
AFRL-RY-WP-TR-2021-0035

# 3D SPLIT OF OBFUSCATION, AUTHENTICATION, AND LICENSING (3D-SOUL)

**Avesta Sasan and Kris Gaj**
**George Mason University**

**SEPTEMBER 2021**
**Final Report**



**STINFO COPY**

**AIR FORCE RESEARCH LABORATORY**
**SENSORS DIRECTORATE**
**WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320**
**AIR FORCE MATERIEL COMMAND**
**UNITED STATES AIR FORCE**

# NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with The Under Secretary of Defense memorandum dated 24 May 2010 and AFRL/DSO policy clarification email dated 13 January 2020. This report is available to the general public, including foreign nationals.

Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2021-0035 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

//Signature//

POMPEI L. ORLANDO
Program Manager
Trusted Electronics Branch
Aerospace Components & Subsystems Division

//Signature//

MATTHEW J. MAIER Lt Col, Chief
Trusted Electronics Branch
Aerospace Components & Subsystems Division

//Signature//

LESTER C. LONG, Lt Col, USAF
Deputy Chief
Aerospace Components & Subsystems Division
Sensors Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

*Disseminated copies will show "//Signature//" stamped or typed above the signature blocks.

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE *(DD-MM-YY)* | 2. REPORT TYPE | 3. DATES COVERED *(From - To)* |
|---|---|---|
| September 2021 | Final | 22 January 2018 – 22 September 2020 |

**4. TITLE AND SUBTITLE**
3D SPLIT OF OBFUSCATION, AUTHENTICATION, AND LICENSING (3D-SOUL)

**5a. CONTRACT NUMBER**
FA8650-18-1-7819

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**
62716E

**6. AUTHOR(S)**
Avesta Sasan and Kris Gaj

**5d. PROJECT NUMBER**
N/A

**5e. TASK NUMBER**
N/A

**5f. WORK UNIT NUMBER**
Y1R2

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

George Mason University
3125 Merten Hall
Fairfax, VA 22030

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH 45433-7320
Air Force Materiel Command
United States Air Force

Defense Advanced Research Projects Agency (DARPA/MTO)
675 North Randolph Street
Arlington, VA 22203

**10. SPONSORING/MONITORING AGENCY ACRONYM(S)**
AFRL/RYDT

**11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)**
AFRL-RY-WP-TR-2021-0035

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.

**13. SUPPLEMENTARY NOTES**
This material is based on research sponsored by the Air Force Research Lab (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-1-7819. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Labs (AFRL), the Defense Advanced Research Projects Agency (DARPA) or the U.S. Government. Report contains color.

**14. ABSTRACT**
This report summarizes the 3D Split of Obfuscation, Authentication and Licensing (3D-SOUL) project which developed mechanisms to protect intellectual properties (IPs) that are manufactured in untrusted facilities. It allows entire or parts of IP in an untrusted chip to be obfuscated, while pushing the mechanism for unlocking and secure activation of untrusted chip out to a trusted chip. 3D-SOUL is an obfuscation scheme designed to work with DARPA SPADE architecture, in which an untrusted chip is 3D-stacked on top of a trusted chip, allowing us to benefit from the scalability and speed of untrusted chip, while taking advantage of the security offered by trusted chip. This approach leads to a solution that is using the best features of two chips. A trusted chip is fabricated in a trusted foundry in an older but secure, well protected, and well-understood technology. In this scheme, the trusted chip, encapsulating the sensitive information, verifies the integrity of untrusted chip, performs sensitive logic monitoring, and controls the obfuscation scheme in an untrusted chip. Additional, features developed under this program allows for a light-weight encypted communication channel to be established between the trusted and un-trusted chipsets. This allows an IP to be manufactured in a leading technology, to obtain a faster and more scalable solution, while at the same time alleviating many hardware security concerns.

**15. SUBJECT TERMS**
3D Split of Obfuscation, Authentication and Licensing (3D-SOUL), Trusted, Un-trusted, Encrypted communication, scalable, intellectual property, integrated circuit manufacturing, split manufacturing

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT: | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON (Monitor) |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | Pompei Orlando |
| Unclassified | Unclassified | Unclassified | SAR | 57 | 19b. TELEPHONE NUMBER *(Include Area Code)* N/A |

# Table of Contents

**Section** **Page**

List of Figures ..... ii
List of Tables ..... iii
1 EXECUTIVE OVERVIEW ..... 1
2 MOTIVATION: WHAT INSPIRED 3D-SOUL ARCHITECTURE? ..... 2
3 A LITTLE BACKGROUND (PRIOR ART) ..... 5
4 3D-SOUL ARCHITECTURE OVERVIEW (HIGH LEVEL VIEW) ..... 7
4.1 Double-Cipher Communication (DCC) ..... 9
4.2 Leaky-Cipher Communication (LCC) ..... 10
4.3 Implementation Detail of 3D-SOUL Architecture ..... 10
4.3.1 Configurable Switching Network (CSN) ..... 11
4.3.2 Authenticated Encryption with Associated Data ..... 12
4.3.3 Random Number Generator (RNG) ..... 13
4.3.4 PUF and Secure PUF Readout ..... 13
5 3D-SOUL SECURITY ANALYSIS ..... 15
5.1 Attacker Capabilities ..... 15
5.2 Attacker Objectives ..... 15
5.3 Side-Channel Attack (SCA) ..... 15
5.4 Reverse Engineering ..... 16
5.5 Algebraic Attacks ..... 17
5.6 Counterfeiting and Overproduction ..... 17
5.7 Removal Attacks ..... 17
6 3D-SOUL IMPLEMENTATION RESULTS ..... 18
6.1 3D-SOUL Area Overhead ..... 18
6.2 3D-SOUL Performance ..... 19
6.2.1 3D-SOUL Performance in LCC Mode ..... 19
6.2.2 Frequency of TRN Updates in LCC Mode ..... 21
6.3 Energy Saving in LCC Mode ..... 21
7 COMPARING 3D-SOUL WITH PRIOR WORK ..... 23
8 DESCRIPTION AND ORGANIZATION OF RTL CODE ..... 25
8.1 UL Top Module: Datapath and Controller ..... 27
8.2 3D-SOUL Random Number Generator: A Mix of True and Pseudo RNG ..... 31
8.3 3D-SOUL Random Number Generator: A Mix of True and Pseudo RNG ..... 31
8.4 3D-SOUL Authenticated Encryption with Associated Data (AEAD) ..... 33
8.5 3D-SOUL Configurable Switching and Toggling Network (CSN/RCSN) ..... 34
8.6 3D-SOUL Serialized Communication between the Trusted and Untrusted sides (P2S/S2P) ..... 35
9 3D-SOUL SETUP EXAMPLE ..... 37
10 PUBLICATIONS ..... 41
11 ABSTRACT: ..... 42
12 STUDENT INFORMATION ..... 45
13 REFERENCES ..... 46
LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS ..... 50

# List of Figures

**Figure** **Page**

Figure 1: FORTIS Overall Architecture ..... 4
Figure 2: 3D-SOUL Architecture for 2.5D Package-integrated ICs..... 8
Figure 3: Modes of Encrypted Communication in 3D-SOUL..... 9
Figure 4: 3D-SOUL Architecture. ..... 10
Figure 5: Logarithmic Network ..... 11
Figure 6: The T-test Results for Unprotected (UnPr) and Protected (Pr) Implementations of AES-GCM and ACORN..... 16
Figure 7: Total Execution Time in Number of Clock Cycles for (AES-GCM + AES-CTR) and (ACORN + Trivium)..... 20
Figure 8: The Power Consumption at LCC Mode of Operation..... 22
Figure 9: Energy Breakdown in 3D-SOUL ..... 23
Figure 10: Connection between Boards in 3D-SOUL Project..... 26
Figure 11: Original One-hot Counter FSM (left) and (right) Obfuscated One-hot Counter FSM..... 37

# List of Tables

**Table** **Page**

Table 1. Main Features of the Two Proposed 3D-SOUL Variants ..... 18
Table 2. Resource Utilization of the 3D-SOUL Architecture for NIST-compliant and Lightweight Solution ..... 18
Table 3. Resource Utilization for ASIC Implementation of NIST-compliant and Light weight 3D-SOUL ..... 19
Table 4. Optimized Results of 3D-SOUL Architecture for NIST-compliant and Lightweight Solution ..... 19
Table 5. SAT Execution Time on *OMEGA*-based Blocking CSN and $LOG_{n,log2(n)-2,1}$ as a Close to Non- blocking ..... 21
Table 6. 3D-SOUL vs. FORTIS ..... 24
Table 7. Area Overhead of 3D-SOUL vs. FORTIS ..... 25

# 1 EXECUTIVE OVERVIEW

In this final report, we

- Describe the 3D-SOUL architecture in detail
- Provide a comparison between 3D-SOUL and related prior art, and highlighted how 3D-SOUL has mitigated many of the shortcomings of prior art solutions used for the same or similar purpose
- Provide a thorough analysis of the security of 3D-SOUL
- Provide details on the implementation of the 3D-SOUL and summarized our implementation results
- Describe the RTL code organization and a description for each of the modules designed and used in 3D-SOUL. The RTL code is released along with this final report. The modules described include top module (data path and controller), AXI interconnects, Random Number Generator Solution, Au- thenticated Encryption with Associated Data (AEAD), Configurable Switching Network (CSN), and Serial communication interface (between trusted and untrusted ICs).
- Provide an example, describing how to set up and test 3D-SOUL using two FPGAs. One FPGA is used as a trusted IC (managing activation), and the other FPGA as an untrusted IC (containing 3D-SOUL key-management architecture and obfuscated module).
- Provide a list of publications resulted from this project, with a short description of each publication.
- Provide information about the students who were supported and their contributions to this project.

## 2 MOTIVATION: WHAT INSPIRED 3D-SOUL ARCHITECTURE?

The increasing cost of IC manufacturing has pushed several stages of the semiconductor device's manufacturing supply chain to the third-party facilities, which are identified as untrusted entities [4]. Fabrication of ICs in an untrusted supply chain has introduced multiple forms of security threats such as the possibility of overproduction, Trojan insertion, Reverse Engineering (RE), Intellectual Property (IP) theft, and counterfeiting [34, 33]. The stage that poses the utmost vulnerability is the fabrication stage, in which an untrusted foundry has the ultimate knowledge about a to-be-fabricated IC, and with minimal effort could reverse engineer the *GDSII* to its gate-level netlist, analyze, copy, and/or alter the design, creating trust and security challenges for the original design house.

Considering that a foundry has the ultimate knowledge about the design, passive protection techniques such as watermarking, IC metering, or camouflaging [51, 1, 43, 28] are not well suited to protect against attacks initiated at this stage of the supply chain, although they can be used to either identify counterfeits or prevent reverse engineering of the manufactured ICs post fabrication. To protect the IP from being reverse engineered, overproduced, or stolen in the manufacturing supply chain, researchers have studied various means of hardware obfuscation [28, 20, 18, 21, 37, 17, 36, 52, 35], which is the process of hiding the true functionality of an IC when no key, or an incorrect key, is present. Only once the correct key is provided, will the IC behave correctly. The requirement for obfuscated solutions is to resist various forms of attack against such circuits including brute force, sensitization, Boolean satisfiability (SAT) or satisfiability modulo theories (SMT), removal, approximate-based, signal probability skew, functional analysis, etc. [26, 41, 29, 38, 11, 30, 46, 40].

To remain hidden, in addition to resisting the attacks against its obfuscated circuit(s), the IC should also resist passive, active, or destructive attacks that could be deployed to read the key values. Hence, neither the activation of such devices nor the storage of key values in them should expose or leak the key information. Activation of an obfuscated IC requires storing the activation key in secure and tamper-proof memory. [42, 25]. However, there exist a group of applications that could use alternative key storage. This alternative solution is to store the key outside the IC, where the IC is activated every time it is needed. This option requires constant connectivity to the key management source and secure communication for key exchange to prevent any leakage of the key. This solution allows an IC designer to store the chip unlock key outside of an untrusted chip. So, no secure and tamper-proof memory is needed. Since the key is stored outside the untrusted chip, constant connectivity to an obfuscation key-management solution is an indispensable requirement for this category of devices. This requirement could be easily met for 2.5D package-stack devices where a single trusted chip is used for key management and activation of multiple obfuscated ICs manufactured in untrusted foundries.

In 2.5D package-integrated ICs, like DARPA SPADE architecture [10], a chip which is fabricated in a trusted foundry, but in a larger technology node, is packaged with an untrusted chip fabricated in an untrusted foundry in a smaller technology node. The resulting solution benefits from the best features of both technologies: The untrusted chip may be used as an accelerator, providing the resulting hybrid solution with the much-needed scalability (higher speed and lower power), while the trusted chip provides the means of trust and security. The untrusted chip is isolated from the outside world and any exchange of information to/from untrusted chip passes through the trusted chip. This removes the need for implementing a tamper- proof memory, for storage of IC activation key, in an untrusted process.

Some reasons why implementing a secure memory in an untrusted foundry may be undesired, or practically unfeasible include:

- Availability: The targeted foundry may not offer the required process for implementing a secure memory with the desired features. An example could be the requirement for storing sensitive information in magnetic tunnel junction (MTJ) memories to prevent probing and attacks that could be deployed against flash based NVMs. Fabricating such ICs requires a hybrid process that supports both CMOS and MTJ devices, which may be unsupported by the targeted foundry.
- Verified Security: The secure memory may be available in the targeted technology, nonetheless may not be fully tested and verified in terms of its resistance against different attacks.
- Cost: Implementing secure memory requires additional fabrication layers and processing steps, increasing the cost of manufacturing. Increasing the silicon area is a far cheaper solution than increasing the number of fabrication layers.
- Reusability: In 2.5D system solutions, a trusted chip could be shared by multiple untrusted chips, manufactured in different foundries. Moving the secure memory to the trusted chip removes the need for implementing the secure memory in all utilized processes. The trusted chip could be designed once with the utmost security for the protection and integrity of data. This also reduces the cost of manufacturing untrusted chips by removing the need for additional processing steps for implementing secure memory.
- Ease of Design: Implementing secure memory requires pushing the design through non-standard physical design flow to implement the tamper-proof layers in silicon and package. In addition, the non- volatile nature of tamper-proof memory requires read and write at elevated voltages, increasing the burden on the power-delivery network design. Reuse of a trusted chip with a tamper-proof memory that could manage activation of other obfuscated ICs, relaxes the design requirement of ICs to standard physical design and fabrication process.

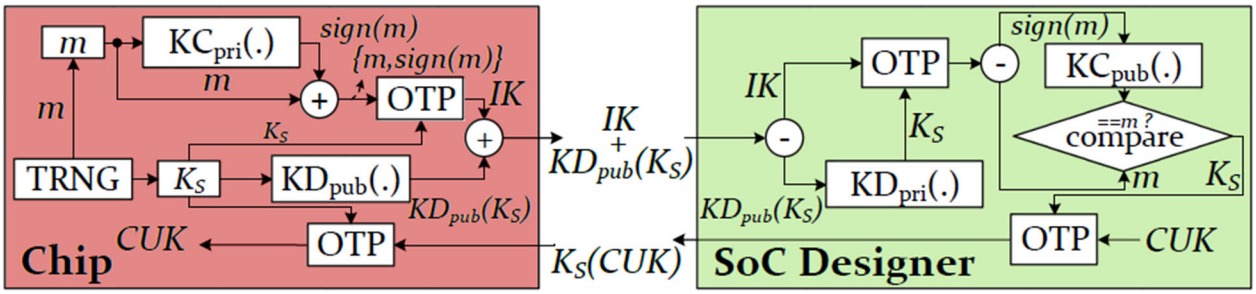



**Figure 1: FORTIS Overall Architecture**

*Comparable prior art*

With 3D-SOUL architecture, we have secured the key-management and communication for the secure activation of obfuscated circuits that are manufactured in untrusted foundries. The 3D-SOUL architecture allows us to (1) push the obfuscation key and obfuscation unlock mechanism off an untrusted chip to the trusted chip, (2) make the key a moving target by changing it for each unlock attempt, (3) uniquely identify each IC, (4) remove the need to implementing a secure memory in an untrusted foundry, and (5) utilize two novel mechanisms for ultra-secure or ultra-fast encrypted communication.

