Vulnerability Analysis and Mitigation of Hardware Cryptographic Modules for Implementation of Secure Silicon

Lydia Chung
Department of Electrical and Computer Engineering
University of Florida, USA

***Abstract*—Hardware security is a critical aspect of modern cryptographic systems, as vulnerabilities in hardware cryptographic modules can expose sensitive data to various attacks. This research focuses on analyzing the security and design of hardware cryptographic modules by evaluating them against well-known threats and identifying potential weaknesses. Specifically, we investigate three key vulnerability types: malicious implants (hardware Trojans), information leakage, and finite-state machine vulnerabilities. Additionally, we explore and implement mitigation strategies to enhance the security of these modules. These cryptographic modules are integrated into a security engine firmware system to ensure their effective deployment in secure environments. This work contributes to the development of more resilient cryptographic hardware implementations, strengthening the foundation of secure computing systems.**

I. Introduction

As technology becomes a larger part of modern life, vulnerabilities in computing systems can lead to data breaches, insecure financial transactions, or system failures. Hardware security is especially important, since weaknesses at the hardware level can undermine even the most robust software-based protections. Sensitive transactions such as online banking, healthcare records, and government data now travel across networks that are vulnerable to interception and manipulation. However, security depends not only on software, but also on reliable hardware components. It requires secure key storage, cryptographic computations, and trusted execution, all of which are rooted in hardware. If the underlying hardware is compromised, an attacker can extract private data or manipulate communications, effectively rendering software protections useless.

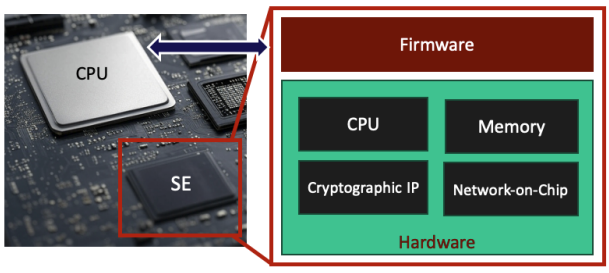

Fig. 1. Security engine firmware structure and components.

A security engine is a specialized subsystem within a chip that typically includes an independent CPU, memory, cryptographic hardware modules, and a network-on-chip (NoC) for internal communication. Inclusion of cryptographic modules, such as AES, RSA, ECDSA, or SHA accelerators, allows the security engine to handle sensitive operations quickly and securely. This firmware is responsible for starting secure operations from the boot time of the device. As the root of trust, the security engine firmware is executed first during system startup (Figure 1). Because it runs in a trusted execution environment, it ensures that only authenticated and verified software is allowed to run on the rest of the system.

The goal of a security system is data integrity, authentication, and privacy (Figure 2). Changes should be made only by authorized users and through authorized mechanisms to preserve integrity. A system must prevent privacy violations from eavesdroppers who spy on communications between users. Authentication spoofing is another important security problem to consider, which is when an attacker impersonates another person's credentials to gain access to a network. Cryptographic IP cores can help prevent these problems by ensuring data is encrypted, hashed, and signed to preserve privacy and authenticity.

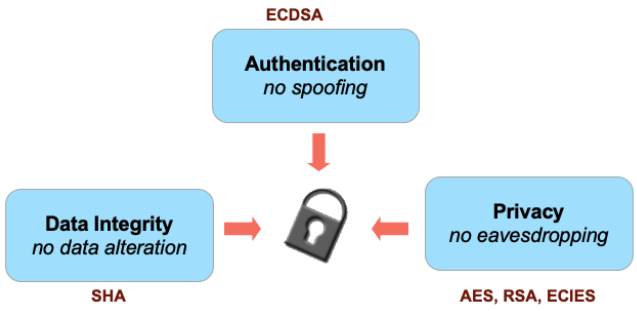

Fig. 2. Cryptographic modules aid in accomplishing the goals of hardware security.

In this honors thesis, the following contributions are made:

1) Analyze common cryptographic modules for malicious implant vulnerabilities, information leakage, and finite state machine vulnerabilities,
2) Implement mitigation strategies on the cryptographic modules based on vulnerabilities found, and
3) Integrate the secured cryptographic modules into a larger security engine firmware system.

