently realizable cryptographic objects.

Hardware-based $i\mathcal{O}$ ($Hi\mathcal{O}$). In this work, we formally introduce the concept of hardware-based indistinguishability obfuscation ($Hi\mathcal{O}$) in Section 2 for the first time in the literature (to the best of our knowledge). Along with that, we clearly define the system model and the adversarial behaviour. Typically, we expect $Hi\mathcal{O}$ to output a hardware module, which behaves as a specific circuit in the presence of correct keys, however, without revealing the underlying circuit. We allow the adversary to have 'glass-box' access to the hardware, which essentially means that the components on the hardware are known to adversary but it can not probe into those components. See Figure 1 for an overview of $Hi\mathcal{O}$.

HardCODE: an instantiation of $\boldsymbol{Hi\mathcal{O}}$**.** We next present a concrete instantiation of $Hi\mathcal{O}$, which we call HardCODE. Given a tamper-proof memory to securely store a small secret key, HardCODE framework achieves $Hi\mathcal{O}$ for all poly-sized circuits while additionally relying on any secure pseudorandom generator (PRG) and certain well-studied assumptions on the security of substitution-permutation networks (abbreviated as SPN) that have been used extensively by the cryptographic community in the design of secure (and standardized) block cipher algorithms such as AES [DR99], PRESENT [BKL$^+$07], and GIFT [BPP$^+$17]. Our approach is thus significantly more efficient in practice as compared to known approaches for achieving provably secure $i\mathcal{O}$.

We provide rigorous formal security guarantees for HardCODE. These guarantees are based on proof arguments that are very similar in flavor to those used in the cryptographic literature on block cipher cryptanalysis. Along the way, we present some novel observations on how one could "encode" any poly-sized Boolean circuit to a set of keys by embedding it using an SPN-based block cipher. To the best of our knowledge, such an approach has not been studied formally within the cryptographic community and could potentially be of independent interest with applications beyond $i\mathcal{O}$.

An Outline of Our Approach. We present a very high-level overview of our approach here. A more detailed technical overview is presented subsequently in Section 3.1. Our proposed HardCODE framework takes as input a Boolean circuit and produces as output an obfuscated circuit (in the form of a hardware module) that consists of two components: (a) an embedding of the circuit inside an SPN-based block cipher, and (b) some additional keys that can be used to evaluate the circuit on any given input. Concretely, the circuit embedding process produces three sets of keys for each round of the SPN-based block cipher, namely the "round" keys, the

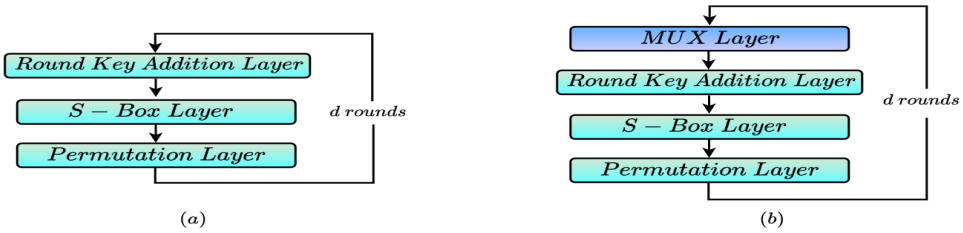


Figure 2: One round of (a) a SPN-based block cipher, (b) the proposed augmented SPN-based block cipher to enable circuit embedding retaining the security of block cipher.

"select" keys, and the "configuration" keys, which together encode the circuit embedding into the SPN block cipher. We defer the detailed overview of how these keys are used to encode any circuit inside an SPN-based block cipher to Section 3.1, but only mention here that, to ensure the security of the overall embedding process based on the security of the SPN-based block cipher, the round keys used by the circuit embedding need to be generated (pseudo)randomly, while the select and configuration keys need to be suitably masked/hidden from the adversary. An efficient yet cryptographically secure way of achieving this is to use a PRG to derive the round keys as well as the mask values for the select and configuration keys. In particular, this allows us to output the masked select and configuration keys as part of the obfuscated circuit (which can be suitably unmasked during evaluation).

The final piece for correct and secure evaluation of the obfuscated circuit is to use a "master secret key" as the seed of the PRG circuit and hide it from the adversary. We achieve this by securely storing the master secret key in tamper-proof hardware. In summary, the security of our $i\mathcal{O}$ construction hinges on securely storing a PRG seed. One feasible solution is employing a tamper-proof HSM [SZR+23, KSS23, Das23] for practical implementations.

Practical Instantiation. We show a concrete, practical, and highly efficient instantiation of the proposed HardCODE framework from (i) a standard HSM-based realization of a tamper-proof memory, and (ii) a novel usage of the SPN underlying the PRESENT block cipher [BKL+07] (an ISO standard for lightweight cryptography [BLA17,EKP+07]). We then use this instance of HardCODE to obfuscate a C-17 circuit of the ISCAS-85 benchmark suite [BF85], while incurring very small practical overheads. To the best of our knowledge, this constitutes the first practical illustration of $i\mathcal{O}$.

Theory vs Practice of $i\mathcal{O}$. Our approach departs from the traditional study of achieving provably secure (but practically inefficient constructions) $i\mathcal{O}$ from mathematical assumptions. We justify our approach by arguing that while the theoretical study of $i\mathcal{O}$ has enriched the understanding of the cryptographic community, the core application of $i\mathcal{O}$ (namely, program obfuscation) remains far from practically instantiable using any of these proposed constructions. Indeed, there is a huge disconnect between the theoretical study of $i\mathcal{O}$ and the non-cryptographic techniques often used in practice for circuit/program obfuscation in applications (such as hardware IP protection).

In particular, while our goal is to achieve practical realizations of $i\mathcal{O}$ and we choose to prioritize efficiency over provable security from mathematical assumptions, we rely on design and cryptanalysis techniques that have been widely studied and used within the cryptographic community over a considerable period, as opposed to heuristic non-cryptographic approaches that are often vulnerable to cryptanalysis [CGH17, BHJ+19, BBKK18, LV17] with potentially catastrophic consequences.

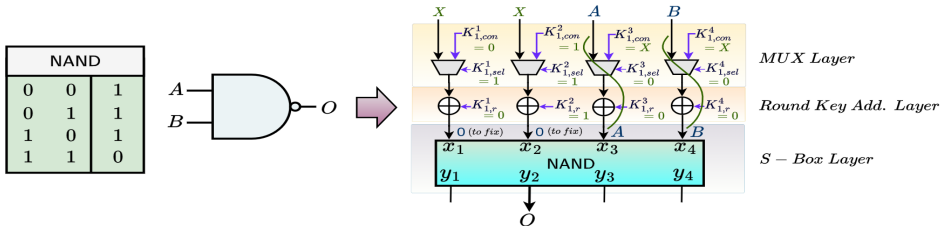


Figure 3: Instance of realizing a two input NAND (universal) gate with PRESENT S-Box. A and B are the inputs whereas O is the output. We set $K^3_{i,sel}$ and $K^4_{i,sel}$ to 0 to pass on A and B, and $K_{i,r}$ is 0 in both positions. $K^1_{i,sel}$ and $K^2_{i,sel}$ are 1 to set the configuration bits. $K^1_{i,con}$ and $K^2_{i,con}$ are decided based on $K_{i,r}$ at those positions. The output is obtained through the second output bit of S-Box.

1.2 Technical Overview

This section presents an overview of our HardCODE framework, which achieves the notion of $Hi\mathcal{O}$, as illustrated in Figure 1. We first discuss the technical centerpiece of HardCODE, which is essentially the novel usage of an "augmented" block cipher[1] to embed any Boolean circuit. We then discuss how HardCODE fits into the desired model of $Hi\mathcal{O}$.

Traditionally, SPN network-based block ciphers such as AES, PRESENT, and GIFT consist of a certain number of rounds, where each round consists of three consecutive layers – (i) the round key addition layer, (ii) the Substitution Box (S-Box) layer, and (iii) the permutation layer.

Embedding a Boolean Circuit. We now describe our approach of using an SPN structure to "embed" a Boolean circuit. To begin with, we augment the traditional three-layered SPN round to include an additional layer consisting of several multiplexer (abbreviated as "MUX") gates[2]. This new MUX layer is inserted between the permutation layer of round-$(i-1)$ and the round key addition layer of round-i (ref. Figure 2(b)). We propose a mechanism whereby the layers of the augmented SPN network (the MUX layer, the round key addition layer, and the S-Box layer from round-i) can be used to embed various Boolean logic gates. The MUX layer from round-i is used to establish the routing of the various wires connecting the gates in round $(i-1)$ with the gates in round i, such that specific wires are forwarded to specific input bits of the S-Boxes in round i depending on the circuit to be embedded.

We now provide an example for embedding a simple circuit consisting of a single (universal) 2-input NAND gate inside a single round of an augmented version of the PRESENT block cipher [BKL+07] (see Figure 3 for an illustration of our approach). A PRESENT S-Box has 4 inputs and 4 outputs. In particular, letting (x_1, x_2, x_3, x_4) denote the input bits and (y_1, y_2, y_3, y_4) denote the output bits of the S-Box, the second output bit (y_2) is given by

$$y_2 = x_1x_2x_4 \oplus x_1x_3x_4 \oplus x_1x_3 \oplus x_1x_4 \oplus x_1 \oplus x_2 \oplus x_3x_4 \oplus 1,$$

Now, observe that if we fix x_1 and x_2 to 0, the above equation simplifies to $y_2 = x_3x_4 \oplus 1$. In other words, we achieve $y_2 = \mathsf{NAND}(x_3, x_4)$, i.e., for a particular configuration of the PRESENT S-Box where the first and second input bits are fixed to 0, the second output bit of the S-Box can be used to evaluate the NAND of the third and fourth input bits.

A natural question is:

[1] By block cipher, we imply its substitution-permutation network (SPN) structure.
[2] Throughout this paper, we only consider $2:1$ MUX gates

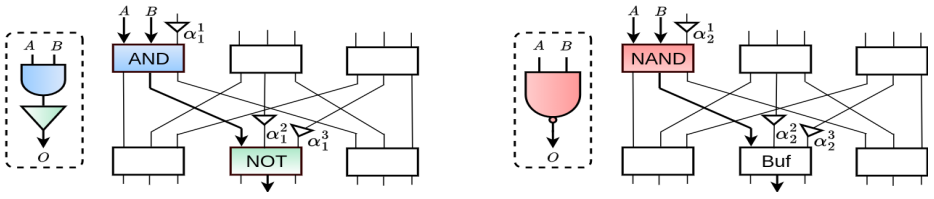


Figure 4: Indistinguishability between the embedding of two functionally equivalent circuits (inset: AND-NOT and NAND) into a sample SPN-based block cipher. Without loss of generality, initial and final buffer rounds, AddRoundKey layer, and the MUX layer for all intermediate bits are not shown. The keys vary in position, denoted as (α_1^1, α_2^1), (α_1^2, α_2^2), and (α_1^3, α_2^3), due to the S-Boxes implementing different functions.

how do we fix specific input bits to 0 *in an intermediate round of the block cipher?*

This is where the MUX layer comes in. It offers the flexibility of fixing certain bits input to the S-Box (x_1 and x_2 in the example above) to 0, while only forwarding the desired input bits (x_3 and x_4 in the example above) from the previous round into the S-Box. In the specific instantiation of the NAND gate illustrated in Figure 3, the corresponding round key bits $K_{1,r}^3$ and $K_{1,r}^4$ were set to 0, which allowed the bits x_3 and x_4 to reach the S-Box unaltered through the round key addition layer. However, in other scenarios, certain round key bits could be set to 1. We address these scenarios in Section 3.2.

The desired flexibility of fixing input bits offered by the MUX layer leads to the requirement of two additional keys per round beyond the usual block cipher round key $K_{i,r}$. The select key $K_{i,sel}$ is used to determine the select bits input to the MUX layer in the i^{th} round, while the configuration key $K_{i,con}$ is used to supply fixed-bit inputs to the MUX layer in the i^{th} round. One needs to set these key bits carefully so that the circuit embedding retains its functional correctness. See Section 3.1 (ref. Figure 8) for a more detailed exposition.

Generating and Masking the Various Keys. When a circuit is embedded into the augmented block cipher using HardCODE, the embedding can be uniquely represented by three sets of keys for each cipher round, namely, round keys ($K_{i,r}$), select keys ($K_{i,sel}$), and configuration keys ($K_{i,con}$). Hence, the distinctive circuit realization lies in the keys, facilitating the SPN-based structure's adaptability to any specified circuit. Illustrating this, we present an example of embedding two functionally equivalent circuits: one featuring a NAND gate and the other an AND-NOT gate in Figure 4. We show that these two circuits, when embedded will have different keys while their underlying structure remains the same. In the initial round, the first S-Box functions as an AND gate in one embedding and a NAND gate in the other, requiring distinct key configurations at positions α_1^1 and α_2^1 in Figure 4. Similar variations in key configurations occur in subsequent rounds, indicating differences between NOT and buffer gates. Note that a circuit can be embedded with different keys, which form a set representing the given circuit. However, for the two functionally equivalent circuits mentioned above, their functionally equivalent keys form disjoint sets. In this way, the proposed HardCODE encapsulates every circuit as a function of keys, formalized in Algorithm 3.

Let us assume that each of the round keys, select keys, and configuration keys of an embedded circuit has size L bits, considering all the rounds together. Then, in our HardCODE construction, we use a pseudorandom generator (PRG) [GGM86], which outputs a pseudorandom string of length $3L$ bits. We summarize the different usage of this $3L$-bits long PRG output in the following:

- *Generating round keys*: The last L bits of PRG output are used the same way as the round

keys in a typical block cipher. Particularly, a round key for i^{th} cipher round is XORed in a bitwise manner with intermediate states, generated from $(i-1)^{th}$ cipher round. The round keys, being the output of PRG, can not be guessed efficiently by a PPT (probabilistic polynomial time) adversary as long as the seed of PRG is secret. Hence, the seed of the PRG essentially serves as the *master secret key* for the overall HardCODE construction, and we use a tamper-proof hardware to securely store the PRG seed, i.e., the master secret key.

- *Masking select and configuration keys*: As already established, select and configuration keys of a HardCODE-obfuscated circuit contain crucial information about circuit embedding; hence, must be kept secret. Further, to alleviate the need for storing long ($2L$-bits, where the value of L linearly increases with the number of cipher rounds required to embed a circuit) secret keys for an obfuscated circuit, we mask (xor) them with pseudorandom bits, generated from the PRG, and make the masked select and configuration keys publicly available. In particular, the first L bits of PRG output are used to mask L-bit select key, whereas next L bits of PRG output are used to mask the L-bit configuration key, thus preventing any information leakage. We then refer to these masked select keys (denoted with $K_{i,msel}$) and masked configuration keys (denoted with $K_{i,mcon}$) as *publicly available keys*.

- *Unmasking the masked select and configuration keys*: When the obfuscated circuit (ref. Figure 9) emulates the behavior of the underlying circuit in the presence of publicly available keys (masked select and masked configuration keys), they are unmasked inside the circuit using the same first $2L$ number of bits of PRG output to retrieve actual select and configuration keys.

Putting Everything Together. From the above description, the augmented 4-layer SPN network can be used to embed any circuit. We refer to this augmented SPN network and the above embedding mechanism for any Boolean circuit as HardCODE (detailed in Section 3.1). Given a circuit ckt, HardCODE outputs an obfuscated circuit in the form emulator box $\mathcal{B}$, two lists IB and OB specifying the positions of circuit inputs and outputs in the augmented SPN network, and a set of two publicly available keys $(K_{i,msel}, K_{i,mcon})$, which are masked using a pseudorandom bit-string generated from a PRG with a fixed size seed. For a practical implementation of our PRG, we employ AES because it is widely regarded as secure and efficient, making it the standard for practical applications [Gue09]. The evaluation of obfuscated circuit needs IB, OB, and publicly available keys for emulating the behavior of original circuit (ref. Figure 9).

HardCODE achieves $Hi\mathcal{O}$. The HardCODE framework adheres to the notion of $Hi\mathcal{O}$ (Figure 1) since it outputs a hardware module (also referred to as emulator box $\mathcal{B}$ described in Figure 9) which is designed to emulate the functionality of the original circuit in the presence of a specific set of keys. The module has an augmented SPN inside it, along with a PRG circuit and a tamper-proof on-chip memory storing the PRG seed. The components inside the module cannot be accessed from outside. When the module is in function (as an obfuscated circuit), one can only provide input to the module through the positions specified in IB and observe the output at positions specified in OB in the presence of the correct (masked select and masked configuration) keys.

Correctness, Efficiency, Security. We formally prove functional correctness of HardCODE as $Hi\mathcal{O}$ in Section 3; the proof relies on the correctness and efficiency guarantees of the underlying cryptographic primitives. We also prove that HardCODE realizes a secure $Hi\mathcal{O}$ in the adversarial model discussed in Section 2 by establishing the following three security guarantees (stated informally here for ease of exposition; see Section 3 for rigorously formal versions and proofs):

Table 1: Resilience of HardCODE against reported attacks on locked circuits.

SAT-based [SRM15]	AppSAT [Sea17]	FALL [SS19]	DIP L. [SCMC22]	[MZS24]	OMLA [APSS21]	NNgSAT [Aea20]
Time Out	**Time Out**	✗	✗	✗	✗	✗

- **Indistinguishability of output keys:** We rely on a secure PRG and fixed size secure storage in the form of tamper-proof memory that stores the PRG seed to show that the publicly available keys corresponding to two different HardCODE-obfuscated circuits are computationally indistinguishable as long as the underlying circuits are functionally equivalent (ref. Lemma 4).
- **Resistance to cryptanalysis:** We prove that HardCODE resists traditional cryptanalytic attacks on block ciphers by relying on the security of the underlying block cipher. In particular, we show that as long as the original SPN-based block cipher is secure against state-of-the-art SAT-solver based cryptanalytic techniques, the augmented block cipher with the MUX layer retains the same security guarantees against cryptanalytic attacks. In other words, introducing the MUX layer does not degrade core security guarantees of the block cipher (ref. Section 4).
- **Resistance to side-channel leakage:** Each HardCODE round contains some S-Boxes functioning as logic gates, contributing to essential computations, while others are active but perform no meaningful computations, adding noise. Hence, we expect HardCODE to have a lower Signal-to-Noise Ratio (SNR) and validated through Test Vector Leakage Assessment (TVLA) of our hardware implementation with 10^5 fixed and 10^5 random traces (ref. Section 5).