# 3 A LITTLE BACKGROUND (PRIOR ART)

Active metering, Secure Split-Test, logic obfuscation, and solutions such as Ending Piracy of Integrated Circuits (EPIC) have been proposed to protect ICs from supply chain-related security threats by initializing the HW control to a locked state at power-up and hiding the design intent [51, 28, 44, 23, 15, 2, 27, 16]. Some of these techniques support single activation, while others support active metering mechanisms. Active metering techniques [51, 44, 15, 27] provide a mechanism for the IP owner to lock or unlock the IC remotely. In these solutions, the locking mechanism is a function of a unique ID generated for each IC, possibly and preferably by a Physical Unclonable Function (PUF) [42]. Only the IP owner knows the transition table and can unlock the IC. Active metering, combined with a PUF, makes the key a moving target from chip to chip. However, there exist a few issues with previous metering techniques: first, the key(s) to unlock each IC remains static. Second, these techniques unlock the chips before they are tested by the foundry. Hence, the IP owner can control how many ICs enter the supply chain, but not how many properly tested ICs exit the supply chain. Finally, these techniques do not respond well to the threat of the foundry requesting more IDs by falsifying the yield to be lower during the test process. Such shortcomings can potentially allow the foundry to ship more out-of-spec or defective ICs to the supply chain.

Many of these shortcomings were addressed in FORTIS [48], shown in Fig. 1. In FORTIS the registers that hold the obfuscation key are made a part of the scan chain, allowing the foundry to carry structural test by assigning test values to these registers prior to the activation of the IC. Authors of [48] argue that placing a DFT compression logic, not only reduces the test size but also prevents the readout of the individual register values. After testing the IC, the obfuscation key is transferred and applied to unlock the circuit using two types of cryptographic modules: a public-key crypto engine, and a One Time Pad (OTP) crypto engine.

In FORTIS, the public and private keys are hardwired in the design. A TRNG is used to generate a random number (m) that is treated as a message. This message is encrypted using the private key of the chip to generate a signature sig(m). The actual message and its signature are concatenated and later used as a means for the authentication of the chip. At the same time, the TRNG generates another random number, $K_S$. This random number is used as the key for OTP, and at the same time is encrypted using the public key of the designer to generate $KD_{pub}(K_S)$. OTP uses $K_S$ for encrypting the $(m, sig(m))$, and the output of OTP is concatenated with the $KD_{pub}(K_S)$. The resulting string of bits is transmitted to the SoC designer. The SoC designer uses an OTP to obtain $m$ and $sig(m)$ for the purpose of authentication. She then uses the private key of the designer to recover $K_S$. Finally, $K_S$ is used by OTP to encrypt the chip unlock key (CUK). The encrypted CUK is transmitted to the chip, decrypted using OTP, and applied to the obfuscation unlock key registers to unlock the circuit.

FORTIS, however, suffers from several security issues including 1) using identical public and private keys in all manufactured chips, and thus its inability for unique device authentication, 2) being vulnerable to modeling attack in which the FORTIS structure is modeled in software for requesting the CUK from SoC designer 3) being vulnerable to side-channel attacks on public-key encryption engine aimed at recovering the private key of the chip, 4) being vulnerable to fault attacks in which the value of $K_S$ is fixated, 5) requiring a secure memory for storage of the obfuscation unlock key, and 6) not addressing the mechanism for generating a unique and truly random seed to initialize PRNG. All these vulnerabilities are addressed in the 3D-SOUL architecture.

The 3D-SOUL architecture fits in the category of active metering techniques. In 3D-SOUL:

- The key is neither static nor stored in the untrusted chip.
- A key that is used to activate the IC at the test time cannot be reused to activate the same or a different IC in the future. Hence, the test facility can accomplish the test process using ATPG tools with a key which is valid for structural/functional test and is not valid for any subsequent activation.
- The communication to/from IC is secured using a side-channel protected cryptographic engine, combined with a dynamic switching and inversion structure that enhances the security of the chip against invasive and side-channel attacks.
- 3D-SOUL provides two useful means of secure communication to/from the untrusted chip, one for added security, and one for supporting a higher throughput.

The 3D-SOUL architecture is a comprehensive solution for the key management of the obfuscated IPs, where the challenges related to the activation of the IC and secure communication to/from the IC are addressed at the same time. However, as discussed earlier, it is not a universal solution and would fit within the context of 2.5D package-integrated solutions, as 3D-SOUL key management solution requires constant connectivity.

# 4 3D-SOUL ARCHITECTURE OVERVIEW (HIGH LEVEL VIEW)

The primary goal of the 3D-SOUL is to remove the need for storing the obfuscation key (OK) on an untrusted chip while securing the communication flow used for activation of the obfuscated circuit in the untrusted chip. The additional benefits of the proposed architecture are the implementation of two new modes of 1) highly secure and 2) very high-speed encrypted communication. Figure 2 captures the overall architecture of the 3D-SOUL architecture.

The DARPA SPADE project [10] explores solutions in which an overall system is split-manufactured between two different technologies, in this solution, a trusted IC, which is constructed in an older yet secure technology, is packaged with an IC fabricated in an untrusted foundry in advanced geometry. The purpose of this solution is to provide the better of two worlds: the security of older yet trusted technology and the scalability, power, and speed of the newer yet untrusted technology. The 3D-SOUL is designed to work with an architecture like the DARPA SPADE architecture. The proposed solution allows an entire or partial IP in an untrusted chip to be obfuscated, while pushing the mechanism for unlocking and secure activation of the untrusted chip out to a trusted chip. In this solution, the trusted chip encapsulates the sensitive information, verifies the integrity of the untrusted chip, performs sensitive logic monitoring, and controls the activation of the untrusted chip. Also, the key to unlock the obfuscated circuit changes per activation, details of which will be explained shortly.

As shown in Figure 2, the 3D-SOUL architecture contains two main parts, the trusted side (green) and the untrusted side (red). In 3D-SOUL, only the trusted chip is equipped with a secure memory. The secure memory stores the OK and the Secret Key (SK) used for encrypted communication between the trusted and untrusted chips. The SK is generated using a PUF in the untrusted chip, thus it is unique for each untrusted chip, and the untrusted chip does not need a secure memory to store the SK. The CSN and Reverse CSN (RCSN) are logarithmic routing and switching networks. They can permute the order and possibly inverting the logic levels of their primary inputs while these signals are being routed to different primary outputs. The RCSN is the exact inverse of the CSN. Hence, passing a signal through CSN-RCSN (or RCSN-CSN) will recover the original input. The switching and inversion behavior of CSN-RCSN is configured using a True Random Number (TRN). This TRN is generated in the trusted chip to avoid any potential weakening/manipulating of the TRNG. In addition, since the TRNG in 3D-SOUL is equipped with standard-statistical-tests applied post-fabrication, such as Repetition-Count test and the Adaptive-Proportion test, as described in NIST SP 800-90B [12], any attempt at weakening the TRNG during regular operation (i.e., fault attack) can be detected by continuously checking the output of a source of entropy for any signs of a significant decrease in entropy, noise source failure, and hardware failure. By using TRN for the CSN-RCSN configuration, any signal passing through the CSN is randomized, and then by passing through the RCSN is recovered. Additional details are provided in section 4.1.1.

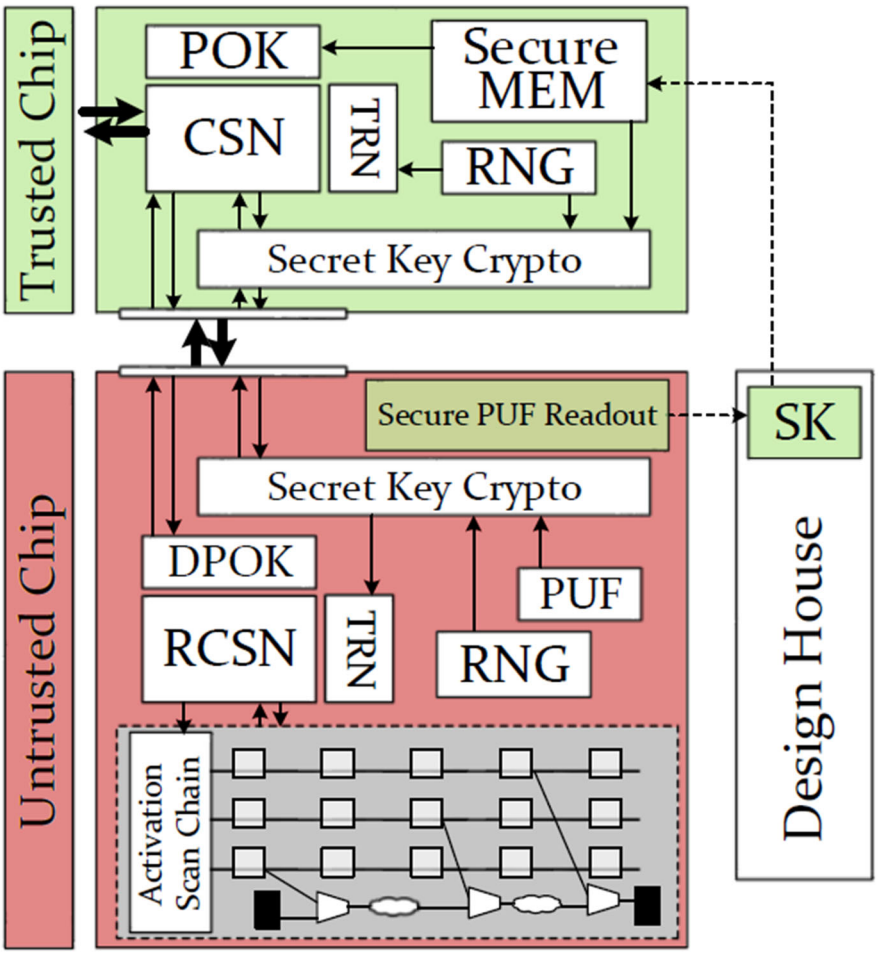
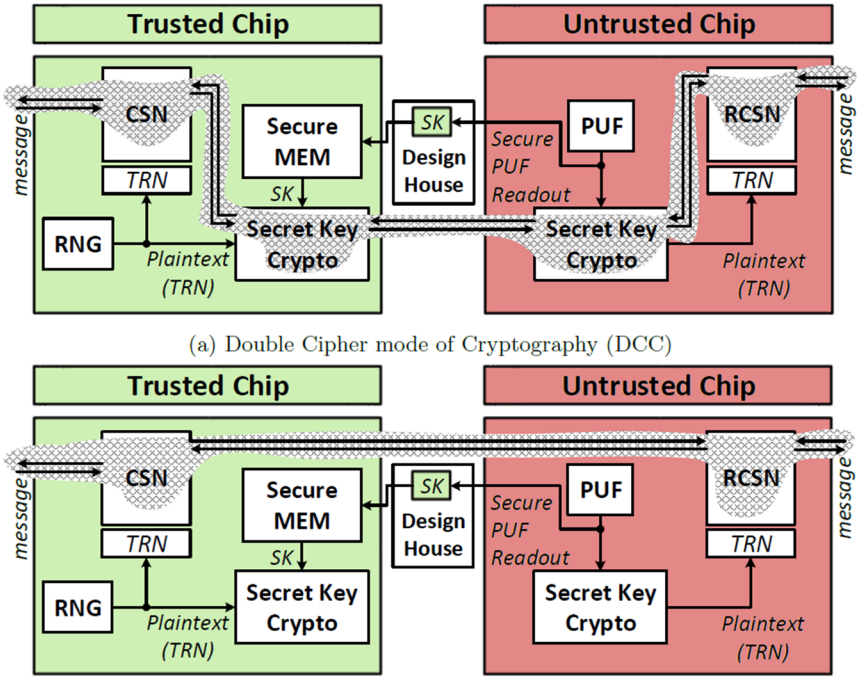



**Figure 2: 3D-SOUL Architecture for 2.5D Package-integrated ICs**

The untrusted chip unlock process in 3D-SOUL is as follows: Prior to each activation, the CSN and RCSN are configured with the same TRN. Since the SK is a PUF-based key generated at the untrusted side, first the SK must be securely read out from the untrusted chip. This is done by deploying public-key cryptography, the details of which are described in section 4.1.4. Then, the trusted chip encrypts the TRN using the SK and sends it to the untrusted chip. To perform an activation, as shown in Fig. 2, the OK is read in segments, denoted as Partial Obfuscation Key (POK), and is passed through the CSN and encryption on the trusted side and the decryption and RCSN on the untrusted side. This process is repeated every time the obfuscated circuit in the untrusted chip is to be activated, each time using a different TRN for configuring the CSN-RCSN. Usage of a different TRN as the configuration input for the CSN-RCSN for each activation randomizes the input data to the Secret key crypto engine. Hence, by using a different TRN for each activation, the obfuscation key (after passing through CSN) is transformed into a one-time license, denoted as Dynamic Activation License (DAL). Since the OK is read and sent in segments (from the trusted chip), the DAL will be received (at untrusted chip) in segments, denoted as Dynamic Partial Obfuscation Key (DPOK), shown in Fig. 2, and is used as an input to RCSN. Passing DPOKs through RCSN recovers the POKs and concatenating the POKs will generate the OK. Note that the DAL is only valid until the TRN is changed. So, the DAL cannot be used to activate other chips or the same chip at a later time.

In 3D-SOUL, the untrusted chip(s) is used as an accelerator, and for safety reasons should not be able to directly communicate to the outside world. Hence, all communication to/from the untrusted chip must go through the trusted chip. In addition, it is possible that the computation, depending on the sensitivity of processed data, is divided between the trusted and untrusted chips. Hence, there is a need for constant communication between the trusted and untrusted chips. The communication needed is sometimes for limited but highly sensitive data, and sometimes for vast amounts of less sensitive data. As illustrated in Fig. 3, the proposed architecture is designed to provide two hybrid means of encrypted communication:

- Double-Cipher Communication (DCC) as ultra-secure communication
- Leaky-Cipher Communication (LCC) as ultra-fast communication mechanism.




(a) Double Cipher mode of Cryptography (DCC)




**Figure 3: Modes of Encrypted Communication in 3D-SOUL**

### 4.1 Double-Cipher Communication (DCC)

As shown in Figure 3(a), DCC is implemented by passing each message through both CSN-RCSN and the secret key cryptography engine, where the TRN used in CSN-RCSN is renewed every $U$ cycles. DCC provides the ultimate protection against side-channel attacks. In DCC mode, two necessary requirements for mounting a side-channel attack are eliminated. The side-channel attack aims to break the cryptography system by analyzing the leaked side-channel information for different input patterns. Hence, (1) the degree of correlation between the input and the leaked side-channel information, and (2) the intensity of side-channel variation, are important. In 3D-SOUL, the attacker cannot control the input to the secret-key cryptography. In addition, the input to the CSN is randomized using a TRN and then passed to the secret-key cryptography, removing the correlation between leaked side-channel info (from secret-key cryptography) and the original input to the CSN. Additionally, the secret-key cryptography engine is side-channel protected to pass a t-test [6]. So, the intensity and variation in side-channel information are significantly reduced, making the DCC an extremely difficult attack target.

### 4.2 Leaky-Cipher Communication (LCC)

LCC is a fast and energy-efficient mode of communication between the trusted and the untrusted chip. As illustrated in Figure 3(b), in this protocol, the CSN-RCSN pair is used for exchanging data. The secret key cryptography engine is used to transmit a TRN from one chip to the other. Since the throughput of TRNG is the bottleneck point compared to the performance of CSN-RCSN, the TRNG is used as a seed generator to the PRNG (which offers higher performance) on both sides, Hence, in LCC mode, PRNG is used to configure the CSN-RCSN to avoid any performance degradation on transmitting data. For $U$ consecutive cycles, the PRNG is kept idle allowing the CSN to use the same PRNG output for $U$ cycles. It not only reduces the power consumption of PRNG and TRNG, but it also provides faster communication in LCC mode. However, using this model of communication is prone to algebraic and SAT attacks as each communicated message leaks some information about the TRN used to configure the CSN-RCSN pair. If an attacker can control the message and observe the output of the CSN, each communicated message leaks some information about the key, reducing its security. Extracting the key from such observations is possible by various attack models, including Satisfiability attacks. Hence, an attacker with enough time and enough traces could extract the TRN and retrieve the communicated messages. Preventing such attacks poses a minimum limit to $U$ (the update frequency of the PRNG). $U$ should be small to prevent SAT and other trace-based learning or analysis attacks, but large enough to be energy efficient. In Section 6, we deploy a SAT attack against LCC and will further elaborate on the required TRN update frequency.