The remainder of this thesis is organized as follows. Section II provides an overview of the cryptographic modules and

vulnerabilities analyzed. Section III describes my work in implementing suitable mitigation for the vulnerabilities found in the modules. Section IV details how I created a wrapper for the cryptographic modules to integrate them into a security engine firmware system. Finally, Section V concludes the thesis.

II. Cryptographic IP Cores

This section provides details on the four cryptographic intellectual property (IP) cores analyzed in my thesis, including their algorithms and hardware implementations.

A. Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) is a symmetric block encryption algorithm. It operates on fixed-size blocks of 128 bits and supports key sizes of 128, 192, or 256 bits. In its basic form, AES counter mode (AES-CTR) encrypts an incrementing counter and XORs it with a plaintext block to generate the ciphertext (Figure 3) [1]. However, AES-CTR does not provide authentication, meaning that attackers can manipulate ciphertext during transmission without detection, making it unsuitable for applications requiring data integrity validation.

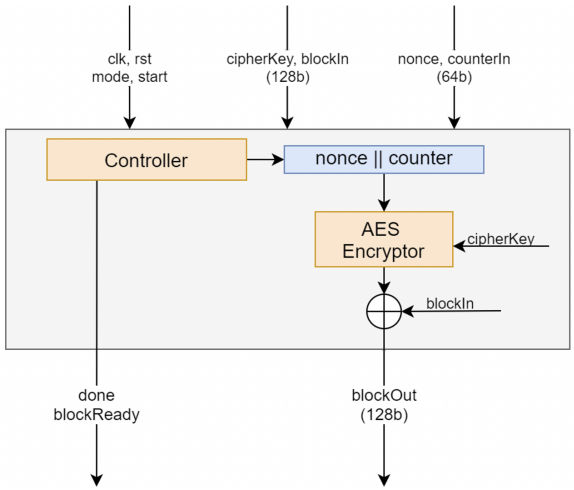

Fig. 3. AES counter mode for keysize of 128 bits.

Galois/Counter mode (AES-GCM) combines AES-CTR with a Galois message authentication code (MAC) to ensure that the ciphertext has not been tampered with. An authentication tag is generated as an additional output by performing a hash function on the ciphertext. The recipient can use both the authentication tag and the ciphertext to verify the integrity of the encrypted data they received [2].

Another mode is the AES XEX Ciphertext Stealing (AES-XTS) operation, commonly used for disk encryption. It is similar to GCM but does not have authentication signals. Instead, data blocks can be modified independently using two different AES keys - one for encryption and one as a tweak key to prevent attackers from detecting patterns in encrypted storage [3].

B. Secure Hashing Algorithm (SHA)

The Secure Hashing Algorithm (SHA) is a cryptographic hash function widely used for digital signatures, password hashing, and blockchain security. SHA takes an input message of any size and produces a fixed-length hash that is deterministic, irreversible, and collision-resistant. There are three versions of the SHA algorithm, notably SHA-1, SHA-2, and SHA-3. SHA-1 has collision vulnerabilities and is deprecated, while SHA-2 is the most widely used today. Both SHA-2 and SHA-3 have multiple variants with different hash sizes that generate outputs of 224, 256, 384, and 512 bits. SHA-256 and SHA-512 are most commonly used for cryptography (Figure 4).

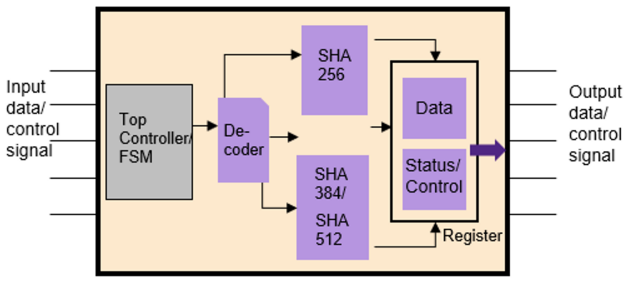

Fig. 4. Hardware implementation of the SHA-2 core.