Comparison with Related Techniques. Logic locking integrates key-dependent logic directly into a circuit's netlist, categorized into pre-SAT and post-SAT methods. Provably-secure Stripped Functionality Logic Locking (SFLL) relies on a hard-coded Hamming-distance comparator but succumbs to FALL (functional analysis-based attack) [SS19]; CAS-Lock's AND-OR chain [SXTF20] can be excised, and Full-Lock's configurable logic networks [KAHS19] with lightweight switch-boxes remain vulnerable to neural-network-guided SAT attacks [Aea20] and topology inference. Conversely, our augmented SPN (central to HardCODE) withstands all tested practical attacks, including SAT-based [SRM15, Sea17], ML-guided [APSS21, Aea20], reverse engineering [SS19], and sensitization-based approaches [MZS24], either failing or timing out (Table 1). Traditional camouflaging techniques require extensive custom layouts and additional mask layers, increasing fabrication complexity and manual effort [RPK+21]. Recognizing the practical limitations of theoretical constructions and the security gaps in non-cryptographic obfuscation like logic locking, we propose HardCODE, the first reported practical attempt bridging theory and implementation in program obfuscation. Although embedding a circuit within block cipher rounds resembles programming an FPGA via encrypted bitstreams [Tri03], a crucial distinction exists: FPGA bitstreams are explicitly decrypted before configuration, activating a fixed logic set. In contrast, our embedding employs a randomized, secret-dependent selection of S-Boxes, actively participating during evaluation, thereby ensuring indistinguishability between functionally equivalent circuits. The authors in [GIS+10] model circuit obfuscation as a two-party protocol using encrypted circuit and input, incorporating tamper-proof hardware similar to ours. However, their underlying technique is fundamentally different from ours, and they provide no practical instantiation.

2 Hardware-based Indistinguishability Obfuscation

Theoretically, an indistinguishability obfuscator transforms a given circuit into an obfuscated circuit that preserves its functionality while concealing its exact structure from a PPT adversary. However, constructing practical instantiations of indistinguishability obfuscators remains a significant challenge. While some efforts [NFR+17, GIS+10] have explored circuit/program obfuscation using hardware, none has been proven to achieve indistinguishability obfuscation. Furthermore, there is a need to formally define indistinguishability obfuscation in the context of practical implementations. In this section, we bridge the gap between theory and practice by formally introducing the notion of hardware-based indistinguishability obfuscator. Specifically, rather than producing an obfuscated circuit, we propose substituting it with a hardware module whose architecture and internal components are entirely publicly available. This hardware module is designed to preserve the correct input-output functionality while preventing extracting any meaningful structural or internal information about the embedded circuit beyond what is intrinsically disclosed by its functional behavior. We formalize this concept as follows.

System model. Consider a scenario where a client U requests a vendor V to provide a hardware solution for a specific functionality. In response, V designs a circuit ckt that meets the specifications outlined by U. However, as the intellectual property (IP) owner, V is unwilling to disclose the circuit details to prevent potential IP theft. To address this, V employs a hardware-based indistinguishability obfuscator to transform the circuit ckt into a hardware module that obfuscates the underlying design. This module and a set of finite-sized keys are then transferred to U. Using the provided hardware module and the correct set of keys, U can accurately simulate the input-output behavior of the desired functionality but is unable to derive any meaningful information about the circuit ckt. In this model, U functions as an adversary, attempting to distinguish between two hardware modules and their corresponding keys, each representing a different design of the same functionality. We now examine the capabilities and limitations of U as an adversary.

Adversarial model. We grant U a "glass-box" access to the hardware module. This "glass-box" access implies that the adversary can observe both the design and layout of the components within the hardware module and the input-output behavior when the correct set of keys is applied. However, based on the extent of the adversary's probing capabilities, we define two distinct types of adversarial models as follows.

- *No Probing Allowed.* Here, we assume that the adversary cannot read from/write to the wires/buses of the hardware module as if it is kept inside a glass-box to prevent any probing. Our instantiation of $Hi\mathcal{O}$, namely $\mathsf{HardCODE}$ (Section 3), is proven to be secure in this adversarial model.
- *Passive Probing Allowed.* The adversary is capable of passively monitoring the hardware module to observe and analyze potential side-channel leakages, such as power consumption. Since our proposed construction ($\mathsf{HardCODE}$) does not explicitly incorporate side-channel countermeasures, we do not claim its security in this adversarial model. However, a preliminary Test Vector Leakage Assessment (TVLA) conducted on our implementation suggests the absence of exploitable power side-channel leakage.

Next, we formally define the hardware-based indistinguishability obfuscator.

Definition 1 (Hardware-based Indistinguishability Obfuscator ($Hi\mathcal{O}$))**.** *A hardware-based in-*

distinguishability obfuscator (HiO) consists of two PPT algorithms as follows,

$$HiO = (HiO.\mathsf{Obfuscate}, HiO.\mathsf{Evaluate}).$$

$\underline{HiO.\mathsf{Obfuscate}(\mathsf{ckt}_f, pp)}$: Given the description of a circuit ckt_f for a function $f : \{0,1\}^\alpha \to \{0,1\}^\beta$ and some public parameters pp, this randomized algorithm produces a tuple $\langle M_f, K_f, I_{aux}\rangle$, where M_f is a hardware module, K_f is a set of finite-sized keys and I_{aux} is some publicly available auxiliary information, which is used by M_f during evaluation of ckt_f.

$\underline{HiO.\mathsf{Evaluate}(\langle M_f, K_f, I_{aux}\rangle, x \in \{0,1\}^\alpha)}$: This is a deterministic algorithm that takes the hardware module M_f, the set of keys K_f, the public information I_{aux}, and any valid $x \in \{0,1\}^\alpha$ as inputs and outputs $y \in \{0,1\}^\beta$. In other words, M_f takes input x and keys K_f and outputs y, such that $y = f(x)$.

We list three desired properties of a HiO below.

- *Correctness:* A hardware-based indistinguishability obfuscator HiO is said to satisfy correctness if the following holds $\forall x \in \{0,1\}^\alpha$ and $\forall \mathsf{ckt}_f \in \mathsf{CKT}_{f,d}$ for some $d \in \mathbb{N}$,

  $$\mathsf{Pr}[HiO.\mathsf{Evaluate}(HiO.\mathsf{Obfuscate}(\mathsf{ckt}_f, pp), x) = f(x)] = 1.$$

  Here, $\mathsf{CKT}_{f,d}$ denotes the set of all possible d-depth circuits for function f. We assume d to be a part of public parameters pp.

- *Polynomial slowdown:* Let $\mathcal{T}(\cdot)$ denote the runtime of some given function on a certain input. We say that HiO satisfies the polynomial slowdown property, if $\forall x \in \{0,1\}^\alpha$,

  $$\mathcal{T}(HiO.\mathsf{Evaluate}(\langle M_f, K_f, I_{aux}\rangle, x)) = \mathsf{poly}(\mathcal{T}(f(x))).$$

- *Security:* Let us consider a security game $\mathsf{Expt}_{1^\lambda,\mathcal{A}}$ between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$ as follows.

  $\underline{\mathsf{Expt}_{1^\lambda,\mathcal{A}}}$:

  1. The challenger $\mathcal{C}$ sends the public parameters pp to the adversary $\mathcal{A}$.
  2. $\mathcal{A}$ chooses two different circuits ckt_f^0 and ckt_f^1 for the same function $f : \{0,1\}^\alpha \to \{0,1\}^\beta$ and sends them to $\mathcal{C}$.
  3. $\mathcal{C}$ chooses a uniform random bit $b \xleftarrow{\$} \{0,1\}$, executes $HiO.\mathsf{Obfuscate}(\mathsf{ckt}_f^b, pp)$, and returns the output $\langle M_f^b, K_f^b, I_{aux}^b\rangle$ to $\mathcal{A}$.
  4. $\mathcal{A}$ outputs a distinguishing bit b', which is considered to the output of the game $\mathsf{Expt}_{1^\lambda,\mathcal{A}}$.

  A hardware-based indistinguishability obfuscator HiO is said to be secure if for any PPT adversary $\mathcal{A}$ with specific adversarial capabilities (discussed later in this section), the following holds,

  $$\big|\mathsf{Pr}[\mathsf{Expt}_{1^\lambda,\mathcal{A}}(HiO.\mathsf{Obfuscate}(\mathsf{ckt}_f^0, pp)) = 0] - \mathsf{Pr}[\mathsf{Expt}_{1^\lambda,\mathcal{A}}(HiO.\mathsf{Obfuscate}(\mathsf{ckt}_f^1, pp)) = 0]\big| < \mathsf{negl}(\lambda).$$

  Informally, no PPT adversary distinguishes between the obfuscations of two functionally equivalent circuits except with negligible probability.

3 HardCODE: An Instantiation of HiO

Our proposed HardCODE (Hardware-based Circuit Obfuscation using Data Encryption) framework essentially represents a practical instantiation of hardware-based indistinguishability obfuscator, described in Section 2. Here, we first provide an overview of circuit obfuscation from SPN-based block cipher in Section 3.1. Then, a formal construction of our proposed hardware-based circuit obfuscator HardCODE in Section 3.2, followed by proof of security in Section 3.3. The backbone of our construction is an augmented block cipher, whereby "augmentation", we mean an additional MUX layer before each round of our block cipher. For ease of exposition, we use PRESENT (ref. Appendix B for an overview) as an underlying block cipher while explaining a few examples in this section. However, we would like to emphasize that it can be replaced with other SPN-based block ciphers, for e.g., GIFT. Before going into the discussion, we introduce some frequently used notations here.

Notations. We write $x \xleftarrow{\$} \mathcal{X}$ to represent that an element x is sampled independently and uniformly at random from a set/distribution $\mathcal{X}$. Inside algorithms, $x \leftarrow a$ implies the assignment of value a to variable x. We denote by $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ any generic (unspecified) polynomial function and negligible function in security parameter $\lambda \in \mathbb{N}$, respectively. We denote the set $\{1, 2, \cdots, n\}$ by $[n]$ for some $n \in \mathbb{N}$. Similarly, $[m, n]$ denotes the set $\{m, m+1, \cdots, n\}$ for some $m, n \in \mathbb{N}$ and $m < n$.

3.1 A Detailed Overview of HardCODE

Our proposed HardCODE construction provides a novel technique for achieving HiO with SPN-based block ciphers. Let an SPN-based block cipher realize a mapping $\mathcal{E} : \mathcal{P} \times \mathcal{K} \rightarrow \mathcal{C}$, where $\mathcal{K} = \{0,1\}^\lambda$ is the key space, $\mathcal{P} = \{0,1\}^n$ is the plaintext space, and $\mathcal{C} = \{0,1\}^n$ is the ciphertext space. Let d be the total number of rounds in the block cipher, then every i^{th} $(\forall i \in [d])$ round of $\mathcal{E}$ contains the round key addition layer (*addRoundKey*), S-Box layer (*sBoxLayer*), and the permutation layer (*pLayer*) as illustrated in Figure 7(*a*). Let $f : \{0,1\}^\alpha \rightarrow \{0,1\}^\beta$ be a function that realizes a mapping from α-bit input space to β-bit output space. One can always represent f as a Boolean circuit ckt_f consisting of logic gates and some fixed interconnections between them. Note that several distinct circuit representations of the same function f are feasible using different logic gates. We call them functionally equivalent circuits. Provided a particular circuit representation ckt_f of the function f, we show how to realize an obfuscation of ckt_f using block cipher $\mathcal{E}$, such that not only the functionality of f is preserved in the obfuscated circuit, but the obfuscation of ckt_f is also indistinguishable from obfuscation of some functionally equivalent circuit ckt'_f, in the view of a PPT adversary. Thus, the proposed HardCODE technique satisfies the *correctness* and *security* properties as required by a hardware-based indistinguishability obfuscator (ref. Definition 1). Furthermore, we can embed ckt_f in $\mathcal{E}$ by using $d = \mathsf{poly}(\ell)$ number of rounds, where ℓ is the number of topological levels of ckt_f. Thus, HardCODE also satisfies the *polynomial slowdown* property, detailed in Lemma 3 (Section 3.2).

A circuit typically consists of logic gates and wires that interconnect them. In a broad sense, our proposed HardCODE realizes logic gates from S-Boxes and the interconnections from the *pLayer* of a block cipher. Figure 5 illustrates the realization of various logic gates from PRESENT S-Box. An S-Box can also realize a buffer gate, which passes a bit unaltered from the S-Box input to the output. The number of signals an S-Box can pass through varies depending on the choice of the block cipher. For example, a PRESENT S-Box acting as a buffer can pass through at most two input signals (the rightmost S-Box in Figure 6). A multiple fan-out buffer gate,

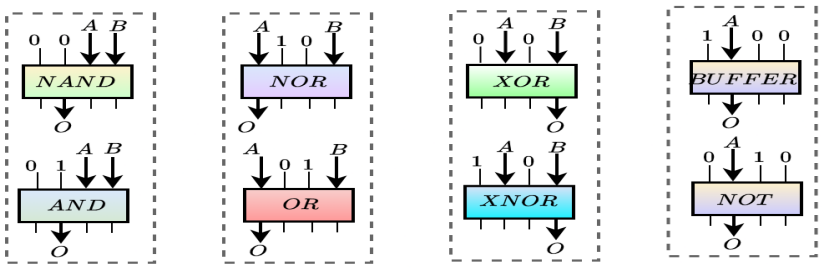


Figure 5: Realization of logic gates from PRESENT S-Box with A, B as input signals and O as the output signal. S-Boxes within dotted lines realize complementary gates having the same input-output positions.

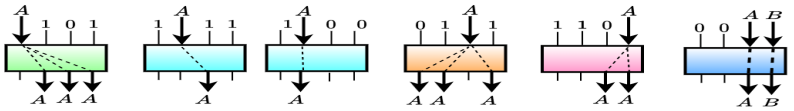


Figure 6: Realization of buffer gates which pass a signal and multiple fan-out buffer gates which duplicate signals from input to output with PRESENT S-Box.

which is able to duplicate an input bit to multiple outputs, can also be realized from an S-Box. Figure 6 illustrates the formation of buffer gates from PRESENT S-Box. Likewise, it is always possible to realize the functionality of logic/buffer gates from S-Boxes of any SPN-based block cipher by just observing the construction and input-output properties of its S-Boxes [SSC+20].

To realize a Boolean gate, a straightforward modification to an SPN-based block cipher is required so that we can fix certain input bits of an S-Box. As a solution, an additional keyed MUX layer is added prior to the *addRoundKey*, (ref. Figure 7(b)). This layer assists in fixing the specific S-Box input bits of *sBoxLayer* in the intermediate rounds to act as the desired logic/buffer gates. MUX layer of each round comprises n MUXs, corresponding to every intermediate state bit of the cipher. Thus, every S-Box input must pass through a MUX gate first. One of the two inputs for each MUX comes from an S-Box of the previous layer, while the other is fixed to 0 or 1, as required. Then the *pLayer* permutes the *sBoxLayer* output and forwards to the next round. Assume that the *pLayer* of the SPN-based block cipher permutes j^{th} bit to $p(j)^{th}$ bit such that $j, p(j) \in [n]$.

For example, consider implementing a 2-input NAND gate using a PRESENT S-Box in the i^{th} round, where input signals (A and B) are directed to the third and fourth S-Box inputs. The resulting output is obtained from the second output, as illustrated in Figure 8. The transmission of this output signal to the $p(j+1)^{th}$ bit of the $(i+1)^{th}$ *sBoxLayer* is affected by the $(i+1)^{th}$ *addRoundKey* layer, influencing the nature of the implemented logic gate at the i^{th} round. The pseudorandom round key value at the $p(j+1)^{th}$ position of the $(i+1)^{th}$ round dictates the outcome: if it is 0 (Figure 8(a)), output signal remains unchanged; if it is 1 (Figure 8(b)), the signal is complemented before reaching the $(i+1)^{th}$ *sBoxLayer*. To address this, we produce the complement of the intended logic gate (AND instead of NAND in Figure 8) with the S-Box when the round key value at the $p(j+1)^{th}$ position of the $(i+1)^{th}$ round is 1. As a result, the complement (O') of the desired output is produced from the i^{th} *sBoxLayer*, and then gets complemented by the $(i+1)^{th}$ *addRoundKey* layer to produce the desired signal at the input of $(i+1)^{th}$ *sBoxLayer*.

Upon determining the gate functionality (original or complement) to be realized with the current

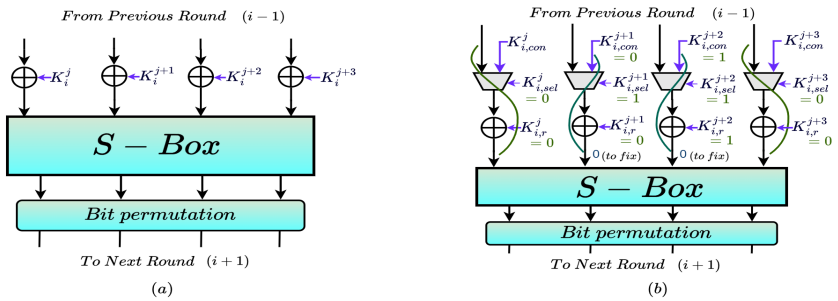


Figure 7: (a) Structure of one round of original SPN-based block cipher. (b) Modified SPN-based block cipher round with inserted MUX layer, used for setting desired bits at S-Box inputs (where $K_{i,sel} = 1$) and passing bits from the previous round (where $K_{i,sel} = 0$). Here, i and j denote current round and intermediate state bit, respectively.

S-Box, the previous layer outputs (A, B) is transmitted to its third and fourth input positions by setting the MUX select line values at positions $j+2, j+3$ to 0. The configuration key values at these positions ($(j+2)^{th}$ and $(j+3)^{th}$) are randomly assigned, as they are not required at those positions. The corresponding MUX select line values at the positions $j, j+1$ are set to 1 to fix the inputs to specific values as required. Subsequently, the MUX configuration key values at these positions are determined based on the round key value. In Figure 8(a), for instance, to fix a 0 at the j^{th} input of S-Box with the corresponding round key value as 0, the configuration key at j^{th} bit is set to 0. Conversely, at position $(j+1)$, the configuration key is set to 1 since the round value is 1 and the desired bit to be set is 0.

Each select line of all n MUXs in the MUX layer used to regulate the functionality of the augmented block cipher and these select keys are denoted with $\{K_{i,sel}\}_{i\in[d]}$ for all d numbers of MUX layers in the block cipher. Also, to configure the behavior of an S-Box as a logic/buffer gate, the binary values required to enter the MUX layer to fix specific inputs of the S-Box, constitute another key. We refer to these configuration keys with $\{K_{i,con}\}_{i\in[d]}$, for all d rounds in the block cipher. Note that the round keys, denoted with $\{K_{i,r}\}_{i\in[d]}$ are pseudorandom. They are secret, and they play an important role in embedding the desired circuit with proper select and configuration keys.