### 4.3 Implementation Detail of 3D-SOUL Architecture

Fig. 4 captures the overall architecture of 3D-SOUL and the relation and connectivity of its macros. As discussed, 3D-SOUL supports both key-management and secure data communication. Based on the selected mode of communication (LCC/DCC), the message passes through {CSN → RCSN} or {CSN → encryption→ decryption → RCSN. RNG, which contains both TRNG and PRNG, is used on both sides. In the trusted chip, RNG is used for implementing the side-channel protected cryptography engine, as well as generating the configuration of the CSN-RCSN (TRN). On the untrusted side, it is used only for implementing the side-channel protected cryptography engine. Finally, PUF is engaged in the untrusted chip for both unique IC authentication and for the generation of the secret key for encryption. As shown in Fig. 4, all modules employ an AXI-stream interface to maximize the simplicity of the overall design and minimize the overhead incurred by the controller of the top module on each side. The description of the behavior of each macro in 3D-SOUL is provided next:

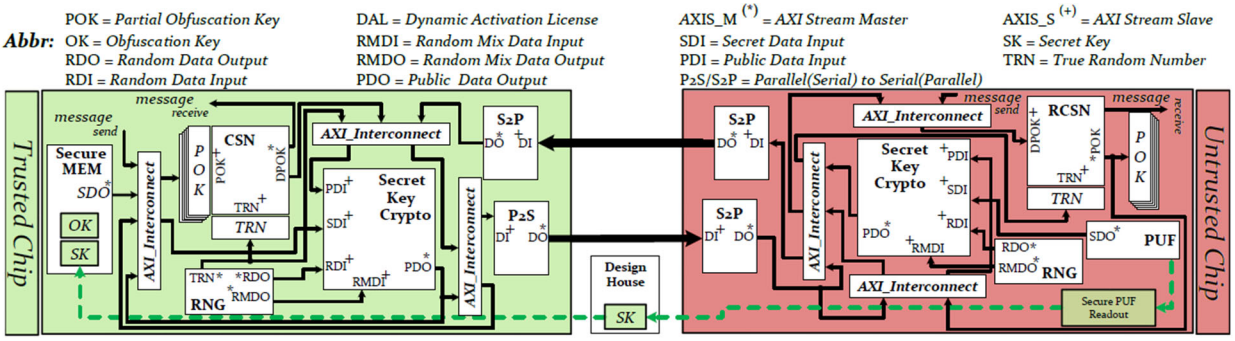



**Figure 4: 3D-SOUL Architecture.**

*The chip that is fabricated in an untrusted foundry (right) and (left): The chip that is fabricated in a trusted foundry.*

### 4.3.1 Configurable Switching Network (CSN)

The CSN is a logarithmic routing network that could route the signals at its input pins to its output pins while permuting their order and possibly inverting their logic levels based on its configuration. Figure 5(a) captures a simple implementation of an 8-by-8 CSN using *OMEGA* [19] network. The network is constructed using routing elements, denoted as Re-Routing Blocks (RRB). Each RRB is able to possibly invert and route each of the input signals to each of its outputs. The number of RRBs needed to implement this simple CSN for $N$ inputs ($N$ is a power of 2) is simply $N/2\ log\ (N)$. Each CSN should be paired with an RCSN. The RCSN is constructed by flipping the input/output pins of RRB and treating the CSN input pins as its output pins and vice versa.

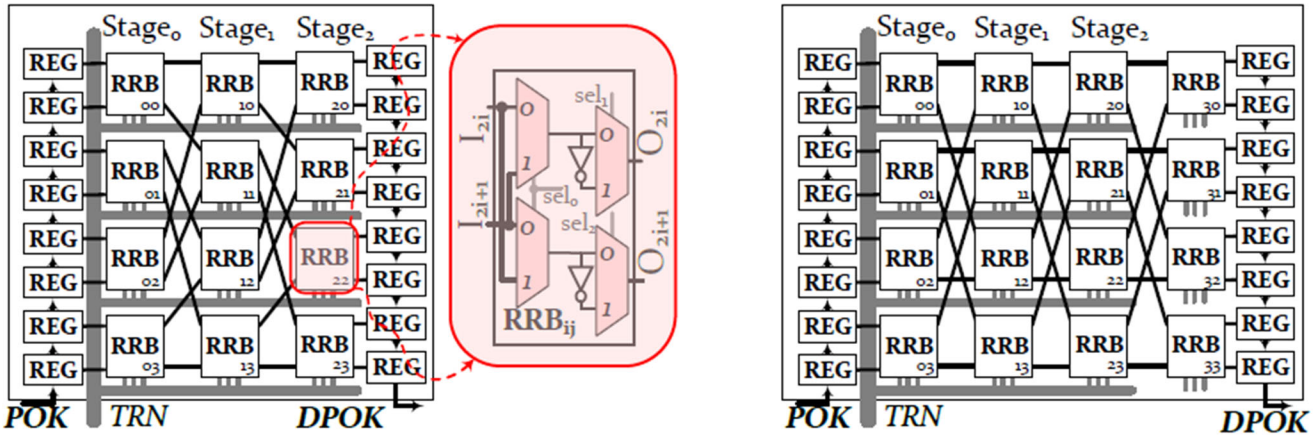



**Figure 5: Logarithmic Network**
*Omega-based Blocking (left) and (right) $LOG_{8,1,1}$ near Non-blocking*

The OMEGA network along with many other networks of such nature (Butterfly, etc.) are blocking networks [19], in which we cannot produce all permutations of input at the network's output pins. This limitation significantly reduces the ability of a CSN to randomize its input. Also, we will show that a blocking CSN can be easily broken by a SAT attack within relatively few iterations.

Being a blocking or a non-blocking CSN depends on the number of stages in CSN. Since no two paths in an RRB can use the same link to form a connection, for a specific number of RRB columns, only a limited number of permutations is feasible. However, adding extra stages could transform a blocking CSN into a strictly non-blocking CSN. Using a strictly non-blocking CSN not only improves the randomization of propagated messages through the CSN but also improves the resiliency of these networks against possible SAT attacks for extraction of a TRN used as the key for a CSN-RCSN cipher. A non-blocking logarithmic network could be represented using $LOG_{n,m,p}$, where $n$ is the number of inlets/outlets, m is the number of extra stages, and p indicates the number of copies vertically cascaded [9].

According to [9], to have a strictly non-blocking CSN for an arbitrary n, the smallest feasible values of p and m impose very large area/power overhead. For instance, for n = 64, the smallest feasible values, which make it strictly non-blocking, are m = 3 and p = 6, which means there exists more than 5 as much overhead compared to a blocking CSN with the same n, resulting in a significant increase in the area and delay overhead. To avoid such large overhead, we employ a close to non-blocking CSN

described in [9] to implement the CSN-RCSN pair. This network is able to generate not all, but almost all permutations, while it could be implemented using a $LOG_{n,\log_2(n)-2,1}$ configuration, meaning it needs $\log_2(n)$ 2 extra stages and no additional copy. Fig. 5(b) demonstrates an example of such a close-to-non-blocking CSN with n = 8. In the results section, we demonstrate that using these close-to-non-blocking CSNs enhances the resiliency of a CSN against SAT attack, even in small sizes of CSNs with significantly lower power, performance, and area (PPA) overhead.

### 4.3.2 Authenticated Encryption with Associated Data

The Authenticated Encryption with Associated Data (AEAD) is used in the DCC mode for communicating messages and in the LCC mode for the initial transmission of the CSN-RCSN key (TRN). Authenticated ciphers incorporate the functionality of confidentiality, integrity, and authentication. The input of an authenticated cipher includes Message, Associated Data (AD), Public Message Number (NPUB), and a secret key. The ciphertext is generated as a function of these inputs. A Tag, which depends on all inputs, is generated after message encryption to assure the integrity and authenticity of the transaction. This tag is then verified after the decryption process. The choice of AEAD could significantly affect the area overhead of the solution, the speed of encrypted communication, and the extra power consumption. To show the performance, power, and area trade-offs, we employ two AEAD solutions: a NIST compliant solution (AES-GCM), and a promising lightweight solution (ACORN).

AES-GCM is the current National Institute of Standards and Technology (NIST) standard for authenticated encryption and decryption as defined in [32]. ACORN is one of two finalists of the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR), in the category of lightweight authenticated ciphers, as defined in [22]. An 8-bit side-channel protected version of AES-GCM and a 1-bit side-channel protected version of ACORN are implemented as described in [50]. Both implementations comply with the lightweight version of the CAESAR HW API [13].

Our methodology for side-channel resistance is threshold implementation (TI), which has wide acceptance as a provably secure Differential Power Analysis (DPA) countermeasure [45]. In TI, sensitive data is separated into shares and the computations are performed on these shares independently. TI must satisfy three properties: 1) Non-completeness: Each share must lack at least one piece of sensitive data, 2) Correctness: The final recombination of the result must be correct, and 3) Uniformity: An output distribution should match the input distribution. To ensure uniformity, we refresh TI shares after non-linear transformations using randomness. We use a hybrid 2-share/3-share approach, where all linear transformations in each cipher are protected using two shares, which are expanded to three shares only for non-linear transformations.

To verify the resistance against DPA, we employ the Test Vector Leakage Assessment methodology, defined in [6]. We leverage a" fixed versus random" non-specific t-test, in which we randomly interleave first fixed test vectors and then randomly generated test vectors, leading to two sequences with the same length but different values. Using means and variances of power consumption for our fixed and random sequences, we compute a figure of merit $t$. If $t > 4.5$, we reason that we can distinguish between the two populations and that our design is leaking information. The protected AES-GCM design has a 5-stage pipeline and encrypts one 128-bit input block in 205 cycles. This requires 40 bits of randomness per cycle. In ACORN-1, there are ten 1-bit TI-protected AND-gate modules, which

consume a total of 20 random reshare, and 10 random refresh bits per state update. In a two-cycle architecture, 15 random bits are required per clock cycle.

### 4.3.3 Random Number Generator (RNG)

An RNG unit is required on both sides to generate random bits for side-channel protection of AEAD units, a random public message number (NPUB) for AEAD, and TRNs for CSN-RCSN. We adopted the ERO TRNG core described in [39], which can generate only 1-bit of random data per over 20,000 clock cycles. In our TI implementations, AES-GCM needs 40 and ACORN 15 bits of random data per cycle. So, we employed a hybrid RNG unit combining the ERO TRNG with a Pseudo-Random Number Generator (PRNG). TRNG output is used as a 128-bit seed to PRNG. The PRNG generates random numbers needed by other components. The reseeding is performed only once per activation.

The choice of PRNG depends on the expected performance and overhead. To support 3D-SOUL, we adopted two different implementations of PRNG: (1) AES-CTR PRNG, which is based on AES, is compliant with the NIST standard SP 800-90A and generates 12.8 bits per cycle. (2) Trivium based PRNG, which is based on the Trivium stream cipher described in [7]. The Trivium-based PRNG is significantly smaller in terms of area and much faster than AES-CTR PRNG. It can generate 64 bits of random data per cycle; nonetheless, it is not compliant with the NIST standard.

### 4.3.4 PUF and Secure PUF Readout

The response of the PUF to a challenge selected randomly by Enrollment Authority (SoC designer) is used as the secret key in AEAD. Hence, the readout of the PUF-response should be protected. The simplest solution for the safe readout of a PUF-generated key is to enable the readout by burning one set of fuses and disabling it by burning the second set of fuses. However, this solution, especially when combined with a weak PUF, is not likely to be resistant against the untrusted foundry, which may possibly burn the first set of fuses, read out PUF key, and then repair fuses before releasing the chip. To avoid this problem, we implement a lightweight one-sided public-key cryptography (encryption only) based on Elliptic-Curve Cryptography (ECC). Considering the PUF readout is a one-time event, the performance of the public-key cryptography engine is not critical.

In order to prevent any attempts at fully characterizing a PUF in the untrusted foundry, only strong PUFs, e.g. an arbiter PUF, are considered. The secure readout of the PUF key is allowed only at the device enrollment time, in the secure facility. During the secure readout, the strong PUF is fed with multiple challenges selected by the Enrollment Authority. The corresponding PUF responses are encrypted by the untrusted chip using the public key of the Enrollment Authority that is embedded in the chip layout or stored in the one-time programmable memory. Only the Enrollment Authority has access to the decrypted responses. Afterward, one of the previously applied challenges is randomly selected and used for the generation of the secret key. This challenge is then hardwired on the untrusted chip, and the PUF response to that challenge is recorded by the Enrollment Authority. This PUF response is then stored in the secure memory of the trusted chip. This process makes each PUF key unique to a given device, and resistant to any unauthorized readout by the untrusted foundry.

Still, additional precautions must be taken to protect this scheme against an attack aimed at replacing a real PUF with a pseudo-PUF, generating randomly looking responses that can be easily calculated by an attacker. An example of such a pseudo-PUF may be a lightweight symmetric-key cipher, with a

fixed key known to the untrusted party, encrypting each challenge and outputting a ciphertext as the PUF response. Such pseudo-PUF should be treated as a Trojan and detected by Enrollment Authority using the best known anti-Trojan techniques, e.g., those based on the measurement and analysis of the power consumption during the operation of the device [8]. Additional methods may be used to differentiate the outputs of a strong PUF from encrypted data, e.g., using known correlations between the PUF responses corresponding to closely related challenges, such as challenges differing on only one bit-position or being mutual complements of each other [24]. These kinds of PUF-health tests may be specific to a particular strong PUF type, e.g., to an arbiter PUF and will be the subject of our future work.

# 5 3D-SOUL SECURITY ANALYSIS

## 5.1 Attacker Capabilities

Different sources of vulnerability are considered in this section to demonstrate the 3D-SOUL security. The attacker can be an adversary in the manufacturing supply chain and have access to either the reverse- engineered or design house-generated netlist of the 3D-SOUL-protected untrusted chip. The attacker can purchase an activated 3D-SOUL-protected IC from the market. The attacker can monitor the side channel information of chips at or post-activation. The attacker can observe the communication between untrusted and trusted chip. She could also alter the communicated data.

## 5.2 Attacker Objectives

The Attacker objective may be:

- Extraction of the OK.
- Illegal activation of the obfuscated circuit without extracting the key.
- Extraction of the long-term (SK.
- Extraction of short-term CSN keys (TRNs)
- Eavesdropping on messages exchanged between the untrusted chip and the external sources.
- Removial of the 3D-SOUL protection.
- 3D-SOUL-protected IC overproduction.

## 5.3 Side-Channel Attack (SCA)

The objective of SCA on 3D-SOUL is to extract either the SK used by AEAD or the TRN used by CSN. Extracting a SK is sufficient to break the obfuscation; extracting a TRN reveals only messages sent in the LCC mode.

DCC significantly increases the SCA difficulty, since (1) the AEAD is side channel protected, and (2) the attacker loses access to the input of AEAD. Fig. 6 captures our assessment of side-channel resistance of AEAD using a t-test for unprotected and protected implementations of AES-GCM and ACORN [49]. As illustrated, both implementations pass the t-test, indicating increased resistance against SCA. On the other hand, the inability to control the input to AEAD comes from the 3D-SOUL requirement of encryption in the DCC mode where a message first passes through the CSN. Hence, there exists no relationship between the power consumption of the AEAD and the original input due to CSN randomization. CSN power consumption is also randomized as it is a function of $n$ inputs (possibly known to the attacker) and $3n$ ($log_2 n$ 1) TRN inputs unknown to the attacker, while the TRN is repeatedly updated based on the value of U. Note that during the physical design of 3D-SOUL, the side channel information on power and voltage noise (IR drop) could be further mitigated using timing aware IR analysis [3], and voltage noise aware clock distribution techniques [31, 5].

The LCC mode is prone to side-channel, algebraic, and SAT attacks aimed at extracting the TRN. However, the attack must be carried out in a limited time while the TRN of the CSN/RCSN is unchanged. As soon as the TRN is renewed, the previous side-channel traces or SAT iterations are useless. The period of TRN updates (U) introduces a trade-off between energy and security and can be pushed to maximum security by changing the TRN for every new input. In section 6.2.2 we investigate the time required to break the LCC using side-channel or SAT attack and accordingly define a safe range for $U$ to prevent such attacks.