C. Rivest-Shamir-Adleman (RSA)

A popular encryption scheme is the Rivest-Shamir-Adleman (RSA) asymmetric key algorithm. It uses large prime numbers to generate a modulus for encrypting and decrypting messages. A public key is chosen using the modulus and used for encrypting messages, while a private key is computed from the public key and used for decryption [4]. In our hardware implementation of RSA, exponentiation is done efficiently with the Montgomery Ladder algorithm (Figure 5). The security of RSA relies on the size of the key and the difficulty in factoring out large prime numbers from the modulus. Our RSA core currently supports key sizes of 2048, 3072, and 4096 bits.

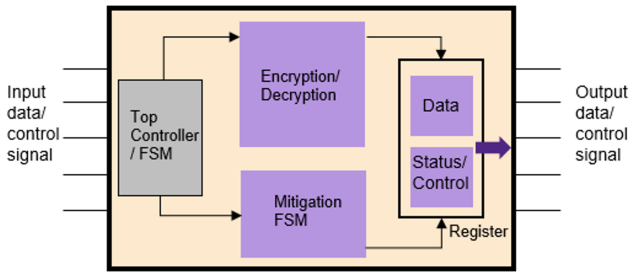

Fig. 5. Hardware implementation of the RSA core.

D. Elliptical Curve Cryptography (ECC)

Elliptical curve cryptography is the mathematical foundation of two cryptographic algorithms, ECDSA and ECIES. Elliptic

Curve Digital Signature Algorithm (ECDSA) is widely used for authentication and integrity verification in digital communications. ECDSA ensures that messages or transactions are signed by an authorized party and have not been tampered with during transmission. The message is first hashed with SHA-256 to modify it into a fixed length. Signature generation and verification use elliptic curve and field operations to perform arithmetic procedures on prime curves (Figures 6 and 7). The hardware implementation of ECDSA uses the private key, message, and elliptical curve parameters to generate a signature pair that can then be verified to ensure message integrity [5].

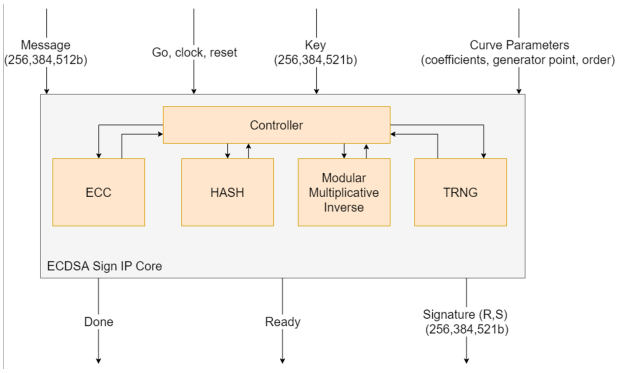

Fig. 6. Hardware implementation of ECDSA signature generation core.

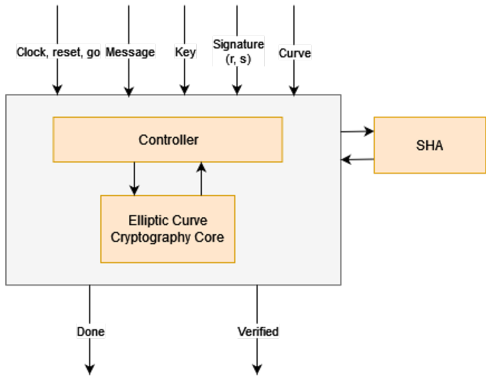

Fig. 7. Hardware implementation of ECDSA signature verification core.

Elliptic Curve Integrated Encryption Scheme (ECIES) is an encryption scheme used for secure data encoding. ECIES encryption and decryption also use elliptic curve and field operations to generate an output (Figure 8). The public key, message, and curve parameters are used for encryption to create a ciphertext, ephemeral public key (shared secret), and authentication tag. Decryption takes these outputs, a private key, and the same curve parameters to retrieve the original message (Figure 9) [5].