After realizing a circuit through a block cipher using the aforementioned method, the circuit inputs must be supplied at specific positions among the first round's n available cipher input bits. These positions may vary depending upon the embedding and the choice of S-Boxes in the first *sBoxLayer*. We denote the list of these specific input bit positions with IB, and |IB| is the primary input size of the circuit to be obfuscated. Likewise, the circuit outputs are also obtained at specific positions denoted with OB, and similarly |OB| is equal to the primary output size of the circuit. For example, consider a circuit consisting of only one NAND gate. When it is embedded using one round of augmented PRESENT, assuming the first S-Box in the first *sBoxLayer* realizes the NAND gate with x_1 and x_2 fixed to 0; then IB is $\{3, 4\}$ and OB becomes a singleton list $\{2\}$, since $y_2 = \mathsf{NAND}(x_3, x_4)$, i.e., the inputs are given to the last two input bits and the output is obtained through the second output bit of the first S-Box.

In a typical block cipher, the round keys are generated by a key scheduling algorithm that utilizes the cipher's secret key as input. In addition to the round keys ($\{K_{i,r}\}_{i\in[d]}$), HardCODE requires the use of two additional keys, i.e., select keys ($\{K_{i,sel}\}_{i\in[d]}$) and configuration keys ($\{K_{i,con}\}_{i\in[d]}$). The total size of the keys, including round keys, select keys, and configuration keys, increases linearly with the number of rounds d (which eventually depends on the number of

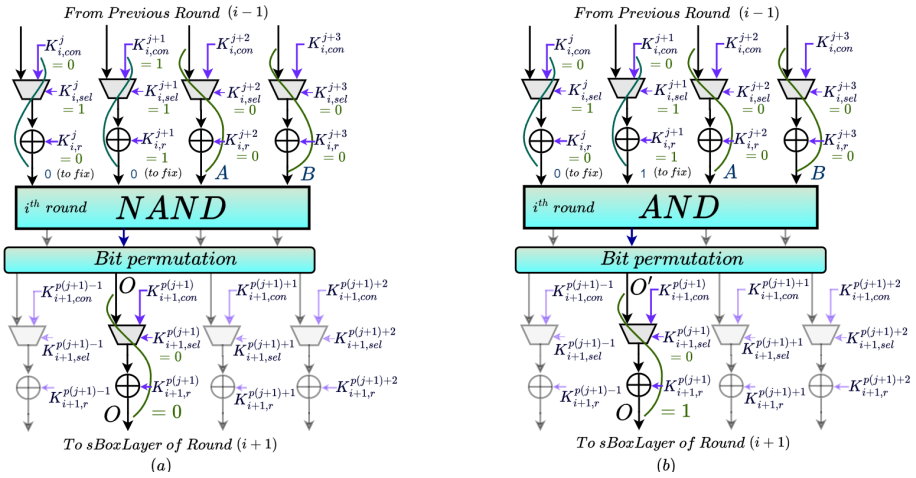


Figure 8: An instance of realizing a two input NAND gate by implementing NAND (G) or AND (G') at the i^{th} round S-Box based on *addRoundKey* ($K^{p(j+1)}_{i+1,r}$) bit value (0 in (a) or 1 in (b)) at $(i+1)^{th}$ round. $p(j+1)$ denotes the permutated bit position of $(j+1)^{th}$ bit from which the output (O of G and O' of G') is produced. The NAND and AND have identical input-output bit positions.

topological levels of the input circuit) needed to obfuscate the circuit. However, a cryptographic construction, whose secret key size varies depending upon input (circuit in our case), is not feasible and therefore, this raises the following question:

Can we fix the secret key size in our construction and derive the other three secret components, namely, the round keys, select keys, and configuration keys from the fixed-size secret key, independent of the circuit?

We affirmatively address this question by considering only a fixed-size secret key for HardCODE construction and refer to it as the master secret key (msk). The secret msk is stored inside a tamper-proof memory e.g., HSM [SZR+23,KSS23,Das23]. A PRG, with msk as its seed, generates a bit sequence of length $(n \times 3d)$, denoted with $R_{1,1}||\cdots R_{1,d}||R_{2,1}||\cdots R_{2,d}||R_{3,1}||\cdots R_{3,d}$, such that each $R_{i,j}$ with $i \in [3], j \in [d]$ is a n-bit pseudorandom string. The last nd bits of PRG output are used as round keys of d cipher rounds. Hence, we write $K^j_{i,r} = R^j_{3,i}, \forall i \in [d], j \in [n]$.

This simple yet effective solution solves the aforementioned problem. It alleviates the requirement of treating select and configuration keys for each round as part of the secret keys. Consequently, we need to store msk securely as the only secret information in output hardware module of HardCODE, regardless of the underlying circuit and the number of cipher rounds used in the embedding.

Till this point, we describe how to embed a given circuit in an SPN-based block cipher. We note that the HardCODE structure aligns with HiO definition as specified in Section 2, since it turns each circuit into a hardware module (M_f) having three components: (i) the SPN-embedding of the given circuit, (ii) a PRG circuit that uses msk as its seed, and (iii) a tamper-proof hardware to store msk. This hardware module is accompanied with three sets of keys, namely round, select, and configuration keys (let us denote them together with K_f for now) and some auxiliary information (I_{aux}) in the form of IB and OB.

Next, we discuss an interesting property of HardCODE. Assuming two functionally equivalent

circuits ckt_0 and ckt_1, when input to HardCODE, it produces $(M_f^0, K_f^0, I_{aux}^0)$ and $(M_f^1, K_f^1, I_{aux}^1)$ as its outputs, respectively. We show that the SPN-embedding components of M_f^0 and M_f^1 can be distinguished in the presence of the corresponding msk's if and only if (K_f^0, I_{aux}^0) and (K_f^1, I_{aux}^1) can be distinguished from each other. Simply, the two components (K_f, I_{aux}) of HardCODE output together reveal the SPN-embedding component of M_f entirely and vice versa. Here, SPN-embedding implies the following information (i) the subset of S-Boxes in each of the d rounds that act as gates/buffers and (ii) the specific logical functions (i.e., which gates) they implement. These actually imply a representation of the underlying circuit.

Lemma 1. *If there exists a PPT adversary $\mathcal{A}$ that is able to distinguish between (K_f^0, I_{aux}^0) and (K_f^1, I_{aux}^1) with ϵ advantage, we can construct another PPT adversary $\mathcal{A}'$ that will be able to distinguish between SPN-embedding components of M_f^0 and M_f^1 with advantage ϵ. Likewise, if there exists a PPT adversary $\mathcal{A}$ that is able to distinguish between the SPN-embedding components of M_f^0 and M_f^1 with advantage ϵ, we can construct another PPT adversary $\mathcal{A}'$ that will be able to distinguish between (K_f^0, I_{aux}^0) and (K_f^1, I_{aux}^1) with advantage ϵ.*

Proof. The equivalence between the aforementioned components is straightforward implication from HardCODE construction. Knowledge of the SPN-embedding specification together with the master key msk allows one to seed the PRG with msk, derive the round keys from its output, and then compute the necessary select key and configuration keys for any chosen IB and OB. This is a randomized means of producing (K_f, I_{aux}) for given SPN-embedding and msk of M_f. Conversely, knowing only (K_f, I_{aux}) enables a deterministic reconstruction of the exact SPN-embedding, without any requirement for the msk. □

Now, we have established the necessity of rendering the K_f components of two HardCODE outputs for two functionally equivalent circuits indistinguishable from each other in order to maintain the indistinguishability of the corresponding SPN-embeddings. We achieve this with the masking technique. In particular, we mask the select keys $(\{K_{i,sel}\}_{i\in[d]})$ with the first nd bits of the PRG output sequence, and the configuration keys $(\{K_{i,sel}\}_{i\in[d]})$ with the next nd bits, as shown in the following,

$$K_{i,msel}^j = R_{1,i}^j \oplus K_{i,sel}^j, \quad K_{i,mcon}^j = R_{2,i}^j \oplus K_{i,con}^j, \quad \forall i \in [d], \forall j \in [n] \tag{1}$$

We declare these masked select $(\{K_{i,msel}\}_{i\in[d]})$ and masked configuration keys $(\{K_{i,mcon}\}_{i\in[d]})$ as publicly available keys which essentially hide the original keys from a PPT adversary as they are XORed with pseudorandom bits. Note that we do not need to mask the round keys since they are already pseudorandom (the last nd bits of PRG output) in nature, and they are not publicly available. *Note that the K_f component of* HardCODE *does not directly include the round keys, select keys, or configuration keys; instead, it comprises only the masked versions of select and configuration keys. Furthermore, msk must be confined within tamper-proof hardware so that an adversary can not distinguish between the two sets of (K_f, I_{aux}) of functionally equivalent circuits in presence of msk. Under the HiO adversarial model (ref. Section 2), probing of the PRG's output bits is forbidden, preventing any external entity from accessing those values.* .

A traditional key scheduling algorithm of a block cipher derives each round key from the previous round key or the master key; hence, there is a considerable dependency between two consecutive round keys. This dependency is also required to enable decryption of the ciphertext using the same master key of the block cipher. However, we are only interested in realizing a circuit through encryption of a block cipher, hence, we require the round keys $\{K_{i,r}\}_{i\in[d]}$ to be

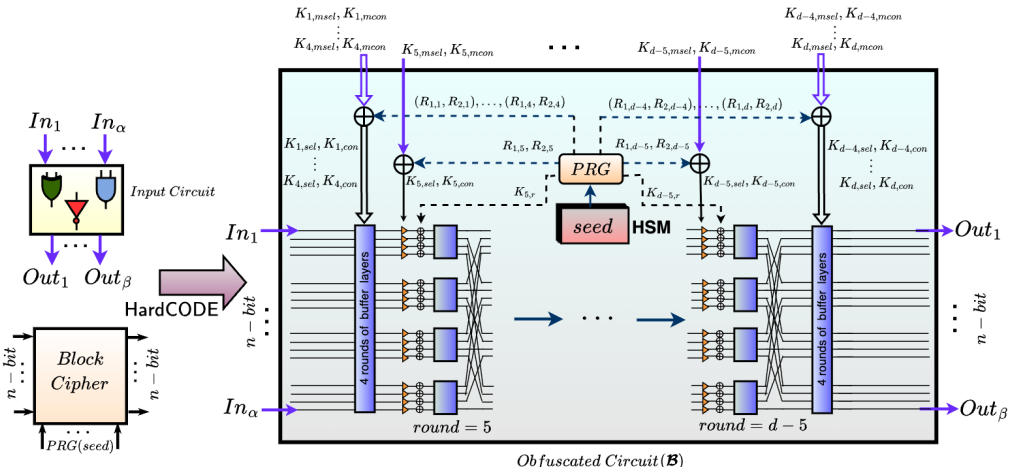


Figure 9: An overview of HardCODE. It takes input a circuit and a block cipher to produce the obfuscated circuit (emulator box $\mathcal{B}$) and the masked publicly available keys ($K_{i,msel}$, $K_{i,mcon}$) for each round. These keys are unmasked (XORed) with PRG generated keys ($R_{1,i}$, $R_{2,i}$) to produce the actual keys ($K_{i,sel}$, $K_{i,con}$) of the MUX Layer. $\mathcal{B}$ contains a PRG circuit with the seed msk protected in secure memory module (HSM) to produce round keys ($K_{i,r}$). The black arrows signify the inputs that must be supplied to simulate $\mathcal{B}$. In_1 to In_α are circuit inputs. Out_1 to Out_β are circuit outputs. $|\mathsf{IB}| = \alpha$ and $|\mathsf{OB}| = \beta$. The dotted arrows is secret PRG outputs to the required positions.

pseudorandom. We also need a total of $2dn$-bit pseudorandom string ($\{R_{1,i}\}_{i\in[d]}$, $\{R_{2,i}\}_{i\in[d]}$) to mask $\{K_{i,sel}\}_{i\in[d]}$ and $\{K_{i,con}\}_{i\in[d]}$, and prevent any information leakage from them. This justifies the use of PRG in generating a total of $3dn$-bit length pseudorandom string, which are partially used as round keys of underlying block cipher $\mathcal{E}$ and partially used as masks for hiding information about the select keys and configuration keys. Finally, HardCODE returns two publicly available keys, namely masked select keys and masked configuration keys (the K_f component) along with the lists IB and OB (the I_{aux} component) to identify the specific bit positions among the n available input and output bits where the circuit's input and outputs are obtained, respectively and an emulator box $\mathcal{B}$ (the M_f component) as shown in Figure 9. Apart from d rounds of MUX-augmented SPN-based block cipher, $\mathcal{B}$ contains (i) a tamper-proof memory to securely store the master secret key msk, (ii) A PRG circuit that takes msk as its seed and derives the pseudorandom string ($\{R_{i,j}\}_{i\in[3],j\in[d]}$), and (iii) a mechanism to unmask $\{K_{i,msel}\}_{i\in[d]}$ and $\{K_{i,mcon}\}_{i\in[d]}$, into $\{K_{i,sel}\}_{i\in[d]}$ and $\{K_{i,con}\}_{i\in[d]}$, respectively. The emulator box $\mathcal{B}$, in presence of $\{K_{i,msel}\}_{i\in[d]}$ and $\{K_{i,mcon}\}_{i\in[d]}$, can emulate the functionality of the targeted circuit. The output of the circuit is obtained at positions specified by OB, upon providing the inputs at positions specified in IB. HardCODE achieves indistinguishability obfuscation since distinguishing between publicly available key components $K_f = (\{K_{i,msel}\}_{i\in[d]}, \{K_{i,mcon}\}_{i\in[d]})$ of two HardCODE-obfuscated functionally equivalent circuits is equivalent to distinguishing between two ciphertexts of a symmetric key encryption (SKE) scheme in presence of an eavesdropper (ref. Section 3.3 for detailed explanation). Later, Section 4 shows that we can reduce the key recovery problem from augmented block cipher into a known plaintext attack scenario to an instance of the 3-SAT problem, which is hard to solve using SAT solver. Hence, we conclude that the hardware module M_f (i.e., emulator box $\mathcal{B}$) does not leak the internal secret keys, namely round keys, select keys, and configuration keys.

Algorithm 1 $\mathsf{HardCODE}_{\mathcal{E}}.\mathsf{Obfuscate}(\mathsf{ckt}, pp)$

Require: Circuit ckt and Public parameters $pp = (\mathcal{E}, d, \mathsf{G})$. Here, $\mathcal{E} : \mathcal{P} \times \mathcal{K} \rightarrow \mathcal{C}$ is a block cipher, d denotes the number of cipher rounds, $\mathsf{G} : \{0,1\}^{\lambda} \rightarrow \{0,1\}^{3dn}$ is a secure PRG.

1: Choose $msk \xleftarrow{\$} \{0,1\}^{\lambda}$ ▷ Sampling the master secret key
2: $R_{1,1}||\cdots||R_{1,d}||R_{2,1}||\cdots||R_{2,d}||R_{3,1}||\cdots||R_{3,d} \leftarrow \mathsf{G}(msk)$ ▷ String of length $3dn$ bits
3: $K^j_{i,r} \leftarrow R^j_{3,i}, \forall i \in [d], j \in [n]$ ▷ Last dn bits of PRG output form the round keys of d cipher rounds
4: $\mathcal{E}', (\{K^j_{i,sel}\}_{i\in[d],j\in[n]}, \{K^j_{i,con}\}_{i\in[d],j\in[n]}, \mathsf{IB}, \mathsf{OB} \leftarrow \textsc{Embed}(\mathcal{E}, \mathsf{ckt}, \{K^j_{i,r}\}_{i\in[d],j\in[n]})$
5: $K^j_{i,msel} \leftarrow R^j_{1,i} \oplus K^j_{i,sel}, \forall i \in [d], j \in [n]$ ▷ Publicly available key
6: $K^j_{i,mcon} \leftarrow R^j_{2,i} \oplus K^j_{i,con}, \forall i \in [d], j \in [n]$ ▷ Publicly available key
7: Create an emulator box $\boldsymbol{\mathcal{B}}$, which encloses a tamper-proof memory containing msk as the seed of PRG, the augmented cipher $\mathcal{E}'$ and a PRG circuit to generate round keys $\{K^j_{i,r}\}_{i\in[d],j\in[n]}$ of $\mathcal{E}'$ and masks $\{R^j_{1,i}\}_{i\in[d],j\in[n]}$, $\{R^j_{2,i}\}_{i\in[d],j\in[n]}$ to retrieve $\{K^j_{i,sel}\}_{i\in[d],j\in[n]}$, $\{K^j_{i,com}\}_{i\in[d],j\in[n]}$ back before feeding to $\mathcal{E}'$ (ref. Figure 9).
8: **return** $\langle M_f, K_f, I_{aux}\rangle = \left\langle \boldsymbol{\mathcal{B}}, \left(\{K^j_{i,msel}\}_{i\in[d],j\in[n]}, \{K^j_{i,mcon}\}_{i\in[d],j\in[n]}\right), \left(\mathsf{IB}, \mathsf{OB}\right)\right\rangle$

Algorithm 2 $\mathsf{HardCODE}_{\mathcal{E}}.\mathsf{Evaluate}(\langle M_f, K_f, I_{aux}\rangle, x \in \{0,1\}^{\alpha})$

Require: $\boldsymbol{\mathcal{B}}$: hardware module with embedded PRG and SPN-embedding, IB: input-port mapping, OB: output-port mapping, K_f: masked select and configuration keys.
Ensure: $y \in \{0,1\}^{\beta}$: obfuscated evaluation output.
1: Supply $x_1, \ldots, x_{\alpha}$ to $\boldsymbol{\mathcal{B}}$'s input ports according to IB.
2: Load (K_f) (masked select & configuration keys) into $\boldsymbol{\mathcal{B}}$.
3: Activate $\boldsymbol{\mathcal{B}}$:

a. PRG uses msk (in tamper-proof memory) as seed to generate $3dn$ bits.

b. Split PRG output:

- Last dn bits $\rightarrow$ round keys for SPN.
- First $2dn$ bits $\oplus$ masked keys in $K_f \rightarrow$ unmask select and configuration keys.

c. Connect unmasked select, configuration, and PRG-derived round keys into the d rounds of SPN-embedding.

4: Collect $y_1, \ldots, y_{\beta}$ from $\boldsymbol{\mathcal{B}}$'s output ports according to OB.
5: **return** $y \in \{0,1\}^{\beta}$ as the output.