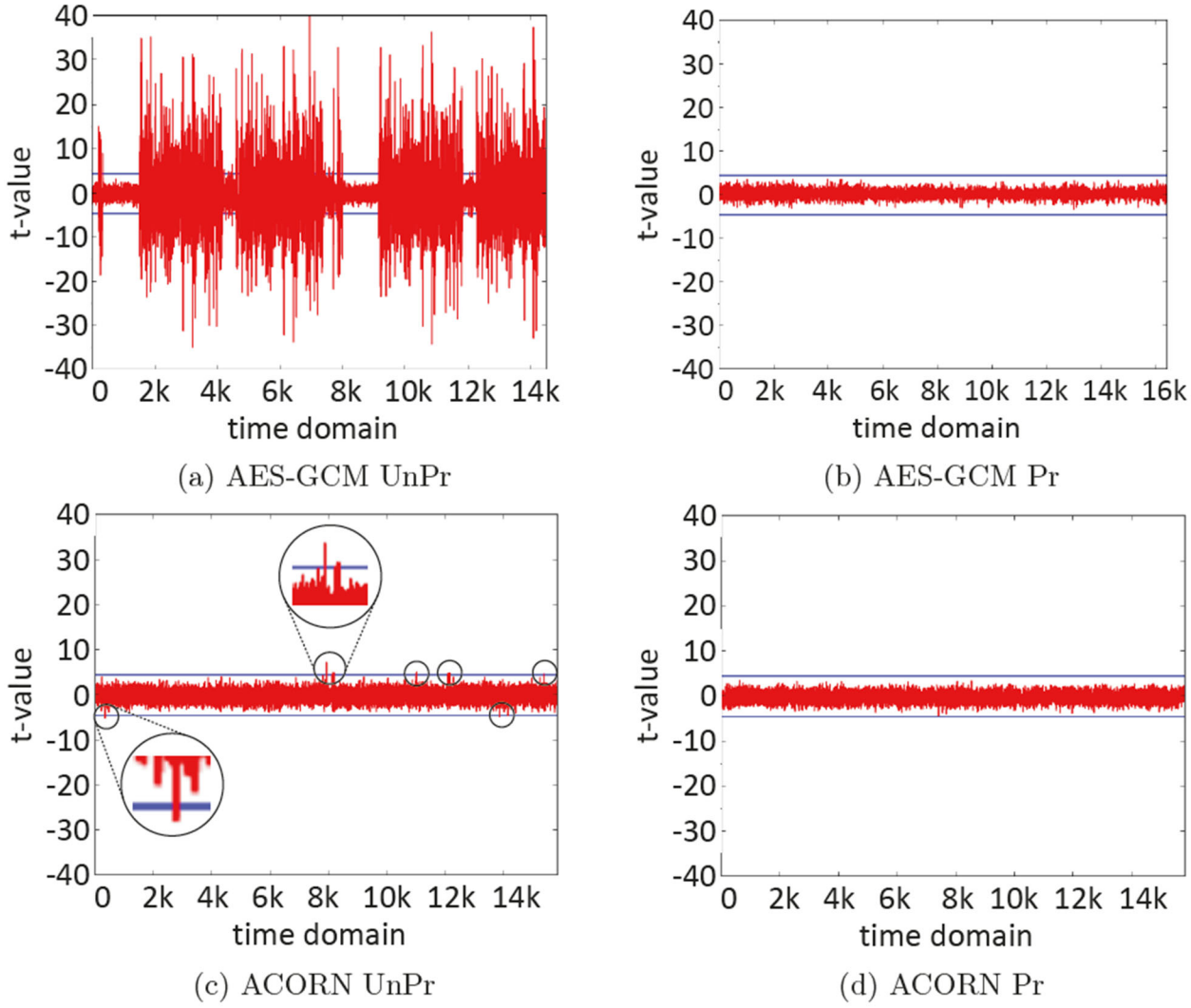



**Figure 6: The T-test Results for Unprotected (UnPr) and Protected (Pr) Implementations of AES-GCM and ACORN**

## 5.4 Reverse Engineering

In 3D-SOUL, RE to extract the secret key from layout is useless as the secret key is not hardwired in the design and is generated based on PUF. RE to extract the key from memory in an untrusted chip is no longer an option as the key is not stored in the untrusted chip. RE to extract the key from the trusted chip's memory is limited by the difficulty of tampering with secure memory in the trusted technology.

### 5.5 Algebraic Attacks

Algebraic attacks involve (a) expressing the cipher operations as a system of equations, (b) substituting in known data for some variables, and (c) solving for the key. AES-GCM and ACORN have been demonstrated to be resistant against all known types of algebraic attacks, including linear cryptanalysis. Therefore, in the absence of any new attacks, the DCC mode is resistant against algebraic attacks. Using CSN and RCSN for fast encryption is new and requires more analysis. CSN can be expressed as an affine function of the data input $x$, of the form $y = A\ x + b$, where $A$ is an $n\ n$ matrix and $b$ is an $n$ 1 vector, with all elements dependent on the input TRN. Although recovering $A$ and $b$ is not equivalent to finding the TRN, it may enable the successful decryption of all blocks encrypted using a given TRN. We protect against this threat in two ways: (1) The number of blocks encrypted using a given TRN is set to the value smaller than $n$, which prevents generating and solving a system of linear equations with $A$ and $b$ treated as unknowns, (2) We partially modify the TRN input of CSN with each block encryption (by simply shifting the input TRN bits), so the values of $A$ and $b$ are not the same in any two encryptions, without the need of feeding CSN with two completely different TRN values.

### 5.6 Counterfeiting and Overproduction

3D-SOUL can be used to prevent the resale of used ICs, usage of illegal copies, and reproduction of a design. During packaging and testing, each 3D-SOUL-protected IC is first tested and then is matched with a trusted chip. So, the untrusted chip can only be activated by the matched trusted chip. Building illegal copies that work without the secure chip and reproduction of the design requires successful RE. Blind reproduction is useless as its activation requires a matching trusted chip. By receiving one or more DALs for testing, the manufacturer cannot activate additional IPs as the DAL changes from activation to activation.

### 5.7 Removal Attacks

Removal of the TRNG fixates the DAL and breaks the LCC mode. In DCC mode, it gives an attacker control over the input to the AEAD, increasing the chances of SCA on the cryptography engine. NIST standard SP 800-90B [12] dictates that continuous health testing must be performed on the TRNG. These tests include repetition counting to detect the catastrophic failure and adaptive proportion testing to detect loss of entropy. Removal of the TRNG would be detected as this would result in insufficient entropy to satisfy the health test, assuming the test is implemented on the trusted side. Removal of 3D-SOUL architectural modules makes the chip non-functional as 3D-SOUL is not a wrapper architecture, but a fused one. Complete removal of 3D-SOUL requires successful RE. Removing the PUF can be made challenging by using a strong PUF, with many challenge-response pairs. Replacing such a PUF with a deterministic function is challenging as such functions are likely to have a substantially different area and power, making them detectable.

# 6 3D-SOUL IMPLEMENTATION RESULTS

For evaluation, all designs have been implemented in VHDL and synthesized for both FPGA and ASIC. For ASIC implementation we used Synopsys generic 32nm educational libraries. For FPGA verification, we targeted a small FPGA board, Digilent Nexys-4 DDR with Xilinx Artix-7 (XC7A100T-1CSG324).

## 6.1 3D-SOUL Area Overhead

We implemented two variants of 3D-SOUL architecture: a NIST compliant solution (denoted by 3D-SOUL1) and a lightweight solution (denoted by 3D-SOUL2). The AEAD and PRNG in 3D-SOUL1 is based on AES- GCM and AES-CTR respectively. The 3D-SOUL2 is implemented by using ACORN for AEAD and Trivium for PRNG, the details of these two variants are summarized in Table 1. The breakdown of area (in terms of Slices, LUTs, and FFs) for these solutions for an FPGA implementation in Xilinx Artix-7 (using Minerva [14]) is reported in Table 2. The breakdown of area (in terms of Cells and $um^2$), critical path, and power consumption for an ASIC implementation is reported in Table 3. Table 4 reports optimized area and frequency results on FPGA for top-level of trusted and untrusted sides. As illustrated, the total area of lightweight solution is around 1/3 of the NIST-compliant solution. The reported numbers in Table 2 include the overhead of all sub-modules including AEAD, CSN-RCSN, RNG, ECC, etc. Due to the optimization on the boundaries among the units, resource utilization in Tables 4 is less than the sum of row values in Table 2.

**Table 1. Main Features of the Two Proposed 3D-SOUL Variants**

| Feature | 3D-SOUL1 | 3D-SOUL2 |
|---|---|---|
| AEAD | AES-GCM | ACORN |
| PRNG | AES-CTR | Trivium |
| BUS Width | 8 | 8 |
| Pins used for Communication | 8 | 8 |
| CSN-RCSN Size | 64 | 64 |
| Trusted Memory | 4 Kbits | 4 Kbits |
| $C_{fix}$: initialization overhead (cycles) | 10,492 | 20,452 |
| $C_{byte}$: cycles needed for encrypting each byte | 72 | 17 |
| $PRNG_{perf}$: Throughput of generating PRN | $128bit/10cycles$ | $64bit/cycle$ |

**Table 2. Resource Utilization of the 3D-SOUL Architecture for NIST-compliant and Lightweight Solution**

| Name | AES-GCM+AES-CTR | | | ACORN+Trivium | | |
|---|---|---|---|---|---|---|
| | Slice | LUT | FF | Slice | LUT | FF |
| Trusted Side | | | | | | |
| AEAD_EXT | 1,336 | 3,804 | 4,432 | 333 | 1,067 | 591 |
| RNG | 712 | 2,226 | 618 | 215 | 601 | 450 |
| CSN | 257 | 540 | 739 | 257 | 540 | 739 |
| Others | 149 | 345 | 144 | 149 | 345 | 144 |
| Untrusted Side | | | | | | |
| AEAD_EXT | 1,336 | 3,804 | 4,432 | 333 | 1,067 | 591 |
| RNG | 738 | 2,352 | 628 | 241 | 683 | 460 |
| RCSN | 252 | 607 | 737 | 252 | 607 | 737 |
| ECC | 563 | 1569 | 1161 | 563 | 1569 | 1161 |
| PUF [47] | 177 | — | — | 177 | — | — |
| Others | 209 | 359 | 257 | 209 | 359 | 257 |

On Xilinx Artix-7 (XC7A100T-1CSG324) FPGA.

### 6.2 3D-SOUL Performance

Figure 7 compares the performance of two solutions in DCC and LCC mode. As illustrated, for small data sizes, the 3D-SOUL1 outperforms the 3D-SOUL2 solution. However, as the size of data increases, the 3D- SOUL2 outperforms the 3D-SOUL1 solution. It is because stream ciphers such as ACORN have a long initialization phase, making them inefficient for small data size. In addition, our AES-GCM implementation benefits from an 8-bit data path, but the ACORN is realized by a 1-bit serial implementation. The total latency in terms of the number of clock cycles for 3D-SOUL1 and 3D-SOUL2 implementations can be calculated using equation (1), in which the number of cycles for the initialization and finalization is fixed and is given in Table 1. The $C_{byte}$ is the number of cycles needed for encrypting each input message byte, which is 17 and 72 for 3D-SOUL2 and 3D-SOUL1, respectively. Hence, in spite of longer initialization, the 3D-SOUL2 outperforms the 3D-SOUL1 for message sizes larger than 128 Bytes.

$$T_{comm} = C_{fix} + Message_{size} \times C_{byte} \quad (1)$$

**Table 3. Resource Utilization for ASIC Implementation of NIST-compliant and Light weight 3D-SOUL**

| Name | AES-GCM+AES-CTR | | | | ACORN+Trivium | | | |
|---|---|---|---|---|---|---|---|---|
| | Cells | $\text{Area}_{um^2}$ | $\text{Tclk}_{ns}$ | $\text{Power}_{mW}$ | Cells | $\text{Area}_{um^2}$ | $\text{Tclk}_{ns}$ | $\text{Power}_{mW}$ |
| **3D-SOUL** | 25338 | 0.11 | 1.97 | 1.62 | 8681 | 0.046 | 1.18 | 0.84 |
| ▷ RNG | 5684 | 0.025 | 1.43 | 0.431 | 1267 | 0.007 | 0.27 | 0.144 |
| ▷ CSN/RCSN | 1749 | 0.008 | 0.08 | 0.11 | 1749 | 0.008 | 0.08 | 0.11 |
| ▷ AEAD | 13675 | 0.061 | 1.67 | 0.704 | 2257 | 0.013 | 0.97 | 0.251 |
| ▷ ECC | 3278 | 0.016 | 1.34 | 0.321 | 3278 | 0.016 | 1.34 | 0.321 |

Using Synopsys generic 32nm libraries

**Table 4. Optimized Results of 3D-SOUL Architecture for NIST-compliant and Lightweight Solution**

| Name | AES-GCM+AES-CTR | | | | ACORN+Trivium | | | |
|---|---|---|---|---|---|---|---|---|
| | Slice | LUT | FF | Freq[MHz] | Slice | LUT | FF | Freq[MHz] |
| Trusted | 2,297 | 7,094 | 5,892 | 103 | 1,030 | 2,901 | 1,924 | 121 |
| Untrusted | 2,818 | 8,781 | 7,169 | 109 | 1451 | 4,182 | 3,156 | 120 |

*On Xilinx Artix-7 (XC7A100T-1CSG324) FPGA.*

#### 6.2.1 3D-SOUL Performance in LCC Mode

In the LCC mode, the AEAD is used to synchronize the initial seed of the PRNG, while the CSN is used for encrypting data. The random (TRN) configuration key for the CSN-RCSN is generated by PRNG, which is updated after transferring every $U$ messages. In 3D-SOUL, the PRNG has a limited

buffer size, and as soon as the buffer is filled with random data, the PRNG stops producing additional bits. The consumption of TRNG output is synchronized (every $U$ messages) and the generation of random inputs is limited by the size of the buffer. Hence, the PRNGs in the trusted and untrusted sides are always in sync. The number of cycles it takes to initialize the LCC mode includes the time to initialize the secret key engine ($C_{fix}$), the encryption and transfer and decryption of PRNG seed ($C_{ENC}$), and the time for the PRNG to generate enough output from a newly received TRN ($C_{PRNG}$):

$$C_{LCC-init} = C_{fix} + C_{ENC} + C_{PRNG} \tag{2}$$

Depending on the AEAD used for transferring the original seed, the $C_{fix}$ is obtained from Table 1. The seed size in our implementation is 16 Bytes, hence the $C_{ENC}$ is simply $C_{bytes} \times 16$, and the $C_{PRNG}$ is:

$$C_{PRNG} = \frac{Bits_{needed}}{PRNG_{perf}} = \frac{3n \times (log_2 n - 1)}{PRNG_{perf}} \tag{3}$$

Finally, after initialization, and by using a CSN of size $n$ when the bus width of 3D-SOUL is BW, the number of cycles to encrypt and transfer one byte of information is:

$$C_{byte}^{LLC} = \frac{8}{n} \times (\frac{n}{BW} + 1) \tag{4}$$

Using a 64-bit CSN and BW of 8 bits, the $C^{LLC} = 9/8$. Compared to $C^{DCC}$ for the 3D-SOUL1 ($C^{DCC}$=72), and for the 3D-SOUL2 ($C^{DCC}$=17), the LCC mode is at least an order of magnitude faster. Figure 7 compares the superior performance of LCC mode compare with DCC mode in both 3D-SOUL variants.

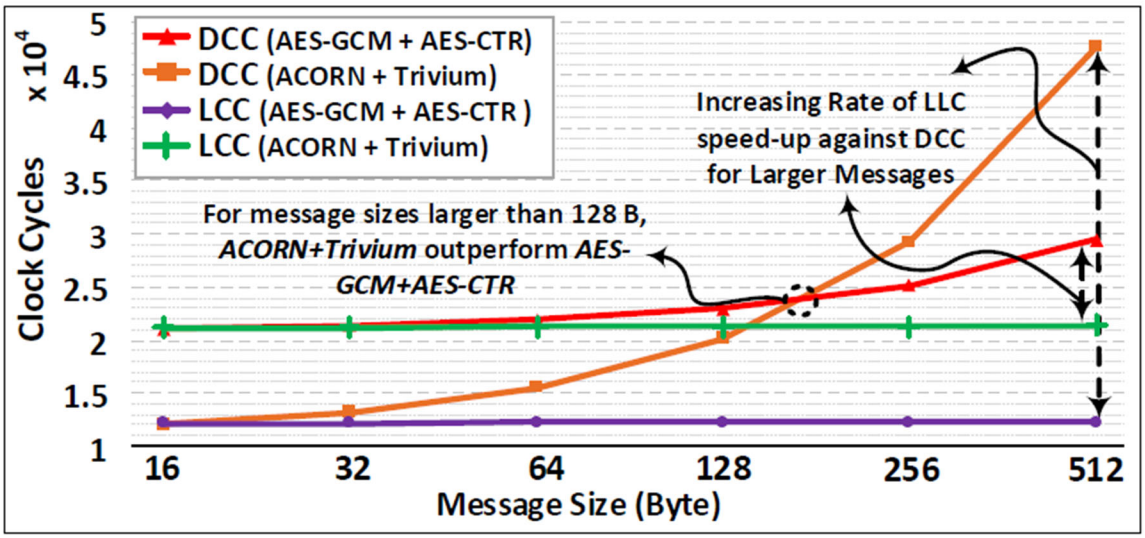



**Figure 7: Total Execution Time in Number of Clock Cycles for (AES-GCM + AES-CTR) and (ACORN + Trivium)**

**Table 5. SAT Execution Time on *OMEGA*-based Blocking CSN and $LOG_{n,log_2(n)-2,1}$ as a Close to Non- blocking**

| CSN Size | 4 | | 8 | | 16 | | 32 | | 64 | | 128 | | 256 | | 512 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mode | blk | non-blk | blk | non-blk | blk | non-blk | blk | non-blk | blk | non-blk | blk | non-blk | blk | non-blk | blk | non-blk |
| SAT Iterations | 6 | 14 | 7 | 18 | 8 | 25 | 12 | 31 | 14 | TO | 24 | TO | 25 | TO | TO | TO |
| SAT Exe. Time (s) | 0.01 | 0.01 | 0.03 | 0.15 | 0.2 | 2.35 | 0.8 | 79.18 | 5.9 | TO | 130.5 | TO | 1136.2 | TO | TO | TO |

*TO: Timeout = $2 \times 10^6$ seconds; The SAT attack is carried on a Dell PowerEdge R620 equipped with Intel Xeon E5-2670 2.6 GHz and 64GB of RAM.*

### 6.2.2 Frequency of TRN Updates in LCC Mode

The frequency of TRN update (U) for LCC is an important design feature. A large $U$ reduces energy as PRNG/TRNG is kept idle for $U$ $P$ cycles. P is the number of required cycles to refill the PRNG buffer after a TRN read. However, when the TRN is fixated for a long duration of time, the possibility of a successful side-channel, algebraic, or SAT attack on the CSN increases. The minimum number of messages required for an algebraic attack (even if such an attack is possible) is $n$, which is the CSN input size. Our experiments show that a SAT attack could recover the key with an even smaller number of inputs. Knowing the number of encryptions/decryptions needed by such attacks, we can set the $U$ to a safe value smaller than the number of required messages to make it resistant to these attacks. So, the value of $U$ should be between $P$ $U$ $n$.