III. Vulnerability Analysis and Mitigations

This section assesses the vulnerabilities of the previously described cryptographic modules and suggests mitigation techniques based on previous literature.

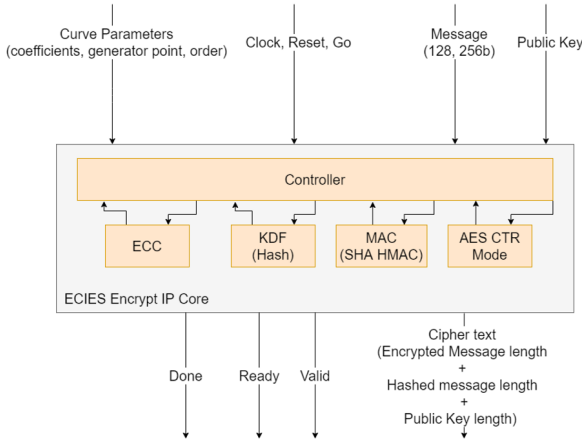

Fig. 8. Hardware implementation of ECIES encryption core.

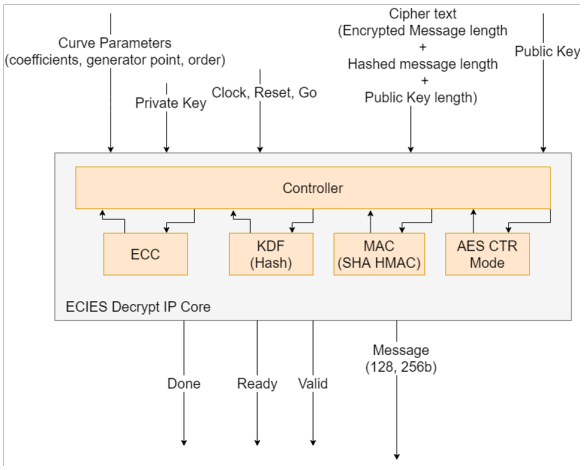

Fig. 9. Hardware implementation of ECIES decryption core.

A. Malicious Implants

An attacker can introduce a hardware Trojan (HT) into a design by exploiting signals that rarely toggle during operation to run a payload. These rare nodes may be deeply embedded in logic or dependent on specific input sequences to reduce detectability (Figure 10). Two methods of automated test pattern generation (ATPG) can be used to efficiently generate test patterns that activate both rare signals (states) and rare branches (transitions). Statistical test generation is based on the N-activation of rare events that ensures that each rare node is activated N times, increasing the likelihood of triggering stealthy hardware Trojans. Maximal clique sampling is a complementary approach that attempts to activate as many rare nodes at the same time to generate more effective test vectors [6].

We applied the ATPG framework to our AES and ECDSA modules to detect rare nodes in the designs. Once rare nodes were detected, proper test vectors were developed to provide appropriate coverage of the rare signals so that verification can find triggers that could be used for hardware Trojans [6].

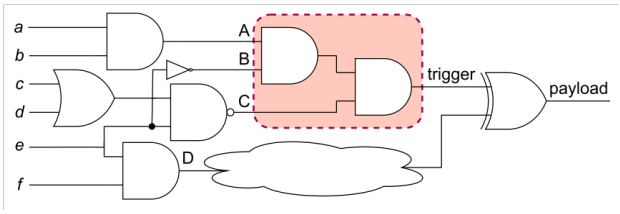

Fig. 10. Rare node used to trigger a payload [6].

B. Information Leakage

On System-on-Chips (SoCs), many functional blocks perform tasks using different pieces of information. For example, a cryptographic module may use a private key, configuration bits, and control signals to perform an encryption operation. These can be separated into primary and secondary assets. Primary assets are valuable or sensitive information that represent the main targets of an attacker, such as cryptographic keys, passwords, or firmware code. If leaked or modified, these directly compromise the integrity of the system. Secondary assets are pieces of data or information that may not be valuable themselves but can be used to infer or access primary assets, such as internal register states, power consumption, or cache access patterns. These are often used by attackers as stepping stones in information leakage or side-channel attacks. Even if primary assets are directly protected, they can still be compromised by secondary assets via side channel attacks that use power, timing, or EM radiation analysis [7].