3.2 Formal Description of HardCODE

We now formally describe our hardware-based circuit obfuscator HardCODE assuming the existence of a block cipher $\mathcal{E} : \mathcal{P} \times \mathcal{K} \rightarrow \mathcal{C}$, where $\mathcal{P} = \mathcal{C} = \{0,1\}^n$ are plaintext and ciphertext space and $\mathcal{K} = \{0,1\}^{\lambda}$ is the secret key space of $\mathcal{E}$. Consequently, each *sBoxLayer* will have n inputs and n outputs. We assume $s \times s$ S-Boxes of $\mathcal{E}$ with $\lceil \frac{n}{s} \rceil$ number of S-Boxes per *sBoxLayer*. Naturally, the length of the round key in each round is n. We denote our hardware-based indistinguishability obfuscator with $\mathsf{HardCODE}_{\mathcal{E}}$ when the underlying block cipher is instantiated with $\mathcal{E}$. Note that, $\mathcal{E}$ can be instantiated with any SPN-based block cipher. In addition, we assume that d is the number of rounds required to successfully embed ckt in $\mathcal{E}$ using $\mathsf{HardCODE}_{\mathcal{E}}$. We argue that a successful embedding of a circuit incorporates hiding information leakage from the input positions (IB) and output positions (OB) as well. So, the number of rounds (d) includes all the rounds required for (i) hiding input bit positions with a few initial buffer layers, (ii) embedding all the gates of the circuits, and (iii) hiding output bit positions with a few final buffer layers.

As detailed in Section 2, our HardCODE, construction realizes $Hi\mathcal{O}$, through two PPT algorithms, namely HardCODE.Obfuscate and HardCODE.Evaluate. Algorithm 1 and Algorithm 2 describe the steps involved in HardCODE.Obfuscate and HardCODE.Evaluate, respectively. Moreover, Algorithm 3 defines the internal subroutine Embed, which is invoked by Algorithm 1. Now, we proceed step-by-step to comprehend the method for obfuscating any circuit using HardCODE.

Obfuscating a circuit. According to Algorithm 1, the obfuscating method involves the

following steps: (i) generation of λ-bit uniformly random master secret key msk, (ii) utilizing PRG G with msk as its seed and expansion to a $3dn$-bit sequence, in which the first dn-bits of the PRG output is used as mask for select keys, the second dn-bits are used as mask for configuration keys, and the last dn-bits are used as round keys for the augmented block cipher, (iii) embedding the circuit in the augmented version of block cipher ($\mathcal{E}$) using n-bit round keys $\{K_{i,r}\}_{i\in[d]}$ for each round to produce two different sets of keys, n-bit select keys $\{K_{i,sel}\}_{i\in[d]}$ and, n-bit configuration keys $\{K_{i,con}\}_{i\in[d]}$ for all the rounds of the augmented cipher, (iv) masking the select keys using the first nd bits of PRG output and the configuration keys using the next nd bits as described in steps 5 and 6. Thus, we obtain d rounds of n-bit masked select keys $\{K_{i,msel}\}_{i\in[d]}$, n-bit masked configuration keys $\{K_{i,mcon}\}_{i\in[d]}$. Finally, an emulator box $\boldsymbol{\mathcal{B}}$ is created as a hardware module M_f with all necessary components to emulate the behavior of ckt (for illustration ref. Figure 9). The emulator box $\boldsymbol{\mathcal{B}}$ with $(\mathsf{IB}, \mathsf{OB})$ as auxiliary information I_{aux}, masked select keys $\{K_{i,msel}\}_{i\in[d]}$, and masked configuration keys $\{K_{i,mcon}\}_{i\in[d]}$ as publicly available keys K_f are released as $\mathsf{HardCODE.Obfuscate}$ (Algorithm 1) output. Note that we assume the input ckt to be represented as a *Directed Acyclic Graph (DAG)* $\mathcal{D}_{\mathsf{ckt}}$ where each logic gate of ckt corresponds to vertex of $\mathcal{D}_{\mathsf{ckt}}$ and each interconnection from output of a logic gate to the input of another in ckt corresponds to a directed edge of $\mathcal{D}_{\mathsf{ckt}}$.

The steps in Algorithm 1 have been discussed and are readily apparent except Step 3, which introduces an embedding function to embed ckt in the augmented block cipher. Algorithm 3 discusses this embedding process in detail.

Embedding a circuit using an SPN-based block cipher. Algorithm 3 presents an overview of embedding an arbitrary circuit ckt in $\mathcal{E}$ with the desired width. By desired width, we imply that the number of S-Boxes in *sBoxLayer* of $\mathcal{E}$ should be greater than or equal to the maximum number of gates across all topological levels of ckt. Algorithm 3 takes as input the block cipher $\mathcal{E}$, the circuit ckt and all the round keys $\{K^j_{i,r}\}_{i\in[d],j\in[n]}$ such that $\mathcal{E}$, with its inherent components such as the *addRoundKey*, *sBoxLayer*, *pLayer*, and an additional MUX layer, is used for realizing the given ckt. Using the embedding method in Algorithm 3, logic and buffer gates are realized using S-Boxes, and their interconnections are realized utilizing the *pLayer*. At first, the input bit positions (IB) are assigned with the primary inputs of ckt in the first round of embedding. Then, a sufficient number of buffer layers are added to obscure the path of ckt input signals from the positions specified in IB to the actual S-Boxes, realizing the logic gates. This obscuring is possible because of the full diffusion property of SPN-based block cipher (ref. Lemma 3). Algorithm 3 maximizes the number of gates at a topological level that can be realized per round of the block cipher. For example, let us consider a circuit with a topological level consisting of four logic gates and a cipher with at least four S-Boxes in its *sBoxLayer*. Our $\mathsf{HardCODE}$ aims to realize all four gates by utilizing S-Boxes exclusively from a single *sBoxLayer*. Further, the signal for gates of the same topological level that can not be realized in the same round is routed to consecutive rounds using buffer gate constructions of S-Boxes. Considering the above example, if the realization of all four gates is impeded due to routing constraints within the *pLayer*, an alternative approach is adopted. Now, it tries to realize three gates within one *sBoxLayer* while routing the signal associated with the remaining gate into the subsequent *sBoxLayer*, deploying buffer S-Boxes. This process continues till all four gates are placed. This algorithm iterates until all the gates in the circuit have been placed. Upon successful embedding of ckt, some more buffer layers are added to make the total number of cipher rounds equal to d. Following this, all the output bit positions (OB) corresponding to ckt primary output are marked. The inclusion of additional buffer layers plays a crucial role in mitigating the risk of information leakage stemming from IB and OB. We will delve deeper into this aspect while discussing the necessary number of cipher rounds for the embedding process.

Algorithm 3 Embedding a circuit in a block cipher

Require: *sBoxLayer* and *pLayer* mapping of $\mathcal{E}$ (assume that *pLayer* of $\mathcal{E}$ permutes the j^{th} position in i^{th} round to $p(j)^{th}$ position in the $(i+1)^{th}$ round); circuit ckt represented as a DAG; S-Box input-output bit configuration and positions for constructing logic gates and buffers; n-bit round keys for all d rounds $\{K^j_{i,r}\}_{i\in[d],j\in[n]}$

function EMBED($\mathcal{E}$, ckt, $\{K^j_{i,r}\}_{i\in[d],j\in[n]}$)
  $i \leftarrow 1$ ▷ counter variable for block cipher round
  $IB \leftarrow$ primary input bit positions of ckt in the first round
  Add a few (say d') initial cipher rounds as buffer ▷ Only for the first round, two round keys are considered. For other rounds, depending on $K_{i+1,r}, i \in [d'-1]$, the buffer layer S-Boxes realize either buffer or complementary NOT gates.
  $i \leftarrow i + d'$
  while (!all topological levels are embedded) **do**
    while (!all gates in the topological level are placed) **do**
      $i \leftarrow i + 1$
      ▷ For a target S-Box let j denote the positions through which input signals are passed and j' denote positions through which configuration bits are set.
      if $K^{p(j)}_{i+1,r} == 1$ **then** ▷ For realizing gates or their complements
        Realize complements of desired gates/buffers from target S-Box in i^{th} round using *sBoxLayer* mapping
      else
        Realize desired gates/buffers from target S-Box in i^{th} round using *sBoxLayer*
      $K^j_{i,con} \xleftarrow{\$} \{0,1\}$ ▷ Don't care configuration bits
      if $K^{j'}_{i,r} == 1$ **then** ▷ For setting configuration bits
        $K^{j'}_{i,con} \leftarrow$ Complement of input bit configuration of S-Boxes to be set in i^{th} round for gate/buffer construction
      else
        $K^{j'}_{i,con} \leftarrow$ Input bit configuration of S-Boxes to be set in i^{th} round for gate/buffer construction
      $K^j_{i,sel} \leftarrow 0, K^{j'}_{i,sel} \leftarrow 1$ ▷ MUX select line keys for S-Box realizing gates/buffer
      $(K^j_{i,sel}, K^j_{i,con}) \xleftarrow{\$} \{0,1\}$ ▷ $\forall j$ corresponding to un-utilized S-Boxes in a round
  Add buffer rounds to make a total of $(d-1)$ cipher rounds by realizing either buffer or NOT gate based on round key bit of the next round. Finally, realize a single buffer layer at the last round (with no *addRoundKey* layer ahead)
  $OB \leftarrow$ primary output bit positions of ckt in the final round
  return $\mathcal{E}'$, $(K^j_{i,sel}, K^j_{i,con})_{i\in[d],j\in[n]}$, IB, OB ▷ $\mathcal{E}'$ is the augmented block cipher

Finally, Algorithm 3 returns the n-bit keys $K_{i,sel}$ and $K_{i,con}$ based on randomly chosen $K_{i,r}$ for each of the i^{th} round (where $i \in [d]$) along with IB and OB. Next, we discuss the role of $K_{i,r}$ in realizing a desired circuit.

Role of PRG-generated Round Keys in Embedding Circuits. The last nd bits of PRG output are utilized as n-bit round keys of all d rounds of the embedding. Each round key bit, being pseudorandom, can take any binary value for a specific position j in i^{th} round, where $i \in [d], j \in [n]$. However, this introduces a concern. For instance, between a particular S-Box at the $(i-1)^{th}$ round configured to realize the logic gate G and the output routed to the $p(j)^{th}$ bit at the i^{th} round S-Box, (where $p(j)$ denotes the permuted position of the $(i-1)^{th}$ round j^{th} bit in i^{th} round), there lies the *addRoundKey* of the i^{th} round. If the $p(j)^{th}$ bit of *addRoundKey* at the i^{th} round is 0, the output of G propagates to i^{th} round S-Box input from $(i-1)^{th}$ round S-Box output without alterations. However, if it is 1, the complement of G (let G') reaches the input of the i^{th} round S-Box, undesirably altering the functionality of the embedded circuit.

We propose the following embedding strategy to address this situation: we examine the i^{th} round *addRoundKey* to determine the functionality for the $(i-1)^{th}$ round S-Boxes. If the $p(j)^{th}$ bit of the *addRoundKey* in the i^{th} round is 0, we configure the $(i-1)^{th}$ round S-Boxes to realize G, producing the desired output at the $p(j)^{th}$ bit of the i^{th} round S- Box. Conversely, if it is 1, we configure the $(i-1)^{th}$ round S-Boxes to the complement of G (i.e., G'). This ensures that the output of G' at the $(i-1)^{th}$ round is neutralized by XORing with 1 in the i^{th} round *addRoundKey* bit, thereby achieving the desired functionality of G at the S-Box input

of the i^{th} round (refer lines 10 - 13, Algorithm 2). We present an illustrative example of the aforementioned situation in Figure 8. A detailed discussion on ensuring consistent S-Box input-output configurations and mitigating dynamic functionality shifts due to the PRG-generated round keys are provided in Appendix D.

After embedding ckt in $\mathcal{E}$, the augmented block cipher, with an additional MUX layer and two sets of keys (select keys and configuration keys), is denoted with $\mathcal{E}'$. Note that, $\mathcal{E}'$ emulates the behavior of $\mathcal{E}$ if (i) all the bits in select key $\{K_{i,sel}\}_{i\in[d]}$ are fixed to 0, consequently making all bit values of $\{K_{i,con}\}_{i\in[d]}$ "don't care" and, (ii) round keys are generated from msk by $\mathcal{E}$'s key scheduling algorithm instead of the PRG (for details on how we use this property to argue security, refer to Appendix F). However, we must assign appropriate values to select and configuration keys to ensure the obfuscated circuit's correctness.

Observation on Circuit Space. Here, we discuss the feasibility of embedding an arbitrary circuit into an SPN-based block cipher utilizing HardCODE. Specifically, we discuss the bounds on the number of gates at a topological level that could be embedded using a block cipher.

Utilizing block ciphers for realizing an arbitrary circuit in our HardCODE construction presents an initial challenge. The maximum number of gates that can be realized in a single round of a block cipher is given in the following lemma.

Lemma 2. *The maximum number of logic gates in a topological level that could be realized in one sBoxLayer is equal to the number of S-Boxes present in the sBoxLayer of the block cipher.*

Proof. We introduce an approach to implement an S-Box as a *single* logic gate by constraining specific input bits to fixed binary values in HardCODE. Consequently, the maximum count of gates that can be realized within a single *sBoxLayer* is equal to the number of S-Boxes present in that particular *sBoxLayer*. □

Thus, a proper choice of SPN-based block cipher is necessary for HardCODE construction. A straightforward solution is to choose an underlying block cipher with total S-Boxes in *sBoxLayer* of each round being greater than or equal to the maximum number of gates occurring in any topological level of the circuit. This is the only constraint of HardCODE for embedding any arbitrary circuits. The desired circuit can be realized utilizing block cipher rounds, which is polynomial in the circuit's total number of topological levels (ℓ). The best-case scenario of embedding arises when every round of block cipher can embed a topological level of the desired circuit, abiding by all properties of HardCODE.

However, since we keep the *pLayer* intact, a situation may arise in which multiple signals try to utilize a specific S-Box in *sBoxLayer* of a particular round. This situation is known as the *collision* of signals. An example of *collision*, a discussion to circumvent the collision problem, and other constraints related to embedding are detailed in Appendix C.

Observation on Number of Rounds. In our proposed HardCODE framework, we introduce a method for embedding a circuit in a topological level-wise fashion. As previously demonstrated, our approach necessitates a number of cipher rounds upper bounded by the total number of logic gates in a topological level, using an SPN-based block cipher to accommodate each topological level. Consequently, to embed the entire circuit successfully, we require a polynomial number of cipher rounds with respect to the circuit topological level. Within HardCODE we employ an SPN-based block cipher incorporating an additional MUX layer for circuit embedding purposes. In HardCODE, we can augment the number of cipher rounds as needed to accommodate the

embedding of the target circuit, provided that the PRG can effectively generate the round keys and the requisite mask for select and configuration keys of all d rounds.

HardCODE.Evaluate requires $I_{aux} = (\mathsf{IB}, \mathsf{OB})$ to ensure correct functioning of the output hardware module $\boldsymbol{\mathcal{B}}$ using publicly available masked select and configuration keys, denoted together as K_f. However, a security concern arises when additional buffer rounds of the augmented cipher are not appended before starting to embed the first topological level and after embedding all gates within ckt. The reason is that from IB and OB, adversaries could potentially extract information about the circuit embeddings by reverse-engineering the fixed and uniform structure of the augmented block cipher. Nevertheless, by incorporating an adequate number of additional buffer rounds that guarantee full diffusion of state bit, the adversary's ability to deduce the actual S-Boxes responsible for implementing the gates in the first and the final topological level of ckt is diminished. The precise number of buffer rounds required is determined by the diffusion property of the *pLayer* [PDJ+19], as mentioned in the following subsection. For example, when employing HardCODE with the PRESENT cipher, it is essential to include three or four additional buffer rounds at the beginning and end to ensure complete diffusion. It is worth noting that our Algorithm 3 inherently encompasses these additional buffer rounds during the circuit embedding process. Thus, d signifies the total number of rounds, including the standard and additional buffer rounds.

Significantly, this augmentation of rounds does not hamper the security of HardCODE. This is because the length of the computationally hard instances for the Boolean satisfiability problem (SAT), as represented in conjunctive normal form (CNF) formulas, increases linearly with the augmentation of cipher rounds.

In summary, our approach in HardCODE enables a systematic and secure embedding of circuits through a topological level-wise technique, leveraging an SPN-based block cipher with a MUX layer. The reliance on a PRG for generating masks permits the flexible augmentation of cipher rounds to accommodate desired circuits without compromising security, as the SAT-hardness of CNF formula instances scales linearly with the increase in cipher rounds.

3.2.1 Proof of Correctness.

Here, we prove that our proposed instantiation of $Hi\mathcal{O}$, namely HardCODE, satisfies the correctness property as described in 2. Our argument has two steps as follows.

1. The Embed function invoked from HardCODE.Obfuscate ensures the functional correctness of any circuit ckt during its embedding into the SPN layers of $\mathcal{E}$ by choosing appropriate select and configuration keys.

2. Now, the correctness of HardCODE.Evaluation is ensured by the fact that the select and configuration keys needed for the desired execution path in SPN layers of $\mathcal{E}$ are always retrieved through unmasking of K_f with the output of PRG, present in the hardware module $\boldsymbol{\mathcal{B}}$.

Hence, by the construction of HardCODE, it is straightforward to see that the obfuscated circuit will retain the functionality of the original circuit.

Proof of Polynomial Slowdown. As mentioned in Definition 1, the polynomial slowdown is a required property of indistinguishability obfuscator. The following lemma shows that our proposed HardCODE indeed achieves this.

Lemma 3. *A logic circuit* ckt *can be embedded utilizing d rounds of an SPN-based block cipher such that d is a polynomial with respect to* ℓ*, where* ℓ *is the number of topological levels of the circuit.*

Proof. A logic circuit ckt, represented using a DAG $\mathcal{D}_{\mathsf{ckt}}$, can be partitioned into topological levels. Let $T_1, T_2, \cdots, T_\ell$ be the topological levels of the $\mathcal{D}_{\mathsf{ckt}}$, and the embedding function maps T_i using d_i rounds of an SPN-based block cipher, such that $\sum_{i=1}^{\ell} d_i = d$. In $\mathsf{HardCODE}$, we embed ckt in an SPN-based block cipher in a topological level-wise manner. The S-Boxes are constructed as logic gates, whereas the *pLayer* acts as the interconnecting circuit wires.

The following properties of a *pLayer* [PDJ+19] are required for supporting an optimal diffusion property of an SPN-based block cipher with $\lceil \frac{n}{s} \rceil$ number of $s \times s$ S-Boxes in each round.