The SAT attack against CSN is implemented like [41]. In this attack, the CSN gate-level netlist and an activated chip is available to the attacker, while the attacker aims to extract the CSN-RCSN configuration signals. Table 5 captures the results of the SAT attack against blocking and near non-blocking CSNs. As illustrated, the time to break a near non-blocking CSN is significantly larger. In each iteration SAT test one carefully selected input message. Hence, if the $U$ is kept smaller than the number of required SAT iterations, the SAT attack could not be completed.

### 6.3 Energy Saving in LCC Mode

As illustrated in Fig. 8(a), in the LCC mode, the TRN is updated every $U$ cycles. $U$ is determined based on the fastest attack on CSN-RCSN pair, which is the SAT attack. After each TRN update, the PRNG takes $P$ cycles to refill its buffer. Note that $P$ cycles required for PRNG could be stacked at the beginning of $U$ cycles or distributed over $U$ cycles depending on the size of PRNG buffer. As long as the TRN completely changes every $U$ cycle, the possibility of attack is eliminated. Hence in each $U$ cycles, for $P$ cycles the PRNG/TRNG and CSN are active, and for $U$ P cycles, the PRNG is clock gated, and only CSN is active. In both cases, the AEAD is active only for the initial exchange of PRNG seed, allowing us to express the power consumption of the LCC mode as:

$$E_{LLC} = C_{PRNG} \times P_H + \left(U\left(\frac{n}{BW} + 1\right) - C_{PRNG}\right) \times P_L \quad (5)$$

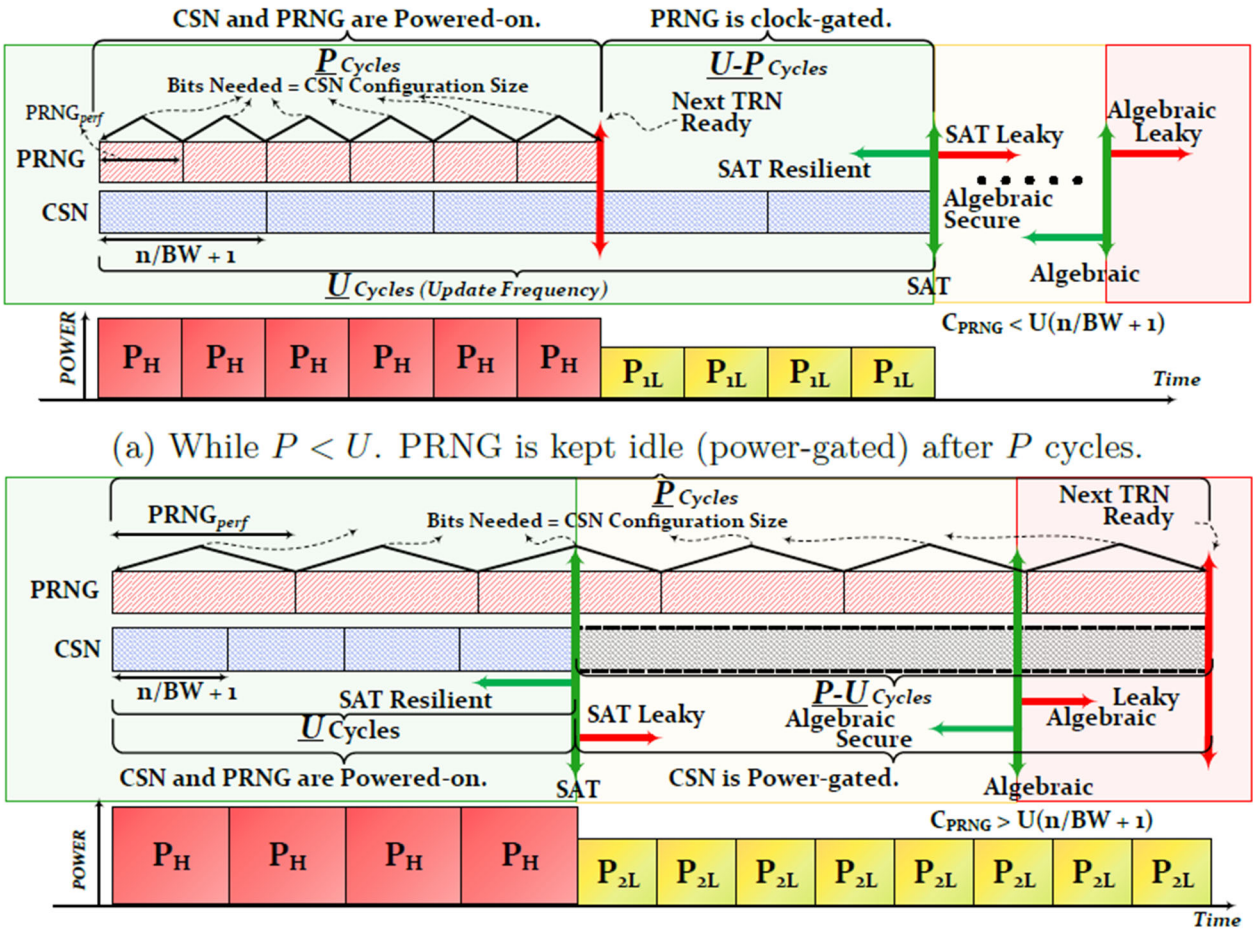



**Figure 8: The Power Consumption at LCC Mode of Operation**

Obviously, the number of required cycles to refill the PRNG buffer after TRN read (P) affects energy consumption and communication throughput. If $P$ < U, as illustrated in Figure 8(a), for $U$ $P$ cycles the PRNG is kept idle (power-gated). However, if $P$ > U, as shown in Figure 8(b), the communication should be stopped for $P$ $U$ cycles till the next TRN is ready and to resist SAT or algebraic attacks. The energy consumption of LCC mode for 3D-SOUL architectures constructed using NIST-compliant and lightweight solution when transmitting different size of messages is captured in Figure 9. As illustrated, the LCC mode, for having to synchronize the two sides using a TRNG seed, is burdened with the initialization cost of AEAD. However, when the CSN-RCSN and PRNG are setup, the energy consumed for exchanging additional messages grow at a much lower rate compare to DCC mode (which is dominated by AEAD and PRNG power consumption as reported in Table 3).

# 7 COMPARING 3D-SOUL WITH PRIOR WORK

To the best of our knowledge, FORTIS [48] is the only comprehensive key-management scheme that was previously proposed. Table 6 compares our proposed solution against FORTIS. 3D-SOUL addresses several shortcomings of the FORTIS:

1) In FORTIS, all chips use identical keys, hence there is no mean of differentiating between chips. In 3D-SOUL each chip has a unique key generated by PUF.
2) In 3D-SOUL, a secret key for communication and authentication is generated by PUF, when FORTIS relies on embedding the private key and public key in GDSII. So, the private key in FORTIS will be known to the fabrication posing the risk that the entire process of activation could be faked in software. In 3D-SOUL, such an attack is prevented as the secret key is generated by PUF and is securely read out using public-key cryptography.
3) In FORTIS, the usage of the private key for chip authentication is vulnerable to SCA. In 3D-SOUL, the secret-key cryptography is side-channel protected, and the public-key encryption is only used once, making 3D-SOUL secure against SCA.
4) In FORTIS, there is also the possibility of deploying a fault attack by fixing the value of session key Ks. In 3D-SOUL, the same attack would require fixing the PUF output or replacing the PUF with a known function. This however could be tested by reading out the output of the PUF using multiple challenges and performing a statistical test on the PUF response (PUF health check).
5) In FORTIS, the activation is done once, hence there is a need to store the obfuscation key in the untrusted chip. In 3D-SOUL, the need to store the obfuscation key in the untrusted chip is removed. In 3D-SOUL, the activation takes place on demand, and the key is removed after power down or reset.
6) 3D-SOUL provides two new mechanisms for communication: a) the DCC mode for added security, and b) the LCC mode for high-speed communication.
7) 3D-SOUL uses a TRNG to produce the seed for PRNG, while FORTIS uses a PRNG without addressing a random source for its seed, increasing its vulnerability.

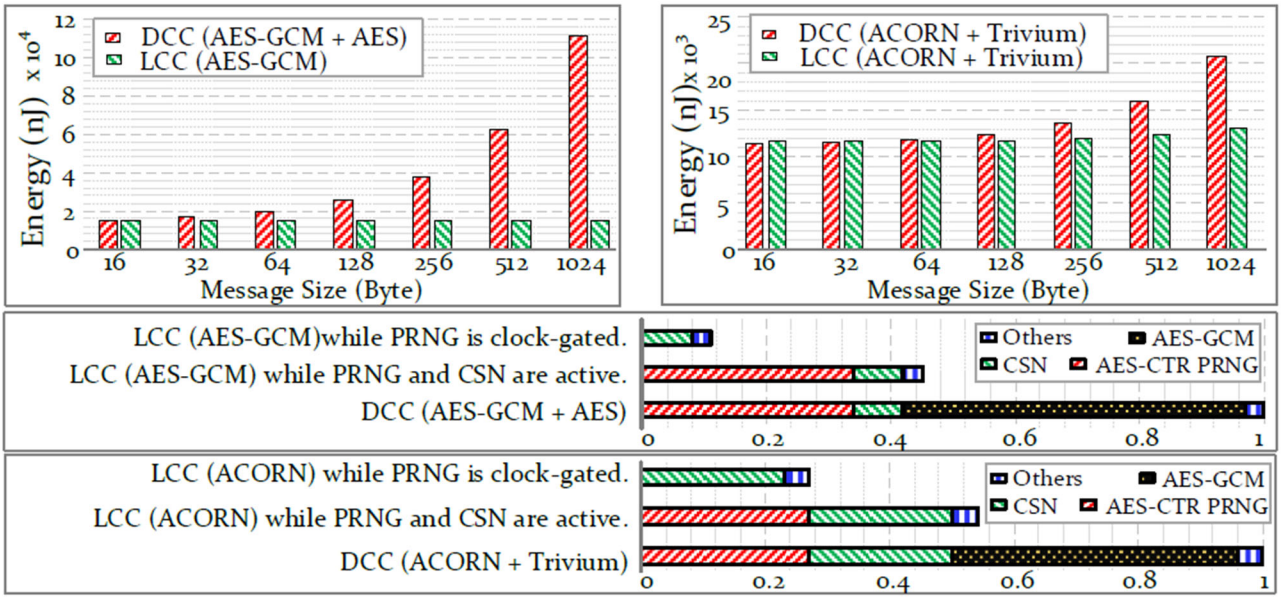



**Figure 9: Energy Breakdown in 3D-SOUL**

**Table 6. 3D-SOUL vs. FORTIS**

| Scheme | Key Management | Data Comm | Private Key | SC Protected | Session Key | Activation | Need to TPM | RNG |
|---|---|---|---|---|---|---|---|---|
| FORTIS | Constant | ✗* | Embedded (known to the fab) | ✗ | Vulnerable to Fault Attack | Once | at Untrusted | PRNG |
| 3D-SOUL | PUF-based Unique | ✓+ | No private key at untrusted | ✓ | Secure | per Demand | at Trusted | TRNG |

**: Not Implemented, but Naturally available using OTP. Limited Performance Due to Lightweight RSA*
*+: Available in Two Variant: DCC (Fully Secure and Limited Performance) and LCC (Leaky yet Secure and High Performance).*

In terms of area overhead, FORTIS [48] provides an estimate for the incurred overhead of their solution, which is around 10K gates. As shown in Table 3, the number of cells for implementing the NIST-compliant (3D-SOUL1) implementation is 25.4K gates, while the lightweight solution (3D-SOUL2) is implemented using 8.7K gates. Table 7 compares the area overhead of FORTIS against 3D-SOUL1 and 3D-SOUL2, when these architectures are deployed to protect a few mid- and large-size benchmarks. Using 3D-SOUL2, which improves the overhead by 14% compared to FORTIS, requires between 0.43% and 21.3% of circuit area in selected benchmarks.

# 8 DESCRIPTION AND ORGANIZATION OF RTL CODE

All source files for the 3D-SOUL project are written in VHDL, and they are in one directory, named 3D-SOUL. The directory 3D-SOULcontains two sub-directories 3DSOULTrustedand 3DSOULUntrusted, which contain the design (in VHDL) for the trusted and untrusted chips. The project is organized and created using Xilinx Vivado 2018.2. For the trusted side, there exists 3DSOUL Trusted.xpr that could be run to execute the HDL project for the trusted side. On the other hand, for the untrusted side, from the sub- directory related to the untrusted side, 3DSOULUntrusted.xprcould be run to load the HDL project for the untrusted side.

For running these projects, the version of the Xilinx Vivado must be 2018.2. For each side of the 3D-SOUL project, i.e. trusted and untrusted, one dedicated Xilinx Vivado is required to be executed.

Also, two Xilinx Nexys-4 DDR boards are required for running the project (XC7A100TCSG324-1). The communication between the trusted chip and untrusted chip is established using a PMOD-based connection. Also, the communication is in the synchronous mode with the same frequency for the system clocks. Figure 10 shows the connection between two chips and the clock generator.

**Table 7. Area Overhead of 3D-SOUL vs. FORTIS**

| Design | Gate Count | FORTIS/Design | 3D-SOUL1/Design | 3D-SOUL2/Design |
|---|---|---|---|---|
| b19 | 40,789 | 24.52% | 62.1% | 21.28% |
| VGA_LCD | 43,346 | 23.07% | 58.45% | 20.02% |
| Leon3MP | 253,050 | 3.95% | 10.01% | 3.43% |
| SPARC | 836,865 | 1.19% | 3.02% | 1.03% |
| Virtex-7 | 2M | 0.5% | 1.26% | 0.43% |

Interconnections between chips are determined and declared in the constraint file of each side's project. For changing the connection, user needs to modify this file to update the connection based on the demand.

Clock generator (demonstrated in Fig. 10) determines the synchronous system clock between both sides (trusted and untrusted), and in this simple example, it is set to 10 MHz.

Also, based on the definitions in the .xdc file, there exists some important push buttons that control the whole operation between two chips.