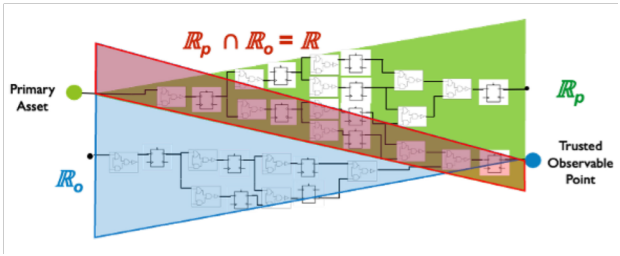

Fig. 11. Overlapping fan out and fan in signals used to search for leaky paths in a design [7].

Asset identification first searches for internal components that are structurally connected to a primary asset using propagation analysis (Figure 11). Of these internal components, some may be classified as secondary assets if it has a correlation with a primary asset. Once vulnerable assets have been identified, they can be obfuscated or otherwise protected to prevent information leakage. This process also traces the information flow in the design to allow leaky paths to be identified and secured [7].

C. Finite State Machine Vulnerability

Finite state machines (FSMs) are used to implement sequences of computation operations. FSMs typically consist of different states with varying privilege levels; for example, a non-protected password checker state may transition into a protected logged-in state. The transition from a non-protected state to a protected state should require an authorized transition mechanism. However, laser-based fault injection (LFI) can allow an attacker to bypass authorization by flipping bits in the FSM's state register, potentially forcing unauthorized transitions (Figure 12). To mitigate this vulnerability, FSM states can be encoded to ensure a high Hamming distance (HD) between each pair of states, making bit-flip attacks less effective [8].

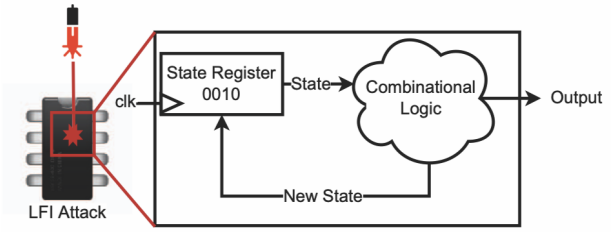

Fig. 12. Laser fault injection attack on an FSM state register [8].

Each of our cryptographic IPs contains an FSM controller. Depending on the number of states in the FSM and the attacker's potential bit flipping capability of N, a new FSM encoding is generated for each IP. Even if an attacker flipped N bits in the FSM state register, they would not be able to transition from a non-protected state to a protected state [8]. After performing mitigation, we synthesized the designs and found an expected small increase in area due to the greater amount of bits required to securely encode FSM states resistant to bit flipping (Figure 13).

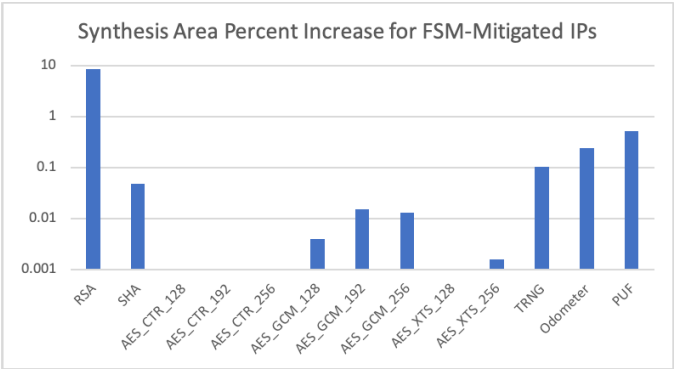

Fig. 13. Synthesis results after FSM mitigation.

IV. Integration into Secure Silicon

This section describes the process of integrating the cryptographic IPs into a security engine firmware. I created SoC wrappers to communicate between the IP cores and firmware according to the AHB bus interface protocol (Figure 14).