- The input of an S-Box in an intermediate round i comprises output bits from s different S-Boxes in round $i-1$, where $i \in [2, d]$.
- The output of an S-Box in round i is distributed across the inputs of s different S-Boxes in round $i+1$, where $i \in [d-1]$.
- The output of the j^{th} S-Box $\{sj, sj+1, ..., sj+(s-1)\}$ in round i entirely constitutes the following inputs $\{j, j+\lceil \frac{n}{s} \rceil, ..., j+(s-1)\lceil \frac{n}{s} \rceil\}$ of S-Boxes in round $i+1$, where $i \in [d-1]$ and $j \in \{0, 1, ...(\lceil \frac{n}{s} \rceil - 1)\}$.
- Full diffusion occurs in $\zeta = \lceil \log_s (s^2 \cdot \lceil \frac{n}{s} \rceil) \rceil$ rounds. Note that for any choices of n, s with $s \leq \lceil \frac{n}{s} \rceil \leq s^2$, we have $\zeta = 3$ or 4.

Thus, it is possible to route any two S-Box output bits to any desired locations in a polynomial number of rounds with respect to the topological level. To construct a logic gate, at least two input signals are required at desired S-Box inputs. Realizing a desired circuit in a topological level-wise manner with ℓ topological levels is possible through polynomial SPN-based block cipher rounds concerning the topological level, with the above properties.

Note that along with the discussed rounds, an additional but constant number of buffer rounds are consistently added to counteract potential security vulnerabilities stemming from IB and OB. Hence, our construction of $\mathsf{HardCODE}$ satisfies the required polynomial slowdown property of $i\mathcal{O}$. □

3.3 Proof of Security

In this section, we prove that $\mathsf{HardCODE}$ realizes a secure $Hi\mathcal{O}$ as per Definition 1. As discussed in Section 3.2, the output of $\mathsf{HardCODE}$ has three components, namely a hardware module in the form of emulator box ($M_f = \boldsymbol{\mathcal{B}}$), the publicly available keys $K_f = \big(\{K_{i,msel}\}_{i\in[d]}, \{K_{i,mcon}\}_{i\in[d]}\big)$, and auxiliary information in the form of two lists of input and output bit positions ($I_{aux} = (\mathsf{IB}, \mathsf{OB})$). When $\mathsf{HardCODE}$ obfuscates any one of two adversarially chosen circuits ckt_0 and ckt_1, $\boldsymbol{\mathcal{B}}$ does not help the adversary in distinguishing between the circuits as it is typically a hardware module that looks structurally identical irrespective of the underlying circuit. Also, note that an adversary having only a 'glass-box' access to $\boldsymbol{\mathcal{B}}$ (as specified in the adversarial model of $Hi\mathcal{O}$ in Section 2) can not distinguish based on values of msk as it is stored in a tamper-proof memory on $\boldsymbol{\mathcal{B}}$. IB and OB do not leak information about embedded structure because a block cipher can diffuse a certain intermediate state bit to any of the available n bits by introducing an adequate number of buffer layers [PDJ+19]. Apart from that, by observing the publicly available key components K_f, an adversary can

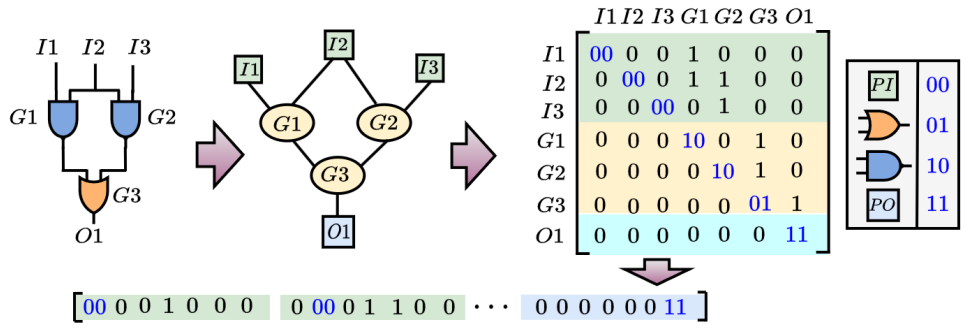


Figure 10: An overview of the proposed Adjacency matrix with circuit information. The binary encoding of logic gates, primary inputs (PI), and output (PO) have been tabulated in the inset.

not distinguish the outputs of HardCODE on two different functionally equivalent circuits ckt_0 and ckt_1 except with negligible probability. We formally state the security of HardCODE in the following theorem.

Theorem 1. *In the 'no probing' adversarial model, our proposed hardware-based indistinguishability obfuscator* HardCODE *is secure according to Definition 1 when the following assumptions hold:*

- G *is a secure PRG, and a realization of tamper-proof memory exists to securely store the small PRG seed.*
- *Key recovery from known plaintext-ciphertext pairs of underlying block cipher* $\mathcal{E}$ *is hard.*
- *Full diffusion property, as specified in [PDJ+19], holds for the pLayer of the underlying block cipher* $\mathcal{E}$*.*

Before we prove the above theorem, we prove some intermediate lemmas.

SKE:
$\mathcal{K}_{\mathsf{SKE}} = \{0,1\}^{\lambda}$, $\mathcal{M}_{\mathsf{SKE}} = \{0,1\}^{l}$, $\mathcal{C}_{\mathsf{SKE}} = \{0,1\}^{2L}$ $(\lambda \leq l \leq L)$ are the keyspace, message space and ciphertext space, respectively.
$\mathsf{G} : \{0,1\}^{\lambda} \to \{0,1\}^{3L}$ is a secure PRG.
$g : \{0,1\}^{l} \times \{0,1\}^{L} \to \{0,1\}^{2L}$ is an invertible keyed function, which is randomized in the forward direction but deterministic in the reverse direction.

- $\mathsf{KeyGen}(1^k)$: Choose a uniformly random key $\mathsf{k} \xleftarrow{\$} \mathcal{K}_{\mathsf{SKE}}$.
- $\mathsf{Encrypt}(\mathsf{k}, m)$: Let $r = \mathsf{G}(\mathsf{k})$ be the output of PRG, which can be interpreted as concatenation of three binary strings each of length L-bits, i.e., $r = r_1 \| r_2 \| r_3$. Given a message m and encryption key k, first compute $m' \leftarrow g(m, r_3)$. Then mask this output m' by XORing it with the pseudorandom bit string $r_1 \| r_2$. Return ciphertext $c = m' \oplus (r_1 \| r_2)$.
- $\mathsf{Decrypt}(\mathsf{k}, c)$: Given a ciphertext c and decryption key k, first compute $m' = c \oplus (r_1 \| r_2)$, where $r = r_1 \| r_2 \| r_3 = \mathsf{G}(\mathsf{k})$. Next apply $g^{-1}(m', r_3)$ to retrieve and return the plaintext m.

Lemma 4. *Suppose there exists an adversary* Ad_1 *which can distinguish publicly available keys (masked select and masked configuration keys, i.e., the* K_f *component) of* HardCODE *outputs for two functionally equivalent circuits. Then there must exist an adversary* Ad_2 *which can distinguish two ciphertexts of a symmetric key encryption (described as* SKE *above) scheme.*

Proof. We provide a reduction-based proof here. In the process, we must encode a circuit as a binary string. To achieve this, we first convert a circuit's DAG into an adjacency matrix and then concatenate its rows into a single binary string. The adjacency matrix differs from

Algorithm 4 Convert circuit to bit string

Require: Circuit ckt, a lookup table T, which stores a unique k-bit binary string for each type of logic gate (for e.g., OR, NAND, XOR, NOT, etc.)

function ENCODE(ckt, T)
  Let g, ip, op be the number of gates, primary inputs (PI), and primary outputs (PO) of circuit ckt respectively. Let v ← (g + ip + op).
  Initialize a matrix M of dimension v × v with all entries set to 0. ▷ Adjacency matrix for graphical representation of ckt
  for $i = 1$ **to** v **do**
    for $j = 1$ **to** v **do**
      if $i = j$ **then** ▷ Diagonal element
        $M[i][j] \leftarrow$ The k-bit binary string in T for i^{th} gate in ckt
      else if An interconnection exists from i^{th} gate to j^{th} gate **then**
        $M[i][j] \leftarrow 1$
  Initialize st as an empty string.
  for $i = 1$ **to** v **do**
    for $j = 1$ **to** v **do**
      $st \leftarrow st \| M[i][j]$ ▷ "‖" denotes concatenation of string
  return st of length v × (k + v − 1)-bits.

a traditional one since this novel representation includes information such as the node's logic gate type, primary input, or primary output in its diagonal, along with capturing the interconnections between nodes. Using diagonal elements of the adjacency matrix to store additional information is feasible since digital circuits usually do not contain self-loops due to their potential to introduce instability and undefined behavior. An illustrative example of the novel adjacency matrix and Algortihm 4 has been presented in Figure 10. The reduction steps are as follows:

1. Ad_1 sends two functionally equivalent circuits ckt_0, ckt_1 of its choice to a challenger Ch_1.

2. Ch_1 now takes part as Ad_2 in SKE game with a challenger Ch_2 as follows:

   (a) Ad_2 encodes the circuits into binary strings as $m_0 = \mathsf{Encode}(\mathsf{ckt}_0, \mathsf{T}), m_1 = \mathsf{Encode}(\mathsf{ckt}_1, \mathsf{T})$ according to the Encode function of Algorithm 4, where, T is a lookup table storing encoding of logic gates as described in the algorithm. If m_0 and m_1 are of different lengths, the smaller string is padded with a required number of zeros. Both m_0 and m_1 are sent to Ch_2.

   (b) Ch_2 retrieves the circuits $\mathsf{ckt}_0 = \mathsf{Decode}(m_0, \mathsf{T})$ and $\mathsf{ckt}_1 = \mathsf{Decode}(m_1, \mathsf{T})$. Note that the procedure Decode for converting a binary string to a corresponding circuit is straightforward, as it is just the reverse of Algorithm 4. Assume total number of gates in ckt_0 and ckt_1 to be g_0 and g_1 respectively. Fix the number of cipher rounds $d = max(\mathsf{g}_0, \mathsf{g}_1)$ so that the size of publicly available keys does not leak any information about which circuit is obfuscated.

   (c) Ch_2 now runs $\mathsf{KeyGen}(1^\lambda)$ to obtain secret k, chooses a bit $b \xleftarrow{\$} \{0,1\}$, and runs $\mathsf{Encrypt}(\mathsf{k}, m_b)$. Within Encrypt, function g is instantiated with Embed (ref. Algorithm 3) for some block cipher $\mathcal{E}$ with L rounds and its input m is instantiated with ckt_b. Note that Embed is an invertible function since the underlying circuit can be uniquely determined from select and configuration keys in the presence of round keys by the construction of HardCODE. The output m' of the function g is the concatenation of select keys and configuration keys returned by Embed function. This output m' is then masked with $(r_1 \| r_2)$, i.e., $2L$ pseudorandom bits, generated by $\mathsf{G}(\mathsf{k})$. Finally, the ciphertext (c_b) is the concatenation of masked select and configuration keys. Ch_2 returns ciphertext c_b to Ad_2 along with $\boldsymbol{\mathcal{B}}$, IB and OB[3].

[3] IB and OB can be fixed by the challenger for all circuits with same primary input and output size irrespective of bit b

3. Ch_1 (i.e., Ad_2), now forwards c_b, IB, OB and $\boldsymbol{\mathcal{B}}$ to Ad_1 as an output of obfuscator when applied on circuit ckt_b.

4. Ad_1 returns a guess bit b' to Ch_1.

5. Ch_1 (i.e., Ad_2) then forwards the same guess bit b' to Ch_2.

It is easy to infer that the $i\mathcal{O}$ game between Ad_1 and Ch_1 is exactly same as the SKE game between Ad_2 and Ch_2, while ignoring the components $\boldsymbol{\mathcal{B}}$, IB and OB. These components can be ignored as they essentially look the same irrespective of the underlying message/circuit. Hence, if Ad_1 can distinguish obfuscation of ckt_0 from that of ckt_1 with non-negligible probability, Ad_2 can distinguish two ciphertexts of SKE in presence of eavesdropper with non-negligible probability. □

Lemma 5. *Provided* G *is a secure PRG,* SKE *has indistinguishable encryptions in the presence of an eavesdropper.*

Proof. We present a reduction here from the indistinguishable encryptions game of SKE in the presence of an eavesdropper to the security game of PRG.

Let us assume that a challenger Ch' challenges a distinguisher D to distinguish the output of a PRG from a truly random output of the same bit length. D makes use of an adversary Ad to accomplish the task, assuming that Ad can distinguish between encryptions of two different plaintexts of its choice in SKE. The reduction is as follows:

1. Ch' samples a bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, it samples $r \xleftarrow{\$} \{0,1\}^{3L}$, else computes $r = \mathsf{G}(\mathsf{k})$, where $\mathsf{k} \xleftarrow{\$} \{0,1\}^{\lambda}$. String r is sent to D.

2. D now engages in an interaction with Ad as described in the following:

   (a) D requests and receives two plaintexts m_0 and m_1 from Ad.

   (b) D chooses a bit $b' \xleftarrow{\$} \{0,1\}$. It then computes the challenge ciphertext $c_{b'} = g(m_{b'}) \oplus r$ and sends it to Ad.

   (c) Ad now guesses a bit b'' and sends to D.

3. D now outputs bit $\hat{b} = 1$, if Ad's guess is correct, i.e., $b' = b''$, else outputs $\hat{b} = 0$.

Note that if r is a truly random string (i.e., $b = 0$), the ciphertext in SKE is obtained by masking $g(m_{b'})$ with a truly random string of same length, which is similar to the construction of one time pad (OTP). In this case, the probability with which Ad guesses b'' correctly (i.e., $b'' = b'$), is exactly $\frac{1}{2}$. Hence,

$$\mathsf{Pr}[\mathsf{Ad} \text{ outputs } b'' = b' | b = 0] = \frac{1}{2}.$$

On the other hand, if r is a PRG-generated string, then the ciphertext is generated exactly according to the way we define the construction of SKE. We denote winning probability of Ad in this case with p, i.e.,

$$\mathsf{Pr}[\mathsf{Ad} \text{ outputs } b'' = b' | b = 1] = p.$$

Since G is a secure PRG, winning probability of D is negligible, i.e.,

$$\left|\mathsf{Pr}[\mathsf{D} \text{ outputs } \hat{b} = b]\right| \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Let us now elaborate on the winning probability of D in the PRG game as follows:

$$\begin{aligned}
&\Big|\mathsf{Pr}[\mathsf{D} \text{ outputs } \hat{b} = b]\Big| \\
&= \Big|\mathsf{Pr}[b=0] \cdot \mathsf{Pr}\big[\frac{\mathsf{D} \text{ outputs } \hat{b}=0}{b=0}\big] + \mathsf{Pr}[b=1] \cdot \mathsf{Pr}\big[\frac{\mathsf{D} \text{ outputs } \hat{b}=1}{b=1}\big]\Big| \\
&= \Big|\frac{1}{2} \cdot (1 - \mathsf{Pr}\big[\frac{\mathsf{D} \text{ outputs } \hat{b}=1}{b=0}\big]) + \frac{1}{2} \cdot \mathsf{Pr}\big[\frac{\mathsf{D} \text{ outputs } \hat{b}=1}{b=1}\big]\Big| \\
&= \Big|\frac{1}{2} \cdot (1 - \mathsf{Pr}\big[\frac{\mathsf{Ad} \text{ outputs } b''=b'}{b=0}\big]) + \frac{1}{2} \cdot \mathsf{Pr}\big[\frac{\mathsf{Ad} \text{ outputs } b''=b'}{b=1}\big]\Big| \\
&= \Big|\frac{1}{2}(1-\frac{1}{2}) + \frac{1}{2} \cdot p\Big| = \Big|\frac{1}{4} + \frac{1}{2} \cdot p\Big|
\end{aligned}$$

This quantity being upper bounded by $\frac{1}{2} + \mathsf{negl}(\lambda)$, we get $p \leq \frac{1}{2} + \mathsf{negl}(\lambda)$. So, the winning probability of a PPT adversary in SKE is negligibly better than $\frac{1}{2}$ or, in other words, SKE has indistinguishable encryptions in the presence of an eavesdropper. □

Lemma 6. *Given* G *is a secure PRG, an adversary can not distinguish the publicly available keys of two functionally equivalent* HardCODE*-obfuscated circuits except with negligible probability* $\mathsf{negl}(\lambda)$*, where* λ *is the length of the PRG seed.*

Proof. Lemma 4 and Lemma 5 together imply this. □

Lemma 7. *Provided that pLayer of the underlying block cipher* $\mathcal{E}$ *satisfies full diffusion property [PDJ+19], the input bit positions* (IB) *and output bit positions* (OB) *of a* HardCODE*-obfuscated circuit does not leak information about the type of logic gates and their positions in the augmented block cipher* ($\mathcal{E}'$).

Proof. By construction of HardCODE, $\mathcal{E}'$ incorporates an adequate number of buffer layers to guarantee full diffusion of the circuit input and output bit positions. Hence, an adversary is not able to deduce the actual S-Boxes responsible for implementing the gates of the first and final topological level and, in turn, the intermediate levels of the ckt. □

Lemma 8. *Assuming that key recovery problem from known plaintext-ciphertext pairs of underlying block cipher* $\mathcal{E}$ *is hard, we can establish the hardness of key recovery from known plaintext-ciphertext pairs of augmented block cipher* $\mathcal{E}'$.

Proof. We defer the detailed discussion and proof of this lemma (stated again as Theorem 2) to Section 4. □

Proof of Theorem 1. Assuming that it is possible to securely store the small PRG seed in tamper-proof hardware and G is a secure PRG, and full diffusion property of *pLayer* holds, it has been proved that publicly available keys and input-output bit positions (IB,OB) of two HardCODE-obfuscated functionally equivalent circuits are indistinguishable. Further, Lemma 8 states that $\mathcal{E}'$ does not leak information about the keys (select keys, configuration keys, round keys) used in embedding through its input-output behavior. Hence, combining Lemmas 6, 7, and 8 proves Theorem 1. □

4 Hardness of Key Recovery from $\mathcal{E}'$

The previous section discussed the security proof for the HardCODE construction. Recall that the HardCODE output includes two public key sets ($K_f = \big(\{K_{i,msel}\}_{i\in[d]}, \{K_{i,mcon}\}_{i\in[d]}\big)$), input/output bit position lists ($I_{aux} = (\mathsf{IB}, \mathsf{OB})$), and an emulator box $\boldsymbol{\mathcal{B}}$, which emulates the circuit's functionality using the public keys (see Figure 9). Within $\boldsymbol{\mathcal{B}}$, the augmented block cipher $\mathcal{E}'$ uses three n-bit key sets: round key ($K_{i,r}$), select key ($K_{i,sel}$), and configuration key ($K_{i,con}$) across d rounds. When an adversary encounters a HardCODE-obfuscated circuit in the form of ($M_f = \boldsymbol{\mathcal{B}}, K_f, I_{aux}$), they may attempt to recover these internal keys by analyzing the input-output behavior of the hardware module in presence of K_f, I_{aux}. Since the core component of $\boldsymbol{\mathcal{B}}$ is the augmented SPN $\mathcal{E}'$, the key recovery problem for the HardCODE-obfuscated circuit reduces to breaking the augmented block cipher/SPN in a known-plaintext attack.