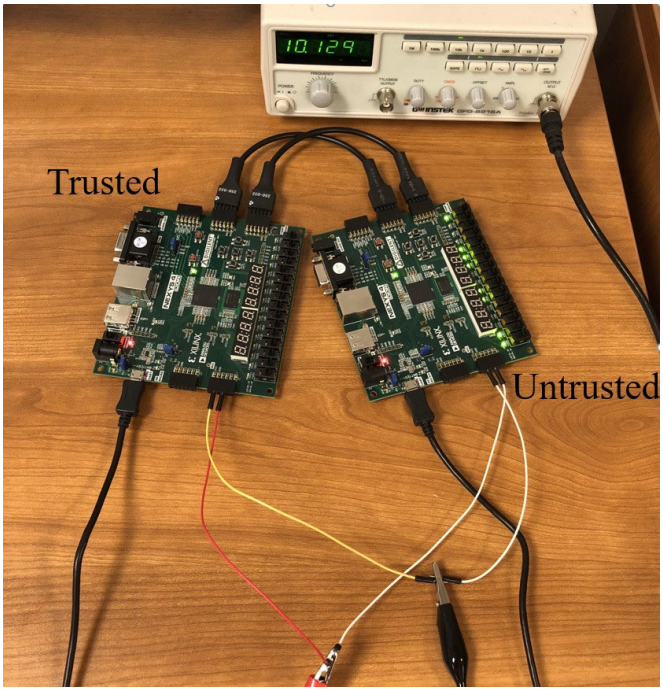
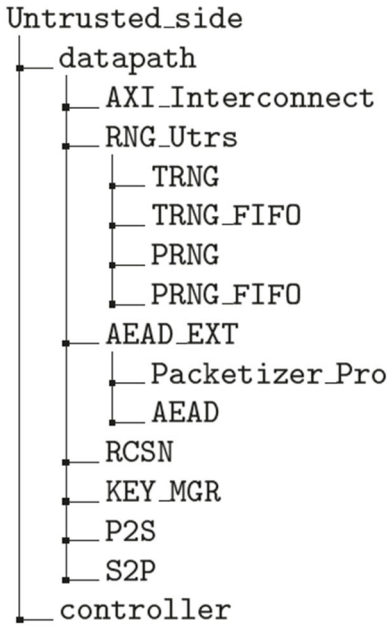



**Figure 10: Connection between Boards in 3D-SOUL Project**

When the project is loaded into Xilinx Vivado 2018.2, the organization and the hierarchy of the HDL files for the Trusted side within the project is as follows:

```
Trusted_side
└── datapath
    ├── AXI_Interconnect
    ├── RNG_trs
    │   ├── TRNG
    │   ├── TRNG_FIFO
    │   ├── PRNG
    │   └── PRNG_FIFO
    └── AEAD_EXT
        ├── Packetizer_Pro
        └── AEAD
            │
            │
            │
            │
            │
            │
            │
            ├── CSN
            ├── S_MEM
            ├── P2S
            └── S2P
    └── controller
```

Similarly, when the project is loaded into Xilinx Vivado 2018.2, the organization and the hierarchy of the HDL files for the Untrusted side within the project is as follows:




### 8.1 UL Top Module: Datapath and Controller

Both the Trustedsideand the untrustedsideconsist of data pathand controller. The data path determines the connectivity between the functional units and the controllercontrols the flow of data in each side. There are many configuration ports that need to be passed between the data path and the controller to synchronize the communication between them. The controller connects the input/output of the macros based on the proper configuration signals. Following are the most important input and output ports of the data path and the controller on both sides.

```
entity data_path_untrusted is
       port(
           --! Global ports
           clk              : in  std_logic;
           rst              : in  std_logic;

           --! Slave data ports
           s_axi_data       : in  std_logic_vector (SERIAL_SIZE-1 downto 0);
           s_axi_valid      : in  std_logic;
           s_axi_ready      : out std_logic;

           --! Master data ports
           m_axi_data       : out  std_logic_vector (SERIAL_SIZE-1 downto 0);
           m_axi_valid      : out  std_logic;
           m_axi_ready      : in   std_logic;

           --! AEAD ports
           start_AEAD             : in  std_logic;
           done_AEAD              : out std_logic;
           error_AEAD             : out std_logic;
           data_size_AEAD         : in  std_logic_vector(15 downto 0);
           instruction_AEAD       : in  std_logic_vector(3 downto 0);

            --! IP ports
            To_IP_data     : out std_logic_vector(w_in-1 downto 0);
```

```
            ready_in      : in std_logic;
            done          : out std_logic;

            --! control ports for RNG
            start_RNG     : in std_logic;
            done_RNG      : out std_logic;
            error_RNG     : out std_logic;
            inst_data_RNG : in std_logic_vector(15 downto 0);
            instruction_RNG: in std_logic_vector(3 downto 0);
    );

end data_path_untrusted;
```

```
entity data_path_trusted is
      port(
          --! Global ports
          clk             : in  std_logic;
          rst             : in  std_logic;
          start           : in  std_logic;

          --! Slave data ports
          s_axi_data      : in  std_logic_vector (SERIAL_SIZE-1 downto 0);
          s_axi_valid     : in  std_logic;
          s_axi_ready     : out std_logic;

          --! Master data ports
          m_axi_data      : out  std_logic_vector (SERIAL_SIZE-1 downto 0);
          m_axi_valid     : out  std_logic;
          m_axi_ready     : in   std_logic;

          --! AEAD ports
          start_AEAD            : in  std_logic;
          done_AEAD             : out std_logic;
          error_AEAD            : out std_logic;
          data_size_AEAD        : in  std_logic_vector(15 downto 0);
          instruction_AEAD      : in  std_logic_vector(3 downto 0);

           --! control ports for RNG
           start_RNG      : in std_logic;
           done_RNG       : out std_logic;
           error_RNG      : out std_logic;
           inst_data_RNG  : in std_logic_vector(15 downto 0);
           instruction_RNG: in std_logic_vector(3 downto 0);

           --! MEM ports
           di_MEM_ready   : out std_logic;
           di_MEM_valid   : in std_logic;
           di_MEM_data    : in std_logic_vector(axi_width-1 downto 0);
    );

end data_path_trusted;
```

The main ports declared in datapathtrustedand datapathuntrustedare (1) system clock, (2) system hard reset, (3) S AXI ports for incoming data, (4) M AXI ports for outgoing data, (5) some direct controlling (configuring) ports for AEAD (controlling the encryption/decryption) module, and (6) some direct controlling ports from the controller for random number generator. The same form of ports is available in the controller of the design. The direct ports (controlling or status) from/to data path trustedand data pathuntrustedare also available in controllertrustedand controlleruntrustedrespectively, such as AEAD and RNG controlling and status signals.

```
entity controller_trusted is
    port(
        --! Global ports
        clk              : in  std_logic;
        rst              : in  std_logic;
        start            : in  std_logic;

        --! Slave data ports
        s_axi_data     : in  std_logic_vector (SERIAL_SIZE-1 downto 0);
        s_axi_valid    : in  std_logic;

        --! Master data ports
        m_axi_ready    : in  std_logic;

        --! AEAD ports
        start_AEAD        : out  std_logic;
        done_AEAD         : in   std_logic;
        error_AEAD        : in   std_logic;
        data_size_AEAD    : out  std_logic_vector(15 downto 0);
        instruction_AEAD      : out  std_logic_vector(3 downto 0);

        --! control ports for RNG
        done_RNG       : in std_logic;
        error_RNG      : in std_logic;
        inst_data_RNG  : out std_logic_vector(15 downto 0);
        instruction_RNG: out std_logic_vector(3 downto 0);
        start_RNG      : out std_logic;
    );

end controller_trusted;
```

```
entity controller_untrusted is
    port(
        --! Global ports
        clk              : in  std_logic;
        rst              : in  std_logic;

        --! Slave data ports
        s_axi_data     : in  std_logic_vector (SERIAL_SIZE-1 downto 0);
        s_axi_valid    : in  std_logic;

        --! Master data ports
        m_axi_ready    : in  std_logic;

        --! AEAD ports
        start_AEAD        : out  std_logic;
        done_AEAD         : in   std_logic;
        error_AEAD        : in   std_logic;
        data_size_AEAD    : out  std_logic_vector(15 downto 0);
        instruction_AEAD      : out  std_logic_vector(3 downto 0);

        --! control ports for RNG
        done_RNG       : in std_logic;
        error_RNG      : in std_logic;
        inst_data_RNG  : out std_logic_vector(15 downto 0);
        instruction_RNG: out std_logic_vector(3 downto 0);
        start_RNG      : out std_logic;
    );

end controller_untrusted;
```

## 8.2 3D-SOUL Random Number Generator: A Mix of True and Pseudo RNG

In this project, we have AXI-basedinterconnection. AXI defines a basic handshake mechanism, composed by an xVALIDand xREADYsignal. The xVALIDsignal is driven by the source (master) to inform the destination entity that the payload on the channel is valid and can be read from that clock cycle onwards. Similarly, the xREADY signal is driven by the receiving entity (slave) to notify that it is prepared to receive data. When both the xVALIDand xREADYsignals are high in the same clock cycle, the data payload is considered *transferred* and the master can either provide a new data payload, by keeping high xVALID, or terminate the transmission, by de-asserting xVALID.

By using AXI-based interconnection, the communication between all macros is standardized. Hence, almost all macros have a finite state machine translating the AXI data incoming/outgoing to the actual data/-control needed per macro. Since the output of some macros are connected to the input of multiple macros, or the input of some macros should choose between multiple output ports of other macros, we have differ- ent AXI Interconnect macros, which handle this kind of communications (multiplexing and demultiplexing AXI-based data). For example, the output of the AEAD in untrusted side should be connected to two input ports of the RCSNand one input port of the P2Smacros. For doing so, we have AXIInterconnect1to3, which provide these connections based on the proper configuration from the controllerthat selects between these three connections. Following is the input and output ports of the AXIInterconnect1to3macros. The input port sel, which comes from the controller, selects one of the three output ports to connect the input to that output.

```
entity AXI_Interconnect_1_to_3 is
  port(
        sel             : in  std_logic_vector (1 downto 0);

        s_axi_data      : in  std_logic_vector (axi_width-1 downto 0);
        s_axi_valid     : in  std_logic;
        s_axi_ready     : out std_logic;

        m_axi_data_1      : out  std_logic_vector (axi_width-1 downto 0);
        m_axi_valid_1     : out  std_logic;
        m_axi_ready_1     : in   std_logic;

        m_axi_data_2      : out  std_logic_vector (axi_width-1 downto 0);
        m_axi_valid_2     : out  std_logic;
        m_axi_ready_2     : in   std_logic;

        m_axi_data_3      : out  std_logic_vector (axi_width-1 downto 0);
        m_axi_valid_3     : out  std_logic;
        m_axi_ready_3     : in   std_logic
   );

end AXI_Interconnect_1_to_3;
```

## 8.3 3D-SOUL Random Number Generator: A Mix of True and Pseudo RNG

As discussed previously, an RNG unit is required on both sides to generate random bits for side-channel protection of AEAD units, a random public message number (NPUB) for AEAD, and TRNs for CSN-RCSN. The macro RNGTrsand RNGUtrsare implemented to support and generate random numbers for these purposes. Like other modules, this unit has some system ports, such as system clock and system reset. It also has some control and status signal from/to controller, such as start, done, error, and

instruction. Apart from these signals, there exist three sets of AXI signals in RNG of the trusted sides and two sets of AXI signals in RNG of the untrusted sides respectively, each provides random numbers for other modules.

The structure of the TRNG is based on the ERO TRNG core described in [39], which can generate only 1-bit of random data per over 20,000 clock cycles. Since in our TI implementations, AES- GCM needs 40 and ACORN 15 bits of random data per cycle, we employed a hybrid RNG unit combining the ERO TRNG with a PRNG. In the hierarchy, we demonstrated that macro RNGconsists of both sub-macro TRNG(and its FIFO TRNG) as well as sub-macro PRNG(and its FIFO PRNG).

Also, the choice of PRNG is configurable in the top module based on two different implementations of PRNG: (1) AES-CTR PRNG, which is based on AES, is compliant with the NIST standard SP 800-90A and generates 12.8 bits per cycle. (2) Trivium based PRNG, which is based on the Trivium stream cipher described in [7].

```
entity RNG_Trs is
    port(
        rst          : in std_logic;
        clk          : in std_logic;
        --! TRN
        trn_tvalid   : out std_logic;
        trn_tready   : in std_logic;
        trn_tdata    : out std_logic_vector(DATA_WIDTH-1 downto 0);
```

```
        --! RMDO
        rmdo_tvalid   : out std_logic;
        rmdo_tready   : in std_logic;
        rmdo_tdata    : out std_logic_vector(RMDW-1 downto 0);
        --! RDO
        rdo_tvalid    : out std_logic;
        rdo_tready    : in std_logic;
        rdo_tdata     : out std_logic_vector(RW-1 downto 0);
        --! Config
        start         : in std_logic;
        done          : out std_logic;
        error         : out std_logic;
        inst_data     : in std_logic_vector(15 downto 0);
        instruction   : in std_logic_vector(3 downto 0)
    );
end RNG_Trs;
```

```
entity RNG_Utrs is
    port(
        rst           : in std_logic;
        clk           : in std_logic;
        --! RMDO
        rmdo_tvalid   : out std_logic;
        rmdo_tready   : in std_logic;
        rmdo_tdata    : out std_logic_vector(RMDW-1 downto 0);
        --! RDO
        rdo_tvalid    : out std_logic;
        rdo_tready    : in std_logic;
        rdo_tdata     : out std_logic_vector(RW-1 downto 0);
        --! Config
        start         : in std_logic;
        done          : out std_logic;
        error         : out std_logic;
        inst_data     : in std_logic_vector(15 downto 0);
        instruction   : in std_logic_vector(3 downto 0)
    );
end RNG_Utrs;
```

### 8.4 3D-SOUL Authenticated Encryption with Associated Data (AEAD)

The general idea of the interface for an authenticated cipher core (AEAD) is composed of five (three + two) major data buses for (1) AXI-based public data inputs (PDI) that is used for sending plaintext/ciphertext into the AEAD core for encryption/decryption; (2) AXI-based secret data inputs (SDI) that is used and dedicated for the secret which is the encryption key; (3) AXI-based data outputs (DO), which is used as the output to read ciphertext/plaintext from the AEAD core after encryption/decryption; (4, 5) AXI-based random data input (RDI and RMDI) that are used as the inputs to get random number from the RNG macro. The AEAD also has some extra ports coming/going from/to the top controller directly, such as `start`, `done`, and `error`. These are controlling and status signals of the AEAD handled and monitored by the top controller.

```
entity AEAD_EXT_pro is
    port(
        --! Global ports
        clk             : in  std_logic;
        rst             : in  std_logic;
        --! Publica data ports
        pdi_data        : in  std_logic_vector(PW-1 downto 0);
        pdi_valid       : in  std_logic;
        pdi_ready       : out std_logic;
        --! Secret data ports
        sdi_data        : in  std_logic_vector(SW-1 downto 0);
        sdi_valid       : in  std_logic;
        sdi_ready       : out std_logic;
        --! Data out ports
        do_data         : out std_logic_vector(PW-1 downto 0);
```

### 8.5 3D-SOUL Configurable Switching and Toggling Network (CSN/RCSN)

The CSN and RCSN are logarithmic switching and toggling networks used for doubling the security of the communication channel between the trusted and untrusted sides. They can permute the order and possibly inverting the logic levels of their primary inputs while these signals are being permuted to different primary outputs. The RCSN is the exact inverse of the CSN. Hence, passing a signal through CSN-RCSN (or RCSN-CSN) will recover the original input. The switching and inversion behavior of CSN-RCSN is configured using a True Random Number (TRN). By using TRN for the CSN-RCSN configuration, any signal passing through the CSN is randomized, and then by passing through the RCSN is recovered. The behavior of CSN/RCSN is described in section 4.1.1. Following are the entities of the CSN and RCSN in VHDL and their important ports. All the inputs and outputs of the CSN and RCSN are AXI-based.

```
entity csn_wrapper is
  port(
    rst            : in  std_logic;
    clk            : in  std_logic;
    al_data_in     : in  std_logic_vector(axi_width - 1 downto 0);
    al_valid_in    : in  std_logic;
    al_ready_out   : out std_logic;
    trng_data_in   : in  std_logic_vector(axi_width - 1 downto 0);
    trng_valid_in  : in  std_logic;
    trng_ready_out : out std_logic;
    dal_data_out   : out std_logic_vector(axi_width - 1 downto 0);
    dal_valid_out  : out std_logic;
    dal_ready_in   : in  std_logic
  );
end entity csn_wrapper;
```

```
entity rcsn_wrapper is
  port(
    rst            : in  std_logic;
    clk            : in  std_logic;
    dal_data_in    : in  std_logic_vector(axi_width - 1 downto 0);
    dal_valid_in   : in  std_logic;
    dal_ready_out  : out std_logic;
    trng_data_in   : in  std_logic_vector(axi_width - 1 downto 0);
    trng_valid_in  : in  std_logic;
    trng_ready_out : out std_logic;
    al_data_out    : out std_logic_vector(axi_width - 1 downto 0);
    al_valid_out   : out std_logic;
    al_ready_in    : in  std_logic
  );
end entity rcsn_wrapper;
```

### 8.6 3D-SOUL Serialized Communication between the Trusted and Untrusted sides (P2S/S2P)

For communicating between the untrusted side and trusted side, we have parallel to serial P2Sand serial to parallel S2P macros. In P2S, as you can see in the following listing, the input ports are parallel and have configurable width, which can be determined and configured in the top module. On the other hand, the output ports is serial. The default serial width is 1 bit. However, this is another configurable parameter that can be set in the top module. The current protocol between the trusted chip and untrusted chips is also AXI-based.

```
entity P2S is
    Port(
        -- parallel input data
        parallel_data : in STD_LOGIC_VECTOR (PARALLEL_SIZE - 1 downto 0);
        parallel_valid : in STD_LOGIC;
        parallel_ready : out STD_LOGIC;
        -- serial output data
        serial_data : out STD_LOGIC_VECTOR (SERIAL_SIZE - 1 downto 0);
        serial_valid : out STD_LOGIC;
        serial_ready : in STD_LOGIC;
        -- Global
        rx_clk : in STD_LOGIC;
        tx_clk : in STD_LOGIC;
        rst : in STD_LOGIC
        );
end P2S;
```

```
entity S2P is
    Port(
        -- serial input data
        serial_data : in STD_LOGIC_VECTOR (SERIAL_SIZE - 1 downto 0);
        serial_valid : in STD_LOGIC;
        serial_ready : out STD_LOGIC;
        -- parallel output data
        parallel_data : out STD_LOGIC_VECTOR (PARALLEL_SIZE - 1 downto 0);
        parallel_valid : out STD_LOGIC;
        parallel_ready : in STD_LOGIC;
        -- Global
        rx_clk : in STD_LOGIC;
        tx_clk : in STD_LOGIC;
        rst : in STD_LOGIC
        );
end S2P;
```

# 9 3D-SOUL SETUP EXAMPLE

In this section, we describe how 3D-SOUL could be used for the protection of an IP. For simplicity, let us assume that the IP that the user is interested in obfuscating is a one-hot counter FSM, shown in Fig. 11. The FSM is first obfuscated using the user-selected obfuscation solution. The 3D-SOUL architecture is independent of the type of obfuscation used, as far as it is a key-based obfuscation solution. Hence, any of the obfuscation solutions previously developed could be utilized. The key values for the obfuscation solutions are stored in key-registers (one register for each key value).

The next step is to create a key-scan chain, allowing the key-value to be loaded into key registers through the scan architecture. In 3D-SOUL, the obfuscation unlock key is stored in the secure memory of the trusted chip. The 3D-SOUL architecture allows the user to have one or multiple key-scan chains. For simplicity, let us assume that the user chooses one key-scan chain. The key-scan chain will have a key-scan input but will not have a key-scan output. With this modification, the obfuscated IP will have one additional key-scan-in input, in addition to its primary input and outputs.

The next step is to load the key from the trusted chip into the untrusted chip through the 3D-SOUL architecture. In 3D-SOUL, it is assumed that the trusted chip, first securely receives the PUF-based SK from the untrusted chip. This is done by employing public-key cryptography, the details of which are described in section 4.1.4. The SK is generated using a PUF in the untrusted chip, thus it is unique for each untrusted chip. There exists a secure read channel from the trusted side to securely read the PUF response. This key will be used as the secret key in the trusted chip. Then, a random number is required to be generated in the chip that is manufactured in the trusted foundry. This random number will serve as the configuration of the CSN/RCSN on both sides (for the trusted and untrusted chip).

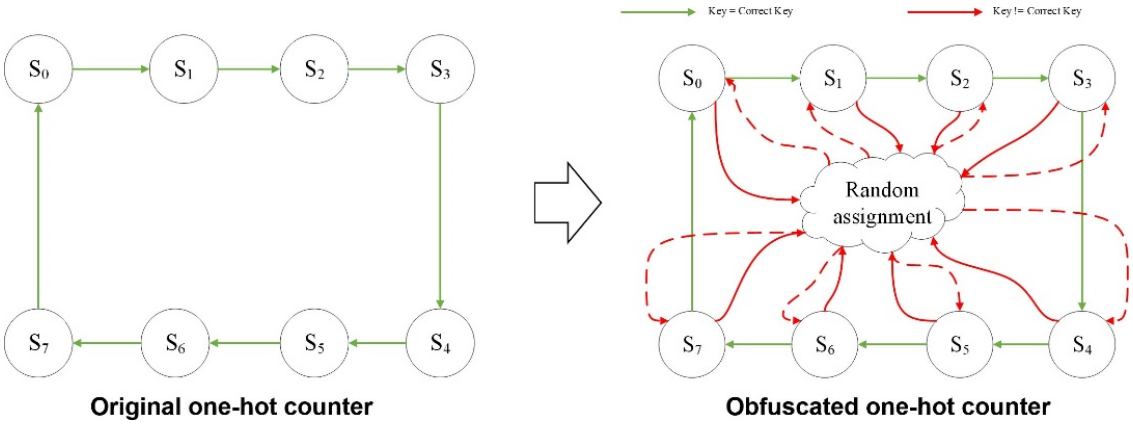



**Figure 11: Original One-hot Counter FSM (left) and (right) Obfuscated One-hot Counter FSM**
*After application of the correct key, the obfuscated one-hot counter will become equivalent to the original circuit, otherwise the state transition will change with the incorrect key.*

Then, the chip that is manufactured in the trusted foundry starts sending the OK. The communication of the obfuscation key is double encrypted and authenticated using the CSN-RCSN and side- channel protected cryptography engine. On the trusted side, the obfuscation key is first permuted/toggled (based on the CSN configuration) and then encrypted. Afterward, the obtained output is sent through S2P into the untrusted side. Each 3D-SOUL-protected device needs to be registered with the trusted side (chip that is manufactured in the trusted foundry) to receive the obfuscation key. The registration is done using the PUF-ID of the untrusted chip. Hence, the PUF is used for both authentication and

generation of the secret key for communication. The SK is generated using a PUF in the untrusted chip, thus it is unique for each untrusted chip, and the untrusted chip does not need a secure memory to store the SK and we only need a secure memory on the trusted side.

On the untrusted side, the dynamic activation license (encrypted permuted/toggled obfuscation key) will be first decrypted through AEAD, and finally, RCSN will recover the actual obfuscation key. Then, the actual obfuscation key will be passed through a wrapper into the targeted IP's scan chain, and by doing so, the actual obfuscation key will be initiated and trapped into the write-only scan chain of the obfuscation IP. The following entity shows a simple example of the one-hot obfuscated counter FSM. In 3D-SOUL ar- chitecture, as we discussed previously, all inter-communications are AXI-based. Hence, the main inputs of the obfuscation counter are provided using AXI. Based on the instruction and the size of the obfuscation key, the obfuscation counter OBFCNTneeds to be initialized with the obfuscation key. This OK is read through the AXI port (data), and then the scan chain of the obfuscated counter (key registers) is initialized using the obfuscation key. Afterward, the counter starts its normal operation. In this example, the 7-segment output ports are connected to the Chip pins to display the value of the one-hot counter FSM on the 7-segment displays of the board under test.

```
1
2  entity OBF_CNT is
3      Port (
4          rst        : in  std_logic;
5          clk        : in  std_logic;
6          -- axi input
7          data_in    : in  std_logic_vector(7 downto 0);
8          data_valid  : in  std_logic;
9          data_ready : out std_logic;
10         -- parallel output
11         data_out  : out std_logic_vector(7 downto 0);
12         data_out_valid  : out std_logic;
13         data_out_ready      : in std_logic;
14         -- 7-segment output
15         seg7_out : out std_logic_vector(6 downto 0);
16         an_en_out : out std_logic_vector(7 downto 0);
17         dp_out  : out std_logic
18 );
19 end OBF_CNT;
```

Replacement of the current example IP with any other IP is straightforward in the 3D-SOUL architecture. Since all communication is standardized based on AXI, there exists a key wrapper macro in the 3D-SOUL architecture helping us to decouple the key initialization from the structure (port-mapping) of the targeted IP. The ports of KEY MGR module that handles this decoupling are as follows:

This module reads the obfuscation key from the output of the RCSN and parallelize them to pass them to the obfuscation module. The paralleled obfuscation key will be passed through `data out` to the

obfuscated IP. Per each segment of the obfuscation key, the obfuscated IP needs to read the key, then it should load it through the scan chain to activate the IP. The size of the key will be sent as a config before the obfuscation key (as a header).

Since all inter-communications in the 3D-SOUL architecture are standardized, by using AXI, to have a new obfuscated IP, the user needs to update two parts. First the generic values determined in the top modules need to be updated accordingly. The following list determines list of configurable generic values in the 3D-SOUL top module:

```
     entity KEY_MGR is
    generic(
        w_out : integer := 64;
        w_in  : integer := 8
    );

    Port (  rst        : in  std_logic;
            clk        : in  std_logic;
            -- axi input
            data_in    : in  std_logic_vector(w_in - 1 downto 0);
            valid_in   : in  std_logic;
            ready_out  : out std_logic;
            -- parallel output
            data_out   : out std_logic_vector(w_in - 1 downto 0);
            ready_in   : in std_logic;
            done       : out std_logic;

            Al_size    : out std_logic_vector((4*w_in) - 1 downto 0)
    );
end KEY_MGR;
```

```
    axi_width     : integer := 8;
    log2_csn_size: integer := 7;

    NPUB_SIZE      : integer := 16;
    COUNTER_SIZE   : integer := 8;
    FIFO_DEPTH     : integer := 4;

    G_W            : integer := PW;
    G_SW           : integer := SW;

    PARALLEL_SIZE    :    INTEGER := 4 * 2;
    SERIAL_SIZE      :    INTEGER := 1 * 2;


    TRNG_O_W        : integer := 16;
    PRNG_I_W        : integer := 128;
    PRNG_O_W        : integer := RW;
    TRNGTOPRNG_FIFODEPTH  : integer := 2;
    RMDI_FIFODEPTH : integer := 2
```

`axi width` determines the width of AXI channel used for inter-communication between different macros in 3D-SOUL architecture.

`log2 csn size` denotes the size of CSN/RCSN in the module (the logarithmic value). At this moment, it is set to 7, meaning that the number of I/Os in CSN/RCSN is $2^7 = 128$ bits. So, this size could be changed based on the demand of the user for a new run.

`NPUB SIZE`, `COUNTER SIZE`, `FIFO DEPTH`, `G W`, and `G SW` determines the configuration values for the AEAD and no need to change them. For the user, the files corresponded to both variants of AEAD (AES-GCM and Trivium), are provided.

`PARALLEL SIZE` and `SERIAL SIZE` determine the sizes of the serial and parallel data of P2S and S2P module. At this moment, the `PARALLEL SIZE` is equal with the AXI width used for the entire system, and the `SERIAL SIZE` is 2 bits.

Parameters `TRNG O W`, `PRNG I W`, `PRNG O W`, and `TRNGTOPRNG FIFODEPTH`, and `RMDI FIFODEPTH` are all con- figurations needed for RNG, and no change is required for then when a new IP is targeted to be placed in 3D-SOUL architecture. For the user, the files corresponded to both variants of RNG (AES-CTR and Triv- ium), are provided.

# 10 PUBLICATIONS

Following is the list of publications that resulted from the full or partial support of this project. The RAID2019 paper is the main paper describing the architecture of 3D-SOUL, while other publications are the obfuscation or scan locking solutions developed in this project.

- **(ICCAD 2020)** Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, Avesta Sasan, "InterLock: An Intercorrelated Logic and Routing Locking", International Conference on Computer Aided Design (ICCAD), 2020 **Nominated for Best Paper Award**

# 11 ABSTRACT:

In this paper, we propose a canonical prune-and-SAT (CP&SAT) attack for breaking state-of-the-art routing-based obfuscation techniques. In the CP&SAT attack, we first encode the key-programmable routing blocks (keyRBs) based on an efficient SAT encoding mechanism suited for detailed routing constraints, and then efficiently re-encode and reduce the CNF corresponded to the keyRB using a bounded variable addition (BVA) algorithm. In the CP&SAT attack, this is done before subjecting the circuit to the SAT attack. We illustrate that this encoding and BVA-based pre-processing significantly reduces the size of the CNF corresponded to the routing-based obfuscated circuit, in the result of which we observe 100% success rate for breaking prior art routing-based obfuscation techniques. Further, we propose a new intercorrelated logic and routing locking technique, or in short InterLock, as a countermeasure to mitigate the CP&SAT attack. In Interlock, in addition to hiding the connectivity, a part of the logic (gates) in the selected timing paths are also implemented in the keyRB(s). We illustrate that when the logic gates are twisted with keyRBs, the BVA could not provide any advantage as a pre-processing step. Our experimental results show that, by using InterLock, with only three 8 8 or only two 16 16 keyRBs (twisted with actual logic gates), the resilience against existing attacks as well as our new proposed CP&SAT attack would be guaranteed while, on average, the delay/area overhead is less than 10% for even medium-size benchmark circuits.

- **(GLSVLSI 2020)** Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, Avesta Sasan, "On Designing Secure and Robust Scan Chain for Protecting Obfuscated Logic", ACM Great Lakes Symposium on VLSI 2020 (GLSVLSI)
  **Abstract:**
  In this paper, we assess the security and testability of the state-of-the-art design-for-security (DFS) architectures in the presence of scan-chain locking/obfuscation, a group of solution that has previously proposed to restrict unauthorized access to the scan chain. We discuss the key leakage vulnerability in the recently published prior-art DFS architectures. This leakage relies on the potential glitches in the DFS architecture that could lead the adversary to make a leakage condition in the circuit. Also, we demonstrate that the state-of-the-art DFS architectures impose some substantial architectural drawbacks that moderately affect both test flow and design constraints. We propose a new DFS architecture for building a secure scan chain architecture while addressing the potential of key leakage. The proposed architecture allows the designer to perform the structural test with no limitation, enabling an untrusted foundry to utilize the scan chain for manufacturing fault testing without having a need to access the scan chain. Our proposed solution poses negligible limitation/overhead on the test flow, as well as the design criteria.

- **(ISVLSI 2020)** Hadi Mardani Kamali, Kimia Zamiri Azar, Avesta Sasan, "SCRAMBLE: The State, Connectivity and Routing Augmentation Model for Building Logic Encryption", Intenational Sympo- sioum on VLSI (ISVLSI), 2020 - **Nominated for Best Paper Award**
  **Abstract:**
  In this paper, we introduce SCRAMBLE, as a novel logic locking solution for sequential circuits while the access to the scan chain is restricted. The SCRAMBLE could be used to lock an FSM by hiding its state transition graph (STG) among a large number of key-controlled false transitions. Also, it could be used to lock sequential circuits (sequential datapath) by hiding the timing paths' connectivity among a large number of key-controlled false connections. Besides, the structure of SCRAMBLE allows us to engage this scheme as a new scan chain locking solution by hiding the correct scan chain sequence among a large number of the key-controlled false sequences. We demonstrate that the proposed scheme resists against both (1) the 2-stage attacks on FSM, and (2) SAT attacks integrated with unrolling as well as bounded-model-checking. We have discussed two

variants of SCRAMBLE: (I) Connectiv- ity SCRAMBLE (SCRAMBLE-C), and (b) Logic SCRAMBLE (SCRAMBLE-L). The SCRAMBLE-C relies on the SAT-hard and key-controlled modules that are built using near non-blocking logarithmic switching networks. The SCRAMBLE-L uses input multiplexing techniques to hide a part of the FSM in a memory. In the result section, we describe the effectiveness of each variant against state-of-the-art attacks.

- **(VTS 2020)** Shervin Roshanisefat, Hadi Mardani Kamali, Kimia Zamiri Azar, Sai Manoj Pudukotai Dinakarrao, Naghmeh Karimi, Houman Homayoun, Avesta Sasan, "DFSSD: Deep Faults and Shallow State Duality, A Provably Strong Obfuscation Solution for Circuits with Restricted Access to Scan Chain", 2020 IEEE 38th VLSI Test Symposium (VTS), 2020
  **Abstract:**
  In this paper, we introduce DFSSD, a novel logic locking solution for sequential and FSM circuits with a restricted (locked) access to the scan chain. DFSSD combines two techniques for obfuscation:
  (1) **D**eep **F**aults, and (2) **S**hallow **S**tate **D**uality. Both techniques are specifically designed to resist against sequential SAT attacks based on bounded model checking. The shallow state duality prevents a sequential SAT attack from taking a shortcut for early termination without running an exhaustive unbounded model checker to assess if the attack could be terminated. The deep fault, on the other hand, provides a designer with a technique for building deep, yet key recoverable faults that could not be discovered by sequential SAT (and bounded model checker based) attacks in a reasonable time.

- **(RAID19)**: Kimia Zamiri Azar, Farnoud Farahmand, Hadi Mardani Kamali, Shervin Roshanisefat, Houman Homayoun, William Diehl, Kris Gaj, Avesta Sasan, "COMA: Communication and Obfusca- tion Manamgement Architecture", Research in Attacks, Intrusions and Defenses (RAID 2019)
  **Abstract:**
  In this paper, we introduce a novel Communication and Obfuscation Management Architecture (COMA) to handle the storage of the obfuscation key and to secure the communication to/from untrusted yet ob- fuscated circuits. COMA addresses three challenges related to the obfuscated circuits: First, it removes the need for the storage of the obfuscation unlock key at the untrusted chip. Second, it implements a mechanism by which the key sent for unlocking an obfuscated circuit changes after each activation (even for the same device), transforming the key into a dynamically changing license. Third, it pro- tects the communication to/from the COMA protected device and additionally introduces two novel mechanisms for the exchange of data to/from COMA protected architectures: (1) a highly secure but slow double encryption, which is used for exchange of key and sensitive data (2) a high-performance and low-energy yet leaky encryption, secured by means of frequent key renewal. We demonstrate that compared to state-of-the-art key management architectures, COMA reduces the area overhead by 14%, while allowing additional features including unique chip authentication, enabling activation as a service for IoT devices), reducing the side channel threats on key management architecture, and providing two new means of secure communication to/from an untrusted chip.