While block ciphers are considered heuristically secure, without formal proofs, their designs rely on cryptographic principles like confusion and diffusion. Some works [GK11, MM00] propose encoding block cipher encryption as a Boolean satisfiability problem (SAT) and using SAT solvers for cryptanalysis, termed logical cryptanalysis. If a block cipher is secure, the corresponding SAT instance is hard to solve, as recovering the secret key via such attacks becomes infeasible [CM97].

In HardCODE, we assume the SPN-based block cipher $\mathcal{E}$ is secure, making the associated key recovery SAT problem hard. However, we modify the cipher by adding a MUX layer and using three key sets (round, select, and configuration keys) in the augmented block cipher $\mathcal{E}'$, compared to the single round key in $\mathcal{E}$. Therefore, the central concept revolves around representing $\mathcal{E}'$ as a SAT problem, as demonstrated in the following lemma.

Lemma 9. *The key recovery problem of an augmented SPN-based block cipher ($\mathcal{E}'$) in* HardCODE *can be presented as a SAT problem.*

Proof. In logical cryptanalysis, the encryption function of a block cipher is expressed through a propositional formulation denoted as $Enc(\mathbf{P}, \mathbf{K}, \mathbf{C})$, where $\mathbf{P}$, $\mathbf{K}$, and $\mathbf{C}$ represent individual propositional variables that correspond to the plaintext, key, and ciphertext of the cipher, respectively. The propositional formula attains a truth value of "true" (i.e., the corresponding bit evaluates to 1) exclusively when the Boolean assignments assigned to $\mathbf{P}$, $\mathbf{K}$, and $\mathbf{C}$ satisfy the mapping defined by the block cipher. Now in a known plaintext attack, an adversary possesses knowledge of the Boolean values v_p representing the plaintext $\mathbf{P}$ and the corresponding ciphertext values (v_c), the task of key recovery is equivalent to identifying a suitable assignment for the unknown $\mathbf{K}$ in propositional formula $Enc(v_p, \mathbf{K}, v_c)$. If q pairs of plaintext-ciphertext are available, the key search can be formalized by conjoining the corresponding formulae [MM00]: $\bigwedge_{i=1}^{q} Enc(v_p^i, \mathbf{K}, v_c^i)$. Now, the fundamental requirement is to transform each of the intermediate layers (MUX layer, *addRoundKey*, *sBoxLayer*, *pLayer*) of a single round of an SPN-based augmented block cipher $\mathcal{E}'$ into an equivalent propositional formula [MM00]. CNF (Conjunctive Normal Form) expression for MUX layer and *addRoundKey* can be found using Tseytin transformation [Tse83] of logic gates. CNF of *sBoxLayer* is found by leveraging the inherent relation between the S-Box's input and output bits. To incorporate the effect of *pLayer* in the encoding process, we employ a pre-processing step to directly integrate the outcome of the previous layer into the permuted input positions of the next layer.

In this manner, we can effectively translate all rounds of $\mathcal{E}'$ into a 3-CNF expression, thereby enabling its representation as a 3-SAT problem. We refer the reader to Appendix E for a detailed layer-wise conversion. □

Finally, we argue the hardness of key recovery from augmented block cipher $\mathcal{E}'$ in the following theorem.

Theorem 2. *Given that the key recovery of $\mathcal{E}$ is hard, the key recovery problem of $\mathcal{E}'$ is also hard.*

Proof. The augmented block cipher $\mathcal{E}'$ in HardCODE, can behave as the underlying block cipher $\mathcal{E}$. The CNF encoding of $\mathcal{E}'$ (ref. Lemma 9) can be reduced to the CNF encoding of $\mathcal{E}$, if all the bits in the select keys ($\{K^j_{i,sel}\}_{i\in[d],j\in[n]}$) are constrained to be 0, and the round keys $\{K^j_{i,r}\}_{i\in[d],j\in[n]}$ are generated from the key scheduling algorithm of $\mathcal{E}$. This scenario corresponds to the situation in which the MUX layer, introduced additionally, does not try to fix bits at S-Box inputs. Hence, the configuration keys become irrelevant, and the MUX layer effectively acts as a transparent pass-through layer, allowing the output of the previous round to flow into the *addRoundKey* operation of the current round. Given that the CNF encoding of $\mathcal{E}$ is already established as a challenging SAT problem, it can be deduced that the CNF encoding of $\mathcal{E}'$ also constitutes a challenging instance. Hence, we can conclude that key recovery from $\mathcal{E}'$ is also difficult based on the complexity of SAT problem presented by its CNF encoding. □

Appendix F illustrates an encoding example from Lemma 9 and a discussion on the hardness of key recovery.

5 Application and Implementation

In this section, we apply HardCODE to obfuscate the C-17 circuit from the ISCAS-85 benchmark [BF85], using PRESENT as the underlying block cipher. We include Figure 11 (in Appendix), which shows two embeddings of functionally equivalent versions of the C-17 circuit (C_1 and C_2) using six rounds of augmented PRESENT. Though the actual implementation includes initial/final buffer rounds for security, those are omitted in the figure for simplicity. The input and output bit positions (IB, OB) are $\{1, 8, 9, 20, 34, 50\}$ and $\{2, 32\}$, respectively, for both embeddings. Both embeddings are functionally identical to C-17 and require the same number of PRESENT rounds, scaling polynomially with their topological levels. Even if the embeddings require a different number of PRESENT rounds, HardCODE provides the flexibility to add buffer rounds to the smaller one without affecting functionality. In Figure 11, color-coded S-Boxes represent logic gates, grey S-Boxes serve as buffers, and blue lines show signal interconnections. Please note that both embeddings in Figure 11 appear identical without knowledge of the three sets of keys, which in turn decides the set of S-Boxes functioning as logic gates/buffers.

After embedding, the keys ($K^j_{i,sel}$, $K^j_{i,con}$) are masked ($K^j_{i,msel}$, $K^j_{i,mcon}$) and publicly released, while the secret key (msk) is securely stored. The adversary observes the uniform code structure and masked keys. To distinguish between two embeddings, they must guess the exact embedding represented by the round, select, and configuration keys.

Implementation Details. We realize a practical implementation of the HSM on a NEXYS4 DDR board, which features an Artix-7 FPGA. The secret key is stored in a BRAM accessible to the logic, and storing an 80-bit key (msk) consumes 0.5 BRAM. In this implementation, we integrate the obfuscated C-17 circuit with an AES-128 core, which is used to generate 1344-bit round keys required by the obfuscated C-17. The AES core operates on an 80-bit key, which is zero-padded to form a 128-bit key. A fixed plaintext is stored within the FPGA, and the AES

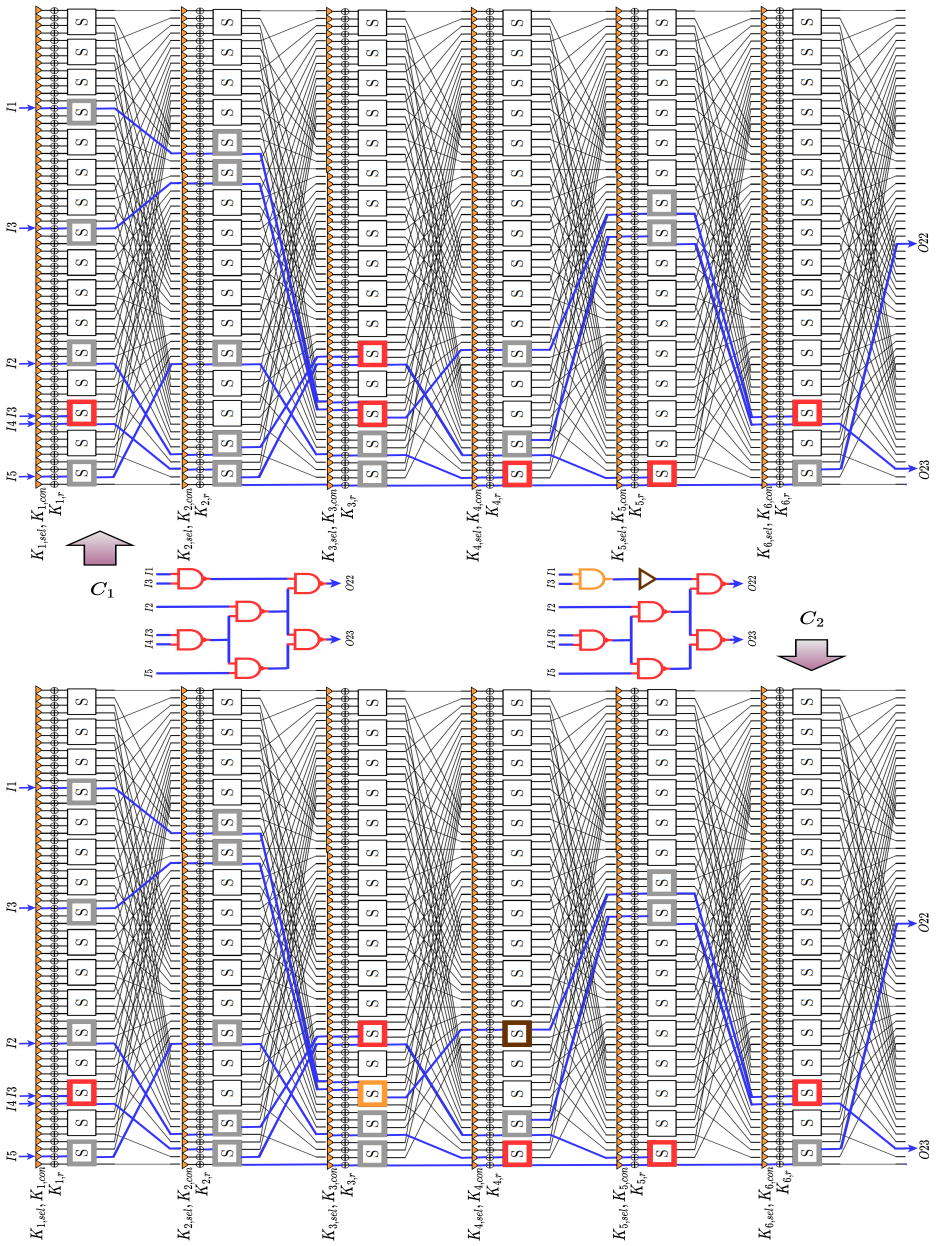


Figure 11: Indistinguishability between embeddings of two functionally equivalent realizations of C-17 (C_1 and C_2) from ISCAS-85 into augmented PRESENT.

Table 2: Resource consumption and timing Details of the Implementation.

Design Decription	LUTs	FFs	Slices	BRAMs	Latency (CC)	Critical Path	Freq. (MHz)
HardCODE	4090	1916	1226	0.5	111	15.3 ns	65.3

engine is executed 11 times to derive the round keys. These are then supplied to the obfuscated C-17 circuit. The complete implementation occupies 4090 LUTs (1226 slices) and has a critical path delay of 15.3 ns. The total latency of the system is 111 clock cycles, which includes the AES-based round key generation process. The details of the resource consumption and the frequency of operation has been illustrated in Table 2.

Side Channel Analysis. Recall that our HardCODE construction is built on an unrolled architecture corresponding to an augmented SPN-based block cipher. Each round contains some S-Boxes functioning as logic gates or buffers, contributing to essential computations, while others are active but perform no meaningful computations, adding noise. As a result, we expect HardCODE to have a lower Signal-to-Noise Ratio (SNR). This expectation is validated through Test Vector Leakage Assessment (TVLA), as described below.

TVLA [LAS00] is a widely used method for identifying side-channel vulnerabilities in cryptographic implementations by applying Welch's t-test [Wel47] to detect differences in the means between two datasets. In our side-channel analysis, 200,000 electromagnetic traces were collected from our implementation on the Cora-Z7 board. TVLA has been performed using 100,000 random and 100,000 fixed traces, with results (ref. Figure 12) indicating that all leakage values remain within ± 4.5. While this does not fully guarantee the resilience of our hardware implementation against side-channel attacks, it does confirm that the proposed implementation demonstrates sufficient resistance to such attacks when tested with a reasonable number of traces.

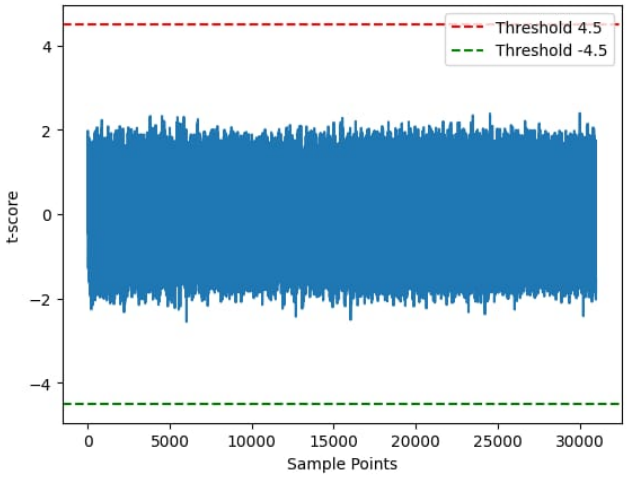


Figure 12: TVLA results for the implemented C-17 with 10^5 fixed and 10^5 random traces.

References

[Aea20] K. Z. Azar and et al. Nngsat: Neural network guided sat attack on logic locked complex structures. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2020.

[Agr19] Shweta Agrawal. Indistinguishability obfuscation without multilinear maps: new methods for bootstrapping and instantiation. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*, pages 191–225. Springer, 2019.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *Annual Cryptology Conference*, pages 308–326. Springer, 2015.

[AJS18] Prabhanjan Ananth, Aayush Jain, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: io from lwe, bilinear maps, and weak pseudorandomness. *Cryptology ePrint Archive*, 2018.

[APSS21] Lilas Alrahis, Satwik Patnaik, Muhammad Shafique, and Ozgur Sinanoglu. Omla: An oracle-less machine learning-based attack on logic locking. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(3):1602–1606, 2021.

[BBKK18] Boaz Barak, Zvika Brakerski, Ilan Komargodski, and Pravesh K Kothari. Limits on low-degree pseudorandom generators (or: Sum-of-squares meets program obfuscation). In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37*, pages 649–679. Springer, 2018.

[BCG+18] Nir Bitansky, Ran Canetti, Sanjam Garg, Justin Holmgren, Abhishek Jain, Huijia Lin, Rafael Pass, Sidharth Telang, and Vinod Vaikuntanathan. Indistinguishability obfuscation for ram programs and succinct randomized encodings. *SIAM Journal on Computing*, 47(3):1123–1210, 2018.

[BDGM23] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate io from homomorphic encryption schemes. *Journal of Cryptology*, 36(3):27, 2023.

[BF85] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator. In *Fortran. ISCAS'85*, 1985.

[BGI+01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. In *Annual international cryptology conference*, pages 1–18. Springer, 2001.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[BHJ+19] Boaz Barak, Samuel B Hopkins, Aayush Jain, Pravesh Kothari, and Amit Sahai. Sum-of-squares meets program obfuscation, revisited. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*, pages 226–250. Springer, 2019.

[BKL+07] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. Present: An ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9*, pages 450–466. Springer, 2007.

[BKP23] Kaartik Bhushan, Venkata Koppula, and Manoj Prabhakaran. Homomorphic indistinguishability obfuscation and its applications. *Cryptology ePrint Archive*, 2023.

[BLA17] William J Buchanan, Shancang Li, and Rameez Asif. Lightweight cryptography methods. *Journal of Cyber Security Technology*, 1(3-4):187–201, 2017.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. Gift: A small present: Towards reaching the limit of lightweight encryption. In *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 321–345. Springer, 2017.

[BR14] Zvika Brakerski and Guy N Rothblum. Black-box obfuscation for d-cnfs. In *Proceedings of the 5th conference on Innovations in theoretical computer science*, pages 235–250, 2014.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings 4*, pages 535–554. Springer, 2007.

[BZ15] Mark Bun and Mark Zhandry. Order-revealing encryption and the hardness of private learning. In *Theory of Cryptography Conference*, pages 176–206. Springer, 2015.

[BZ17] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. *Algorithmica*, 79:1233–1285, 2017.

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Advances in Cryptology—CRYPTO'97: 17th Annual International Cryptology Conference Santa Barbara, California, USA August 17–21, 1997 Proceedings 17*, pages 455–469. Springer, 1997.

[CD08] Ran Canetti and Ronny Ramzi Dakdouk. Extractable perfectly one-way functions. In *Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II 35*, pages 449–460. Springer, 2008.

[CGH17] Yilei Chen, Craig Gentry, and Shai Halevi. Cryptanalyses of candidate branching program obfuscators. In *Advances in Cryptology–EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30–May 4, 2017, Proceedings, Part III 36*, pages 278–307. Springer, 2017.

[CHN+16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 1115–1127, 2016.

[CLT13] Jean-Sébastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *Annual Cryptology Conference*, pages 476–493. Springer, 2013.

[CM97] Stephen A Cook and David G Mitchell. Finding hard instances of the satisfiability problem: A survey. *Satisfiability Problem: Theory and Applications*, 35:1–17, 1997.

[CRV10] Ran Canetti, Guy N Rothblum, and Mayank Varia. Obfuscation of hyperplane membership. In *Theory of Cryptography: 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings 7*, pages 72–89. Springer, 2010.

[Das23] Dipnarayan Das. *Design and Implementation of Cross-Platform PUF-based Hardware security for Computer System*. PhD thesis, National Institute of Technology Durgapur, India, 2023.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[EKP+07] Thomas Eisenbarth, Sandeep Kumar, Christof Paar, Axel Poschmann, and Leif Uhsadel. A survey of lightweight-cryptography implementations. *IEEE Design & Test of Computers*, 24(6):522–533, 2007.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*, pages 1–17. Springer, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *Theory of Cryptography: 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II 12*, pages 498–527. Springer, 2015.

[GGH+16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *Theory of Cryptography: 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings 7*, pages 308–326. Springer, 2010.

[GJK18] Craig Gentry, Charanjit S Jutla, and Daniel Kane. Obfuscation using tensor products. *Cryptology ePrint Archive*, 2018.