- **(DAC19)**: Hadi Mardani Kamali, Kimia Zamiri Azar, Homayoun Homayoun, Avesta Sasan, "Full- Lock: Hard Distributions of SAT instances for Obfuscating Circuits using Fully Configurable Logic and Routing Blocks", Design Automation Conference (DAC 2019)
  **Abstract:**
  In this paper, we propose a novel and SAT-resistant logic-locking technique, denoted as Full-Lock, to obfuscate and protect the hardware against threats including IP-piracy and reverse-engineering. The

Full-Lock is constructed using a set of small-size fully Programmable Logic and Routing block (PLR) networks. The PLRs are SAT-hard instances with reasonable power, performance and area overheads which are used to obfuscate (1) the routing of a group of selected wires and (2) the logic of the gates leading and proceeding the selected wires. The Full-Lock resists removal attacks and breaks a SAT attack by significantly increasing the complexity of each SAT iteration.

- **(ISVLSI 2018)**: Hadi Mardani Kamali, Kimia Zamiri Azar, Kris Gaj, Houman Homayoun, **Avesta Sasan**, "LUT-Lock: A Novel LUT-Based Logic Obfuscation for FPGA-Bitstream and ASIC-Hardware Protection", IEEE Computer Society Annual Symposium on VLSI (ISVLSI) 2018: 405-410

  **Abstract:**
  In this work, we propose LUT-Lock, a novel Look-Up-Table-based netlist obfuscation algorithm, for protecting the intellectual property that is mapped to an FPGA bitstream or an ASIC netlist. We, first, illustrate the effectiveness of several key features that make the LUT-based obfuscation more resilient against SAT attacks and then we embed the proposed key features into our proposed LUT-Lock algorithm. We illustrate that LUT-Lock maximizes the resiliency of the LUT-based obfuscation against SAT attacks by forcing a near exponential increase in the execution time of a SAT solver with respect to the number of obfuscated gates. Hence, by adopting LUT-Lock algorithm, SAT attack execution time could be made unreasonably long by increasing the number of utilized LUTs.

- **(FPT 2018)**: William Diehl, Farnoud Farahmand, Abubakr Abdulgadir, Jens-Peter Kaps, Kris Gaj, "Face-off between the CAESAR Lightweight Finalists: ACORN vs. Ascon", 2018 International Conference on Field Programmable Technology, FPT 2018, Naha, Okinawa, Japan, Dec. 10-14, 2018

  **Abstract:**
  Authenticated ciphers potentially provide resource savings and security improvements over the joint use of secret-key ciphers and message authentication codes. The CAESAR competition aims to choose the most suitable authenticated ciphers for several categories of applications, including a lightweight use case, for which the primary criteria are performance in resource-constrained devices, and ease of protection against side channel attacks (SCA). Recently, two of the candidates from this category, ACORN and Ascon, were selected as CAESAR contest finalists. In this research, we compare two SCA- resistant FPGA implementations of ACORN and Ascon, where one set of implementations has area consumption nearly equivalent to the defacto standard AES-GCM, and the other set has throughput (TP) close to that of AES-GCM. The results show that protected implementations of ACORN and Ascon, with area consumption less than but close to AES-GCM, have 23.3 and 2.5 times, respectively, the TP of AES-GCM. Likewise, implementations of ACORN and Ascon with TP greater than but close to AES-GCM, consume 18% and 74% of the area, respectively, of AES-GCM.

# 12 STUDENT INFORMATION

This project provided full or partial funding support to the following Ph.D. students:

- **Kimia Zamiri Azar**, Female, US Person Research Area: Hardware Security
  Ph.D. Status: Near completion, with expected defense date on Aug 2021
  Future Plans: Kimia is joining University of Florida as a postdoctoral fellow to continue her research on Hardware Security. She is planning to join Academia after the conclusion of her postdoctoral fellowship in the domain of hardware security.

  **Hadi Mardani Kamali**, Male, US Person Research Area: Hardware Security
  Ph.D. Status: Near completion, with expected defense date on Aug 2021
  Future Plans: Hadi is joining University of Florida as a postdoctoral fellow to continue his research on Hardware Security. He is planning to join Academia after the conclusion of his postdoctoral fellowship in the domain of hardware security.
- **Shervin Roshanisefat**, Male, International Student
  Research Area: Hardware Security
  Ph.D. Status: Near completion, with expected defense date on May 2021 Future Plans: Shervin is currently seeking employment in Industry.
- **Farnoud Farahmand**, Male, International Student Research Area: Cryptographic Engineering
  Ph.D. Status: Completed his degree in August 2020
  Current Position: Farnoud is currently an SOC Power Engineer at Apple (Since Jun 2020).
- **William Diehl**, Male, US Person Research Area: Cryptographic Engineering
  Ph.D. Status: Completed his degree in May 2018
  Current Position: William is currently a Researcher at the Army Research Laboratory (Since Jun 2020).

# 13 REFERENCES


[1] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe. Watermarking Techniques for Intellectual Property Protection. In *Proceedings of the Annual Design Automation Conference (DAC)*, pages 776–781, 1998.

[2] A. Baumgarten, A. Tyagi, and J. Zambreno. Preventing IC Piracy using Reconfigurable Logic Barriers. *IEEE Design & Test of Computers*, 27(1):66–75, 2010.

[3] A. Vakil, H. Homayoun, and A. Sasan. IR-ATA: IR annotated timing analysis, a flow for closing the loop between PDN design, IR analysis & timing closure. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 152–159, 2019.

[4] A. Yeh. Trends in the Global IC Design Service Market. *DIGITIMES research*, 2012.

[5] B. Gunna, L. Bhamidipati, H. Homayoun, and A. Sasan. Spatial and Temporal Scheduling of Clock Arrival Times for IR Hot-Spot Mitigation, Eeformulation of Peak Current Reduction. In *IEEE/ACM Int'l Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2017.

[6] B. J. Gilbert Goodwill, J. Jaffe, and P. Rohatgi. A Testing Methodology for Side-Channel Resistance Validation. In *NIST Non-Invasive Attack Testing Workshop*, volume 7, pages 115–136, 2011.

[7] C. De Canniere and P. Bart. Trivium Specifications. In *eSTREAM, ECRYPT Stream Cipher Project*, 2005.

[8] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan Detection using IC Finger- printing. In *IEEE Symposium on Security and Privacy (SP)*, pages 296–310, 2007.

[9] D.-J. Shyy and C.-T. Lea. Log/sub 2/(N, m, p) Strictly Nonblocking Networks. *IEEE Transactions on Communications*, 39(10):1502–1510, 1991.

[10] D. S. Green. Leveraging the Commercial Sector and Providing Differentiation through Functional Dis- aggregation, 2013. `https://www.darpa.mil/attachments/DisaggregatetheCircuit_Slides.pdf`.

[11] D. Sirone and P. Subramanyan. Functional Analysis Attacks on Logic Locking. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 936–939, 2019.

[12] E. Barker and J. Kelsey. Recommendation for the Entropy Sources used for Random Bit Generation. *Draft NIST Special Publication*, pages 800–900, 2012.

[13] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, P. Yalla, J.-P. Kaps, and K. Gaj. CAESAR Hardware API. *Cryptology ePrint Archive, Report 2016/626*, page 669, 2016.

[14] F. Farahmand, A. Ferozpuri, W. Diehl, and K. Gaj. Minerva: Automated Hardware Optimization Tool. In *Int'l Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8, 2017.

[15] F. Koushanfar. Provably Secure Active IC Metering Techniques for Piracy Avoidance and Digital Rights Management. *IEEE Transactions on Information Forensics and Security*, 7(1):51–63, 2012.

[16] G. Contreras, Md. T. Rahman, and M. Tehranipoor. Secure Split-Test for Preventing IC Piracy by Untrusted Foundry and Assembly. In *IEEE Int'l Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 196–203, 2013.

[17] G. Kolhe, H. M. Kamali, M. Naicker, T. D. Sheaves, H. Mahmoodi, S. M. PD, H. Homayoun, S. Rafatirad, and A. Sasan. Security and Complexity Analysis of LUT-based Obfuscation: From Blueprint to Reality. In *IEEE/ACM Int'l Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019.

[18] G. Kolhe, S. M. PD, S. Rafatirad, H. Mahmoodi, A. Sasan, and H. Homayoun. On custom lut-based obfuscation. In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, pages 477–482, 2019.

[19] H. Ahmadi and W. E. Denzel. A Survey of Modern High-Performance Switching Techniques. *IEEE Journal on Selected Areas in Communications*, 7(7):1091–1103, 1989.

[20] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan. Full-Lock: Hard Distributions of SAT Instances for Obfuscating Circuits Using Fully Configurable Logic and Routing Blocks. In *Proceedings of the Annual Design Automation Conference (DAC)*, pages 89:1–89:6, 2019.

[21] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, and A. Sasan. LUT-Lock: A Novel LUT-Based Logic Obfuscation for FPGA-Bitstream and ASIC-Hardware Protection. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 405–410, 2018.

[22] H. Wu. ACORN: A Lightweight Authenticated Cipher (v3). *Candidate for the CAESAR Competition*, 2016. `https://competitions.cr.yp.to/round3/acornv3.pdf`.

[23] J. B. Wendt and M. Potkonjak. Hardware Obfuscation using PUF-based Logic. In *IEEE/ACM Int'l Conference on Computer-Aided Design (ICCAD)*, pages 270–271, 2014.

[24] J. Delvaux and I. Verbauwhede. Attacking PUF-based Pattern Matching Key Generators via Helper Data Manipulation. In *Cryptographers' Track at the RSA Conference*, pages 106–131, 2014.

[25] J. Guajardo, S. S. Kumar, G.-J. Schrijen , and P. Tuyls. Physical Unclonable Functions and Public-Key Crypto for FPGA IP Protection. In *Int'l Conference on Field Programmable Logic and Applications (FPL)*, pages 189–195, 2007.

[26] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri. Security Analysis of Integrated Circuit Camouflag- ing. In *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security (CCS)*, pages 709–720, 2013.

[27] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Security Analysis of Logic Obfuscation. In *Proceed- ings of the Annual Design Automation Conference (DAC)*, pages 83–89, 2012.

[28] J. Roy, F. Koushanfar, and I. L. Markov. Ending Piracy of Integrated Circuits. *Computer*, 43(10):30–38, 2010.

[29] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan. SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES)*, pages 97–122, 2019.

[30] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan. Threats on Logic Locking: A Decade Later. In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, pages 471–476, 2019.

[31] L. Bhamidipati, B. Gunna, H. Homayoun, and A. Sasan. A Power Delivery Network and Cell Placement Aware IR-Drop Mitigation Technique: Harvesting Unused Timing Slacks to Schedule Useful Skews. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 272–277, 2017.

[32] M. J. Dworkin. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Technical report, 2007. NIST Technical Report.

[33] M. M. Tehranipoor, U. Guin, and S. Bhunia. Invasion of the Hardware Snatchers. *IEEE Spectrum*, 54(5):36–41, 2017.

[34] M. Rostami, F. Koushanfar, and R. Karri. A Primer on Hardware Security: Models, Methods, and Metrics. *Proceedings of the IEEE*, 102(8):1283–1295, 2014.

[35] 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 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1601–1618, 2017.

[36] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu. SARLock: SAT Attack Resistant Logic Locking. In *Int'l Symposium on Hardware Oriented Security and Trust (HOST)*, pages 236–241, 2016.

[37] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu. TTLock: Tenacious and traceless Logic Locking. In *IEEE Int'l Symposium on Hardware Oriented Security and Trust (HOST)*, pages 166–166, 2017.

[38] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran. Security Analysis of Anti-SAT. In *Asia and South Pacific Design Automation Conf. (ASP-DAC)*, pages 342–347, 2017.

[39] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet. A Survey of AIS-20/31 Compli- ant TRNG Cores Suitable for FPGA Devices. In *Int'l Conference on Field Programmable Logic and Applications (FPL)*, pages 1–10, 2016.

[40] P. Chakraborty, J. Cruz, and S. Bhunia. SAIL: Machine learning guided structural analysis attack on hardware obfuscation. In *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 56–61, 2018.

[41] P. Subramanyan, S. Ray, and S. Malik. Evaluating the Security of Logic Encryption Algorithms. In *IEEE Int'l Symposium on Hardware Oriented Security and Trust (HOST)*, pages 137–143, 2015.

[42] P. Tuyls, G.-J. Schrijen, B. Škorić, J. Van Geloven, N. Verhaegh, and R. Wolters. Read-Proof Hard- ware from Protective Coatings. In *Int'l Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 369–383, 2006.

[43] R. P. Cocchi, L. W. Chow, J. P. Baukus, and B. J. Wang. Method and Apparatus for Camouflaging a Standard Cell based Integrated Circuit with Micro Circuits and Post Processing, 2013. US Patent.

[44] R. S. Chakraborty and S. Bhunia. HARPOON: An Obfuscation-based SoC Design Methodology for Hardware Protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1493–1502, 2009.

[45] S. Nikova, C. Rechberger, and V. Rijmen. Threshold Implementations against Side-Channel Attacks and Glitches. In *Int'l Conference on Information and Communications Security*, pages 529–545, 2006.

[46] S. Roshanisefat, H. K. Thirumala, K. Gaj, H. Homayoun, and A. Sasan. Benchmarking the Capabilities and Limitations of SAT Solvers in Defeating Obfuscation Schemes. In *IEEE Int'l Symposium on On-Line Testing And Robust System Design (IOLTS)*, pages 275–280, 2018.

[47] T. Machida, D. Yamamoto, M. Iwamoto, and K. Sakiyama. Implementation of Double Arbiter PUF and its Performance Evaluation on FPGA. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 6–7, 2015.

[48] U. Guin, Q. Shi, D. Forte, and M. M. Tehranipoor. FORTIS: A Comprehensive Solution for Establishing Forward Trust for Protecting IPs and ICs. *ACM Transactions on Design Automation of Electronic Systems*, 21(4):63, 2016.

[49] W. Diehl, A. Abdulgadir, F. Farahmand, J.-P. Kaps, and K. Gaj. Comparison of Cost of Protection against Differential Power Analysis of Selected Authenticated Ciphers. *Cryptography*, 2(3):26, 2018.

[50] W. Diehl, F. Farahmand, A. Abdulgadir, J.-P. Kaps, and K. Gaj. Face-Off between the CAESAR Lightweight Finalists: ACORN vs. Ascon. In *Int'l Conference on Field Programmable Technology (ICFPT)*, 2018.

[51] Y. Alkabani and F. Koushanfar. Active Hardware Metering for Intellectual Property Protection and Security. In *USENIX Security Symposium*, pages 291–306, 2007.

[52] Y. Xie and A. Srivastava. Mitigating Sat Attack on Logic Locking. In *Int'l Conference on Cryptographic Hardware and Embedded Systems (CHES)*, pages 127–146, 2016.

# LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS

| | |
|---|---|
| 3D-SOUL | 3D Split of Obfuscation, Authentication and Licensing |
| AD | Associated Data |
| AEAD | Authenticated Encryption with Associated Data |
| AXI | Advanced Extensible Interface |
| BVA | Bounded Variable Addition |
| BW | Bandwidth |
| CMOS | Complementary Metal-Oxide Semiconductor |
| COMA | Communication and Obfuscation Management Architecture |
| CSN | Configurable Switching Network |
| CUK | Chip Unlock Key |
| DAL | Dynamic Activation License |
| DCC | Double-Cipher Communication |
| DFS | Design-For-Security |
| DO | Data Outputs |
| DPA | Differential Power Analysis |
| DPOK | Dynamic Partial Obfuscation Key |
| DTF | Design For Test |
| ECC | Elliptic-Curve Cryptography |
| FPGA | Field Programmable Gate Array |
| FSM | Finite State Machine |
| IC | Integrated Circuits |
| IP | Intellectual Property |
| LCC | Leaky-Cipher Communication |
| MTJ | Magnetic Tunnel Junction |
| NPUB | Public Message Number |
| OK | Obfuscation Key |
| OTP | One Time Pad |
| PDI | Public Data Inputs |
| POK | Partial Obfuscation Key |
| PPA | Performance and Area |
| PRNG | Pseudo-Random Number Generator |
| PUF | Physical Unclonable Function |
| RAID | Research in Attacks, Intrusions and DefenseS |
| RCSN | Reverse Configurable Switching Network |
| RE | Reverse Engineering |
| RRB | Re-Routing Blocks |
| RTL | Register Transfer level |
| SAT | Satisfiability |

| | |
|---|---|
| SCA | Side Channel Analysis |
| SCA | Side Channel Attacks |
| SK | Secret Key |
| SMT | Satisfiability Modulo Theories |
| TI | Threshold Implementation |
| TRN | True Random Number |
| TRN | True Random Number |
| VHDL | VHSIC Hardware Description Language |