[GK11] Matthew Gwynne and Oliver Kullmann. Towards a better understanding of hardness. In *The Seventeenth International Conference on Principles and Practice of Constraint Programming (CP 2011): Doctoral Program Proceedings*, pages 37–42, 2011.

[GMM17] Sanjam Garg, Mohammad Mahmoody, and Ameer Mohammed. Lower bounds on obfuscation from all-or-nothing encryption primitives. In *Annual International Cryptology Conference*, pages 661–695. Springer, 2017.

[GP21] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 736–749, 2021.

[GR07] Shafi Goldwasser and Guy N Rothblum. On best-possible obfuscation. In *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings 4*, pages 194–213. Springer, 2007.

[Gue09] Shay Gueron. Intel's new aes instructions for enhanced performance and security. In *International Workshop on Fast Software Encryption*, pages 51–66. Springer, 2009.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from lwe. In *Annual Cryptology Conference*, pages 503–523. Springer, 2015.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.

[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 *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[KSS23] Abhishek Kumar, Manoj Sindhwani, and Shippu Sachdeva. An innovative architecture of dram puf. *Journal of Integrated Circuits and Systems*, 18(2):1–9, 2023.

[LAS00] Tal Lev-Ami and M Sagiv. Tvla: A framework for kleene logic based static analyses. *Master's thesis, Tel Aviv University*, 2000.

[Lin16] Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I 35*, pages 28–57. Springer, 2016.

[LM18] Huijia Lin and Christian Matt. Pseudo flawed-smudging generators and their application to indistinguishability obfuscation. *Cryptology ePrint Archive*, 2018.

[LV17] Alex Lombardi and Vinod Vaikuntanathan. Limits on the locality of pseudorandom generators and applications to indistinguishability obfuscation. In *Theory of Cryptography Conference*, pages 119–137. Springer, 2017.

[MM00] Fabio Massacci and Laura Marraro. Logical cryptanalysis as a sat problem. *Journal of Automated Reasoning*, 24(1-2):165, 2000.

[MZS24] Isaac McDaniel, Michael Zuzak, and Ankur Srivastava. Removal of sat-hard instances in logic obfuscation through inference of functionality. *ACM Transactions on Design Automation of Electronic Systems*, 29(4):1–23, 2024.

[NFR+17] Kartik Nayak, Christopher W Fletcher, Ling Ren, Nishanth Chandran, Satya V Lokam, Elaine Shi, and Vipul Goyal. Hop: Hardware makes obfuscation practical. In *NDSS*, volume 2017, page 0, 2017.

[PDJ+19] Sikhar Patranabis, Nilanjan Datta, Dirmanto Jap, Jakub Breier, Shivam Bhasin, and Debdeep Mukhopadhyay. Scadfa: Combined sca+ dfa attacks on block ciphers with practical validations. *IEEE Transactions on Computers*, 68(10):1498–1510, 2019.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34*, pages 500–517. Springer, 2014.

[RPK+21] Nikhil Rangarajan, Satwik Patnaik, Johann Knechtel, Shaloo Rakheja, and Ozgur Sinanoglu. *Next Era in Hardware Security*. Springer, 2021.

[SB25] Sora Suegami and Enrico Bottazzi. Diamond iO: A straightforward construction of indistinguishability obfuscation from lattices. Cryptology ePrint Archive, Paper 2025/236, 2025.

[SCMC22] Akashdeep Saha, Urbi Chatterjee, Debdeep Mukhopadhyay, and Rajat Subhra Chakraborty. Dip learning on cas-lock: using distinguishing input patterns for attacking logic locking. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 688–693. IEEE, 2022.

[Sea17] K. Shamsi and et. al. AppSAT: Approximately deobfuscating integrated circuits. In *HOST*, pages 95–100. IEEE, 2017.

[SRM15] P. Subramanyan, S. Ray, and S. Malik. Evaluating the security of logic encryption algorithms. In *HOST*, pages 137–143. IEEE, 2015.

[SS19] D. Sirone and P. Subramanyan. Functional analysis attacks on logic locking. In *DATE*, pages 936–939. IEEE, 2019.

[SSC+20] Akashdeep Saha, Sayandeep Saha, Siddhartha Chowdhury, Debdeep Mukhopadhyay, and Bhargab B Bhattacharya. LoPher: SAT-Hardened Logic Embedding on Block Ciphers. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 475–484, 2014.

[SXTF20] Bicky Shakya, Xiaolin Xu, Mark Tehranipoor, and Domenic Forte. Cas-lock: A security-corruptibility trade-off resilient logic locking scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 175–202, 2020.

[SZR+23] Simranjeet Singh, Furqan Zahoor, Gokul Rajendran, Sachin Patkar, Anupam Chattopadhyay, and Farhad Merchant. Hardware security primitives using passive rram crossbar array: Novel trng and puf designs. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, pages 449–454, 2023.

[Tri03] Stephen M Trimberger. Method and apparatus for protecting proprietary configuration data for programmable logic devices, November 25 2003. US Patent 6,654,889.

[Tse83] Grigori S Tseitin. On the complexity of derivation in propositional calculus. *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, pages 466–483, 1983.

[Wel47] Bernard L Welch. The generalization of 'student's'problem when several different population varlances are involved. *Biometrika*, 34(1-2):28–35, 1947.

[YLK21] Sze Ling Yeo, Duc-Phong Le, and Khoongming Khoo. Improved algebraic attacks on lightweight block ciphers. *Journal of Cryptographic Engineering*, 11:1–19, 2021.

[YMRS16] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu. SARLock: SAT attack resistant logic locking. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 236–241. IEEE, 2016.

[YSN+17] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu. Provably-secure logic locking: From theory to practice. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1601–1618, 2017.

A Some Related Definitions

Definition 2 (Pseudorandom Generator (PRG))**.** *Let λ be the security parameter and l is a polynomial in λ. Let G be a deterministic polynomial-time algorithm such that for any input $s \in \{0,1\}^\lambda$, it outputs $\mathsf{G}(s) \in \{0,1\}^l$. We say that G is a pseudorandom generator if the following two conditions are satisfied.*

- ***Expansion:*** *For any λ, $l > \lambda$ should hold.*
- ***Pseudorandomness:*** *The pseudorandomness property of PRG is the key element of security of PRG. The security of PRG can be represented in the form of negligible distinguishing advantage of a PPT distinguisher $\mathcal{D}$ in a game against a challenger $\mathcal{C}$ as described in the following.*

  1. *Challenger $\mathcal{C}$ chooses a bit $b \xleftarrow{R} \{0,1\}$.*
  2. *If $b = 0$, challenger $\mathcal{C}$ samples $y \xleftarrow{R} \{0,1\}^l$. Else chooses a uniform random $s \xleftarrow{R} \{0,1\}^\lambda$ and computes $y = \mathsf{G}(s)$. Return y to $\mathcal{D}$.*
  3. *The above step can be repeated a polynomial number of times.*
  4. *$\mathcal{D}$ outputs a guess bit $b' \in \{0,1\}$.*

  We say that G is pseudorandom if the probability of correct guess by $\mathcal{D}$ is negligibly greater than $\frac{1}{2}$, or in other words, distinguishing advantage of $\mathcal{D}$ is $\mathsf{negl}(\lambda)$.

$$\mathsf{Pr}[\mathcal{D} \text{ outputs } b' = b] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Definition 3 (Symmetric Key Encryption Scheme)**.** *A symmetric-key encryption scheme consists of three polynomial-time algorithms KeyGen, $\mathsf{Encrypt}$, and $\mathsf{Decrypt}$ as follows.*

- $\mathsf{KeyGen}(1^\lambda)$*: Given security parameter λ, it outputs a key k.*
- $\mathsf{Encrypt}(k, m)$*: Given a plaintext message m, it outputs ciphertext c using encryption key k.*
- $\mathsf{Decrypt}(k, c)$*: Given a ciphertext c, it outputs plaintext m, using the decryption key k.*

Indistinguishable Encryption in Presence of Eavesdropper. *We say that a symmetric key encryption scheme has indistinguishable encryptions in the presence of a PPT eavesdropper $\mathcal{A}$, if $\mathcal{A}$ wins the following game against a challenger $\mathcal{C}$ with probability only negligibly greater than $\frac{1}{2}$. Note that $\mathcal{A}$ can win the game with probability $\frac{1}{2}$ just by making a random guess.*

1. *$\mathcal{C}$ runs KeyGen to generate key k.*
2. *$\mathcal{A}$ sends two plaintexts m_0 and m_1 to $\mathcal{C}$ such that $|m_0| = |m_1|$.*
3. *$\mathcal{C}$ chooses a bit $b \xleftarrow{\$} \{0,1\}$. Then it computes and sends $\mathsf{Encrypt}(k, m_b)$ to $\mathcal{A}$.*
4. *$\mathcal{A}$ outputs a guess bit b'.*

$\mathcal{A}$ wins the game if $b' = b$. A symmetric key encryption scheme is secure if for any adversary $\mathcal{A}$ there exists negligible function $\mathsf{negl}(\lambda)$ such that,

$$\left|\mathsf{Pr}[\mathcal{A} \text{ wins}]\right| \leq \frac{1}{2} + \mathsf{negl}(\lambda) \implies \left|\mathsf{Pr}\left[\frac{b'=1}{b=0}\right] - \mathsf{Pr}\left[\frac{b'=1}{b=1}\right]\right| \leq \mathsf{negl}(\lambda).$$

B An overview of PRESENT

PRESENT [BKL+07] is a lightweight block cipher designed with a focus on efficiency and security for use in resource-constrained environments, such as low-power IoT devices, smart cards, and RFID tags. It was developed by the Orange Labs (France), Ruhr University Bochum (Germany), and the Technical University of Denmark in 2007. The block size of PRESENT is 64 bits and the key size is 80 bits or 128 bits. PRESENT has a substitution-permutation network structure, which obscures the relation between plaintext and ciphertext. For 80-bit keys, it typically uses 31 rounds, while for 128-bit keys, it employs 32 rounds. Addition of more rounds contributes to the higher security level. By construction each *SBoxLayer* of PRESENT consists of 16 S-boxes and each of them is a 4-bit to 4-bit S-Box with specific input-output property.

C A Note on Collision of Signals

We explain *collision* with an example. Considering Figure 13, in which we try to embed a circuit (Figure 13(a)), by realizing the logic gates of the circuit with S-Boxes, following the color coding for different gates in Figure 13(a). The circuit has three topological levels. While embedding the entire first topological level utilizing a single round of the SPN-based block cipher, we come across a situation in which the gate output of the first and the third S-Box tries to pass through the first S-Box in the second round of the cipher (marked in red). Hence, a collision occurs.

A solution to circumvent the *collision* problem is to realize one topological level using multiple rounds of the underlying block ciphers. This corresponds to a situation in which the number of S-Boxes in a layer is always more than the number of gates that need to be embedded. So, a non-colliding path always exists to route signals to the desired S-Box input locations. For example, in Figure 13(c), we avoid collision by realizing the first topological level utilizing two cipher rounds.

Let us consider an extreme scenario in which we embed each gate of a topological level of an arbitrary circuit using one round of the block cipher. In such a case, the number of rounds required to embed a single topological level will be equal to the number of gates in that particular level, which remains constant for each level. We can infer that this is the worst-case scenario of embedding a circuit in a block cipher that requires the maximum number of rounds to embed.

For an S-Box in any intermediate round to realize a desired logic gate, the gate input signals must be passed on from the first round of the cipher. One can not inject input signals in intermediate rounds without impeding the underlying block cipher algorithm; hence it is not feasible in HardCODE. S-Boxes in HardCODE can behave as buffer gates passing the signal from one cipher round to another. The number of gates that can be realized depends on the number of gate input signals that can be passed through buffer S-Boxes.

D Role of PRG Round Keys in Embedding Circuits

To realize gates utilizing S-Boxes, the inputs must be provided at particular positions governed by the previous rounds' *pLayer*. Consequently, determining the functionality shift between G and G' dynamically in any intermediate round based on the *addRoundKey* bit value of the

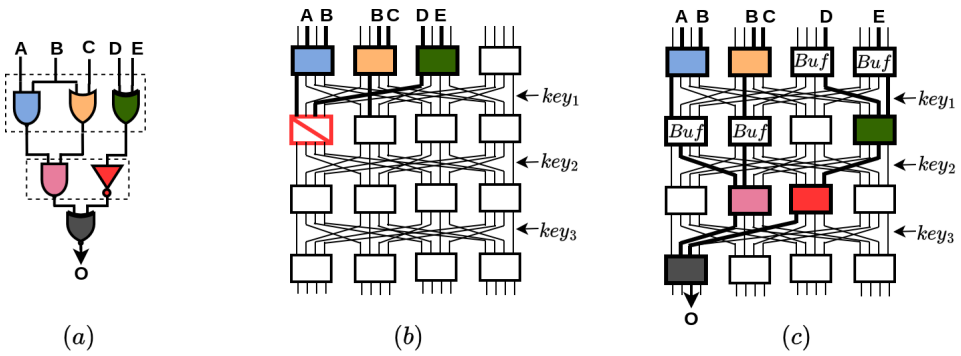


Figure 13: An instance of collision. (a) A circuit with three topological levels, highlighted in dotted enclosures. (b) Occurrence of collision in the second round S-Box, marked in red. (c) Collision is resolved when the first topological level is realized across multiple cipher rounds.

subsequent round could disrupt the structural integrity of prior rounds. This challenge can be mitigated by ensuring that G and G' exhibit identical input-output bit configurations for S-Boxes. It is noteworthy that achieving two complementary gates with matching input bit positions and generating output through the same output bit, as observed in both GIFT [BPP+17] and PRESENT (refer Figure 5, inset), is feasible by utilizing distinct configuration bits at the input of the S-Boxes. This design approach facilitates gate complementarity (G and G') and preserves the consistency of input-output bit configurations, thereby addressing the potential distortion issue associated with dynamic functionality shifts between rounds. As previously discussed, the initial rounds of HardCODE incorporate buffer layers to prevent potential information leakage through the set IB. The input bits within set IB passes through *addRoundKey* in the first round before traversing to the S-Boxes realizing the first round buffers. The input bits may be complemented if the *addRoundKey* value is 1 for the corresponding bit positions. A straightforward countermeasure to this involves passing complemented values of the input bits at the initial round. However, this approach raises concerns about potential information leakages related to the first *addRoundKey* bit values.

Our proposed embedding algorithm addresses this challenge by configuring S-Boxes in the first round by examining the round key bits of both the present (1^{st}) round and the subsequent (2^{nd}) round with the corresponding permuted bits, instead of examining round key bits of only the subsequent round. Accordingly, we configure the S-Boxes of the first round to function as buffers if the round keys for both rounds are identical (00 or 11). Conversely, if the round keys differ (01 or 10), we configure the S-Boxes to NOT gates. This approach ensures inputs consistently reach the second round S-Boxes unaltered.

Likewise, the final rounds of HardCODE incorporate buffer layers to prevent potential information leakage through the set OB. The S-Boxes in the last round, which generate the final outputs within set OB after the *pLayer*, are simply configured as buffers since there are no further XOR operations after that.

The values of the select and the configuration keys are assigned in the following manner. A select key bit $K^j_{i,sel}$ (for some $i \in [d], j \in [n]$) is set to 0 for a signal that requires routing from one layer to another and 1 for a signal that requires fixation for constructing logic gates or buffers as illustrated in Figure 7 (step 19, Algorithm 3). In the first case, the corresponding value of configuration key bit $K^j_{i,con}$ becomes "don't care" (step 14, Algorithm 3). In the second case, the $K^j_{i,con}$ is assigned as required depending on the pseudorandom $K^j_{i,r}$ bit (step 15-18, Algorithm 3). For unutilized S-Boxes in a round, random binary values are assigned to $K_{i,con}$

and $K_{i,sel}$ (step 20, Algorithm 3). Note that embedding of ckt in $\mathcal{E}$ is a randomized procedure where randomization is introduced in the following manner: (i) the algorithm randomly chooses which S-Boxes among $\lceil \frac{n}{s} \rceil$ S-Boxes are utilized as gates/buffers in the first cipher round that embeds the first topological level of ckt, (ii) any one way among multiple ways to realize a specific gate/buffer from an S-Box is chosen randomly, (iii) round key is chosen randomly for utilized S-Boxes, (iv) the select and configuration keys are randomly chosen for unutilized S-Boxes, and (v) number of appended buffer layers may vary depending upon choices of previous randomized steps.

E Encoding $\mathcal{E}'$ as SAT problem

In logical cryptanalysis, we express the encryption function of a block cipher through a straightforward propositional formulation denoted as $Enc(\mathbf{P}, \mathbf{K}, \mathbf{C})$, where $\mathbf{P}$, $\mathbf{K}$, and $\mathbf{C}$ represent individual propositional variables that correspond to the plaintext, key, and ciphertext of the cipher, respectively. The propositional formula attains a truth value of "true" (i.e., the corresponding bit evaluates to 1) exclusively when the Boolean assignments assigned to $\mathbf{P}$, $\mathbf{K}$, and $\mathbf{C}$ satisfy the mapping defined by the block cipher. We will now explore how standard cryptanalysis-based attacks on block ciphers, involving the task of key recovery can be represented and characterized using the function $Enc(\mathbf{P}, \mathbf{K}, \mathbf{C})$ [MM00]. Consider the following scenario: in a known plaintext attack, we possess knowledge of the Boolean values v_p representing the plaintext $\mathbf{P}$ and the corresponding ciphertext values (v_c). In this particular context, our objective is to identify a suitable assignment for $Enc(v_p, \mathbf{K}, v_c)$. In this scenario, it is crucial to emphasize that only the value of $\mathbf{K}$ remains unknown.

If q pairs of plaintext-ciphertext are available, the key search can be formalized by conjoining the corresponding formulae [MM00]:

$$\bigwedge_{i=1}^{q} Enc(v_p^i, \mathbf{K}, v_c^i) \tag{2}$$

In logical cryptanalysis, this approach allows us to represent a block cipher as a SAT problem. When attacking a cipher through a known plaintext attack, the objective is to discover a suitable assignment for $\mathbf{K}$ within the abovementioned formulae.

A fundamental requirement in the logical cryptanalysis of a block cipher is the transformation of the intermediate operations within the encryption process into a propositional representation. This conversion enables the representation of the logical characteristics inherent to the underlying block cipher. All the intermediate stages of a single round in an SPN-based augmented block cipher $\mathcal{E}'$ are illustrated in Figure 14. In the following, we provide a comprehensive elucidation of the process of converting these intermediate operations within $\mathcal{E}'$, which are employed in the HardCODE construction.

In our approach to expressing an SPN-based block cipher through propositional logic, we adopt a systematic methodology involving transforming each cipher operation into an equivalent propositional formula. This approach aligns with the principles described in [MM00], which outlines an effective strategy for obtaining a manageable set of formulae for cryptographic analysis. Instead of explicitly encoding every operation within the formulae, we employ a preprocessing step to directly integrate the outcomes of certain operations into the final formulae. Specifically, we abstain from explicitly incorporating the block cipher's bit permutation operation within the formulae.

Encoding the *MUX Layer*. To integrate the MUX layer into the SAT formulation of HardCODE, we generate CNF expressions for the MUX layer using the Tseytin transformation [Tse83] of logic gates. The functional operation of a MUX expressed as $O = SA + \overline{S}B$, where A and B represent the inputs and S denotes the select line, leads to the following CNF expression:

$$(S \vee \overline{A} \vee O) \wedge (S \vee A \vee \overline{O}) \wedge (\overline{S} \vee \overline{B} \vee O) \wedge (\overline{S} \vee B \vee \overline{O}) \tag{3}$$

To encode all n MUX operations of a single MUX layer, n such formulae are ANDed.

Encoding the *addRoundKey*. The *addRoundKey* in a block cipher is a crucial layer in the encryption process. In this layer, a round-specific key, also known as a round key, is bitwise combined (using XOR) with the current state of the plaintext. Each round of encryption uses a different portion of the secret key, which ensures a layer of security in the encryption process, making it challenging for attackers to discern the original key. Utilizing the Tseytin transformation [Tse83] of XOR gate results in the following CNF expression:

$$(\overline{A} \vee \overline{B} \vee \overline{O}) \wedge (A \vee B \vee \overline{O}) \wedge (A \vee \overline{B} \vee O) \wedge (\overline{A} \vee B \vee O) \tag{4}$$

Where A and B are inputs and O in the XOR output. We conjoin n such CNF formulae to encode a single layer of *addRoundKey*.

Encoding the *sBoxLayer*. To encode the *sBoxLayer* of an SPN-based block cipher into a CNF formula suitable for solving the 3-SAT problem, we leverage the inherent relation between the S-Box's input and output bits, which can be expressed through propositional logic. This encoding process is particularly straightforward due to the generalized relationship that exists between the output bits and input bits of the S-Box.

Formally, for a block cipher with d rounds, each of which is characterized by a n-bit intermediate state and a total of $\lceil \frac{n}{s} \rceil$ $s \times s$ S-Boxes in *sBoxLayer*, our methodology for constructing CNF formulas for 3-SAT problem resolution is as follows.

$$S_j = \bigwedge_{j=1}^{s} M_j; \quad SB_j^i \quad = \bigwedge_{j=1}^{\lceil \frac{n}{s} \rceil} S_j, \forall i \in [d] \tag{5}$$

where M_j is the set of conjunctive normal form (CNF) clauses for each output bit of the j^{th} S-Box, composed of its input bits, whereas, SB_j^i represents the formulae combining all the S-Boxes present in the i^{th} round of the cipher. Hence, SB_j^i constitutes the final formula of *sBoxLayer* for the i^{th} round.

Encoding the *pLayer*. In the context of SPN-based block ciphers, the permutation operation governing the rearrangement of bits follows a generalized formula. This formula accepts the input bit position of an intermediate state bit as its argument and produces the corresponding permuted bit position as its output. An effective approach for encoding this permutation operation involves the translation of the generalized formula into its corresponding Conjunctive Normal Form (CNF) clauses.

As previously mentioned, we incorporate a pre-processing step to streamline the final formulae derived from the encoding of the SPN-based block cipher. This step integrates the permuted state output into the formulae for each round of the cipher [MM00]. The pre-processing step comprises a straightforward mapping of input state variables to their respective output state

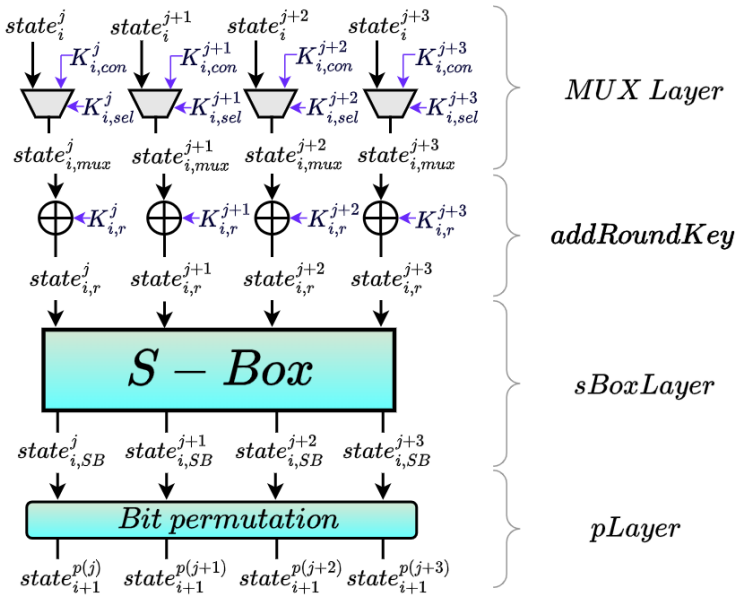


Figure 14: An overview of a single round of HardCODE. Here, $state_i^j$ denotes a signal entering the i^{th} round of augmented SPN at j^{th} bit position, $\forall j \in [n]$. Also, $state_{i,mux}^j$, $state_{i,r}^j$, and $state_{i,SB}^j$ denotes output state variables after the signal passes through MUX layer, *addRoundKey* layer and *sBoxLayer* respectively. Finally, $state_{i+1}^{p(j)}$ is output state variable after signal passes through *pLayer* and enters $(i+1)^{th}$ round of augmented SPN at $p(j)^{th}$ bit position.

variables, guided by their corresponding permuted positions. The actual implementation of the block cipher's permutation is facilitated through a lookup method.

This method utilizes the input positions of the state bits as input and, in turn, produces the permuted output bit positions that correspond to their respective inputs. In this manner, we have effectively translated all the operations within each round of $\mathcal{E}'$ into CNF expressions, thereby enabling its representation as a SAT problem.

E.1 Key recovery from $\mathcal{E}'$.

After converting all the intermediate steps of $\mathcal{E}'$ into CNF formulae, we obtain a SAT instance ready to be fed into a SAT solver.

$\mathcal{E}$ is assumed to be secure and impervious to any known cryptanalytic attack aimed at recovering its key, we assert that the CNF encoding of the intermediate operations (*addRoundKey*, *sBoxLayer*, *pLayer*) of $\mathcal{E}$ using the set of variables $\{K_{i,r}^j\}_{i\in[d],j\in[n]}$ presents a hard instance of the SAT problem. It is important to note that the CNF encoding of $\mathcal{E}'$ utilizes three times the number of variables, incorporating ($\{K_{i,sel}^j\}_{i\in[d],j\in[n]}$, $\{K_{i,con}^j\}_{i\in[d],j\in[n]}$,$\{K_{i,r}^j\}_{i\in[d],j\in[n]}$) as compared to $\mathcal{E}$. Additionally, it introduces an increased number of clauses due to the introduction of an additional Multiplexer (MUX) layer. Formally, Theorem 2 states that the key recovery problem of $\mathcal{E}'$ is hard.

F Logical Cryptanalysis of Augmented Block Cipher $\mathcal{E}'$ - An Example with PRESENT

In this section, we utilize PRESENT [BKL+07] as the underlying SPN-based block cipher $\mathcal{E}$ and formulate the CNF expression of $\mathcal{E}'$, which is required to perform logical cryptanalysis.

To represent $\mathcal{E}'$-instantiated PRESENT as propositional logic, we follow a step-by-step process. We begin with the encoding of the MUX layer and then encode *addRoundKey*, *sBoxLayer*, and *pLayer* one after another. We utilize the Tseytin transformation [Tse83] of logic gates to generate their corresponding CNF expressions. Recall that each of the above layers in PRESENT has 64-bit input and 64-bit output.

1. The *MUX Layer* is the introduced modification in our HardCODE construction. Precisely, we introduce a layer of a keyed MUX before the state bits are XORed with the round key. The state bits first pass through the keyed MUX layer, and the output of the MUX layer is XORed with the round key bits or the *addRoundKey* operation. Hence, the CNF formula for the MUX layer is as follows:

$$\begin{aligned}\bigwedge_{j=0}^{63}(K_{i,sel}^{j} \vee \overline{state_{i}^{j}} \vee state_{i,mux}^{j}) \wedge (K_{i,sel}^{j} \vee state_{i}^{j} \vee \overline{state_{i,mux}^{j}}) \\ \wedge(\overline{K_{i,sel}^{j}} \vee \overline{K_{i,con}^{j}} \vee state_{i,mux}^{j}) \wedge (\overline{K_{i,sel}^{j}} \vee K_{i,con}^{j} \vee \overline{state_{i,mux}^{j}})\end{aligned} \tag{6}$$

   $state_{i,mux}^{j}$ is the output state bits of the keyed MUX layer for the i^{th} round. The MUX's select line ($K_{i,sel}^{j}$) is a key bit, and the configuration bit to set at the S-Box input is also a key ($K_{i,con}^{j}$).

2. The *addRoundKey* operation in PRESENT is a simple process in which the 64−bit state is XORed with the round key ($K_{i,r}$). Formally, the $state_{i,r}$ is obtained by performing the XOR operation between the 64−bit state vector ($state_i$) and the round key ($K_{i,r}^{j}$), where j denotes the state bits ($0 \leq j \leq 63$) and i denotes the number of rounds ($1 \leq i \leq 32$). By applying the Tseytin transformation of the XOR operation to the aforementioned formula, we can produce the following CNF expression for the *addRoundKey* operation of PRESENT as follows.

$$\begin{aligned}\bigwedge_{j=0}^{63}(\overline{state_{i}^{j}} \vee \overline{K_{i,r}^{j}} \vee \overline{state_{i,r}^{j}}) \wedge (state_{i}^{j} \vee K_{i,r}^{j} \vee \overline{state_{i,r}^{j}}) \wedge \\ (state_{i}^{j} \vee \overline{K_{i,r}^{j}} \vee state_{i,r}^{j}) \wedge (\overline{state_{i}^{j}} \vee K_{i,r}^{j} \vee state_{i,r}^{j})\end{aligned} \tag{7}$$

   However, incorporating the MUX layer, utilizing $state_{i,mux}^{j}$ in formula 7, gives the CNF formula for the *addRoundKey* function of the PRESENT with the MUX layer required for HardCODE.

$$\begin{aligned}\bigwedge_{j=0}^{63}(\overline{state_{i,mux}^{j}} \vee \overline{K_{i,r}^{j}} \vee \overline{state_{i,r}^{j}}) \wedge (state_{i,mux}^{j} \vee K_{i,r}^{j} \vee \overline{state_{i,r}^{j}}) \\ \wedge(state_{i,mux}^{j} \vee \overline{K_{i,r}^{j}} \vee state_{i,r}^{j}) \wedge (\overline{state_{i,mux}^{j}} \vee K_{i,r}^{j} \vee state_{i,r}^{j})\end{aligned} \tag{8}$$

Here, $state^{j}_{i,r}$ is the output bit of the *addRoundKey* operation of the PRESENT for each round i.

3. The *sBoxLayer* is the next operation in PRESENT. The 4×4 S-Box in PRESENT can be represented in algebraic normal form (ANF) [SSC+20] as follows:

$$
\begin{aligned}
y_1 &= x_1x_2x_4 \oplus x_1x_3x_4 \oplus x_1 \oplus x_2x_3x_4 \oplus x_2x_3 \oplus x_3 \oplus x_4 \oplus 1 \\
y_2 &= x_1x_2x_4 \oplus x_1x_3x_4 \oplus x_1x_3 \oplus x_1x_4 \oplus x_1 \oplus x_2 \oplus x_3x_4 \oplus 1 \\
y_3 &= x_1x_2x_4 \oplus x_1x_2 \oplus x_1x_3x_4 \oplus x_1x_3 \oplus x_1 \oplus x_2x_3x_4 \oplus x_3 \\
y_4 &= x_1 \oplus x_2x_3 \oplus x_2 \oplus x_4
\end{aligned} \tag{9}
$$

To represent the *sBoxLayer* as SAT problem, we convert the S-Box ANF form to the desired CNF form. HardCODE does not modify the *sBoxLayer* of PRESENT. *sBoxLayer* operates on the output of the *addRoundKey* function ($state^{j}_{i,r}$), and each S-Box in *sBoxLayer* maps it to another set bit $state_{i,SB}$ based on an affine transformation. PRESENT utilizes 16 (4×4) S-Boxes in *sBoxLayer* to operate on 64−bit input $state^{j}_{i,r}$ to produce 64−bit $state_{i,SB}$. The CNF expressions of each output bit of an S-Box are as follows:

$$
\begin{aligned}
\bigwedge_{l=0}^{15} (state^{4l}_{i,SB} \leftrightarrow (\overline{x_{4l+3}} \vee \overline{x_{4l+2}} \vee x_{4l}) \wedge (\overline{x_{4l+3}} \vee \overline{x_{4l}} \vee x_{4l+1}) \\
\wedge (\overline{x_{4l}} \vee \overline{x_{4l+2}} \vee x_{4l+1}) \wedge (x_{4l} \vee \overline{x_{4l+2}} \vee \overline{x_{4l+1}}) \\
\wedge (x_{4l+3} \vee x_{4l+2} \vee x_{4l})) \wedge ((state^{4l+1}_{i,SB} \leftrightarrow (\overline{x_{4l+3}} \vee x_{4l+1}) \\
\wedge (\overline{x_{4l+2}} \vee x_{4l+1}) \wedge (\overline{x_{4l+1}} \vee x_{4l+2} \vee \overline{x_{4l}}) \\
\wedge (\overline{x_{4l+1}} \vee x_{4l+2} \vee x_{4l+3}) \wedge (x_{4l} \vee \overline{x_{4l+3}}) \\
\wedge (x_{4l} \vee x_{4l+1})) \wedge ((state^{4l+2}_{i,SB} \leftrightarrow (\overline{x_{4l+3}} \vee \overline{x_{4l}} \vee \overline{x_{4l+1}}) \\
\wedge (\overline{x_{4l+2}} \vee x_{4l} \vee x_{4l+1}) \wedge (\overline{x_{4l}} x_{4l+2} \overline{x_{4l+3}}) \\
\wedge (x_{4l+1} \vee \overline{x_{4l+2}} \vee x_{4l+3}) \wedge (x_{4l+3} \vee \overline{x_{4l+2}} \vee x_{4l}) \\
\wedge (x_{4l+3} \vee \overline{x_{4l}} x_{4l+1})) \wedge ((state^{4l+3}_{i,SB} \leftrightarrow (\overline{x_{4l+3}} \vee \overline{x_{4l+2}} \vee x_{4l}) \\
\wedge (\overline{x_{4l+2}} \vee \overline{x_{4l}} \vee x_{4l+3}) \wedge (\overline{x_{4l}} \vee \overline{x_{4l+3}} \vee x_{4l+2}) \\
\wedge (\overline{x_{4l}} \vee \overline{x_{4l+3}} \vee \overline{x_{4l+1}}) \wedge (x_{4l+1} \vee x_{4l} \vee \overline{x_{4l+3}}) \\
\wedge (x_{4l+2} \vee x_{4l} \vee \overline{x_{4l+1}}) \wedge (x_{4l+2} \vee x_{4l} \vee x_{4l+3}) \wedge (x_{4l+3} \vee \overline{x_{4l}} \vee x_{4l+1}))
\end{aligned} \tag{10}
$$

The 16 S-Boxes in *sBoxLayer* of PRESENT are denoted using ($0 \leq l \leq 15$) in the above CNF expression. The bits $x_{4l+\alpha}$, such that $\alpha \in \{0, 1, 2, 3\}$ signifies the S-Box input bits. Likewise, each output bit in each round i is denoted using $state^{4l+\alpha}_{i,SB}$ in the above CNF representation.

4. The generalized formula determining the *pLayer* of PRESENT is $p(j) = (16 \times j)\ \%\ 63$. Here j denotes the input bit position, and $p(j)$ denotes the output bit position after applying the *pLayer*. The straightforward approach to encoding this into the CNF formulation is to convert the formula and represent it as CNF clauses. However, the permutation of bits in PRESENT remains fixed, just like most SPN-based block ciphers.

   Hence, to simplify the CNF formula, we implement the *pLayer* as a lookup table such that the input *state* bits of the subsequent round of PRESENT is replaced with the

permuted ($p(state)$) output of the existing layer output. The final equation that is fed to a SAT solver to solve for the secret keys of HardCODE is the result of appending all the operations as mentioned earlier, like *sBoxLayer*, *MUX layer, addRoundKey*, and the *pLayer* of each layer.

Considering this fact, to incorporate the *pLayer* of PRESENT, we replace $state_{i+1}^{p(j)}$ with $state_{i,SB}^{j}$ bits abiding the *pLayer*. We change the j value from round i to $(i+1)$, as per the required permutation $p(j)$, while appending the round-wise CNF clauses to generate the final formula for the SAT solvers.

Comment on the Hardness. As previously mentioned, we will now demonstrate the reduction of $\mathcal{E}'$ to $\mathcal{E}$. In our HardCODE construction, we introduce an additional MUX layer while keeping the other layers of the block cipher unchanged. The CNF expression for this MUX layer, constructed using the Tseytin transformation, has been presented in Equation 6. As discussed earlier, when we set $\{K_{i,sel}^{j}\}_{i\in[d],j\in[n]}$ to 0 and ignore $\{K_{i,con}^{j}\}_{i\in[d],j\in[n]}$ in Equation 6, it simplifies to the following:

$$\bigwedge_{j=0}^{63} (\overline{state_i^j} \vee state_{i,mux}^j) \wedge (state_i^j \vee \overline{state_{i,mux}^j}) \tag{11}$$

Remarkably, this is the CNF expression for a buffer gate. Considering Figure 14, we shall have $state_{i,mux}^{j} = state_i^j$. Consequently, we demonstrate that the system of CNF equations for augmented block cipher $\mathcal{E}'$ ultimately reduces to that of the underlying block cipher $\mathcal{E}$ (in this case, PRESENT). As a result, we can conclude that solving the CNF equations for $\mathcal{E}'$ is as hard as solving for the key in $\mathcal{E}$. To the best of our knowledge, the most potent attack against the PRESENT cipher can only partially recover the round keys up to the 9^{th} round [YLK21]. Thus, if we use a sufficient number of rounds in $\mathcal{E}'$ while embedding a circuit in $\mathcal{E}$, key recovery from $\mathcal{E}'$ by SAT solver is hard.