Four cryptographic modules were integrated into the security engine: AES-CTR, AES-GCM, SHA, and RSA. I wrote specifications for the design of wrappers based on the communication protocol required by the security engine. The AHB bus can only transfer 32 bits per clock cycle, so the data transfer interface required multiple cycles to bring the

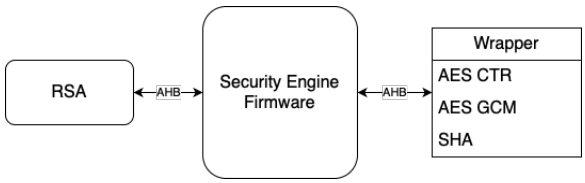

Fig. 14. Security engine overview with cryptographic modules.

full amount of data required into the IPs before computation could begin. In addition, the security engine uses memory-mapped inputs and outputs (I/O), so data must be read from addresses in firmware memory and provided to the IPs in an input stream. Output data must also be written into specified memory addresses. The wrappers I created used control, data, and key interfaces to send control signals, perform data I/O, and securely deliver keys into the IPs respectively.

A. SHA Wrapper

The SHA module had the simplest wrapper and only required one data and one control interface to transfer a single input and output (Figure 15). The control interface would wait for start and stop signals to trigger the SHA IP core. Since SHA is a hashing algorithm, it uses the hash control and hash data interfaces that existed as separate resources from the cipher control, data, and key interfaces used by the cipher cores RSA and AES. Because it does not share the same resources, SHA can run in parallel with the other two IPs.

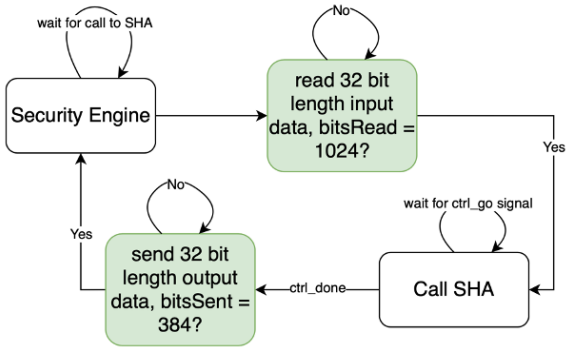

Fig. 15. Data input and output for the SHA wrapper.

B. RSA Wrapper

The RSA module required the most memory transfers since it used the largest input data of 3072 bits. Because of the scale of the inputs required, RSA reads directly from the security engine's RAM instead of the wrapper's local memory. It uses the cipher control, data, and key interfaces, since RSA is an encryption algorithm. In addition, RSA runs separately from the AES cores because it is an asymmetric cipher unlike AES.

C. AES Wrapper

For the AES module, the control interface was directly connected to the start and stop signals of the IP and included parameters like key size and number of data bytes. The data interface transferred the input and output cipher data and authentication data for GCM mode and stored the resulting ciphertext and authentication tag into the security engine's memory (Figure 16). The key interface securely reads the nonce and cipher key for encryption and decryption operations.

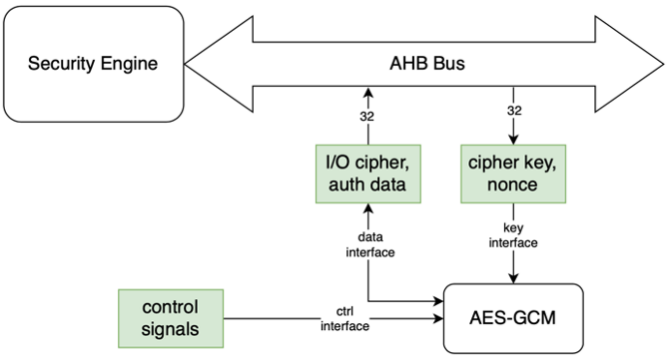

Fig. 16. Control, data, and key interfaces for the AES wrapper.

V. Conclusion

Secure firmware development is vital for preserving privacy, maintaining trust, and securing data. Attackers can exploit hardware vulnerabilities in IPs like hardware Trojans, information leakage, or finite state machine encoding. In this honors thesis, I investigated the vulnerabilities in hardware cryptographic IP implementations and how to mitigate them using state-of-the-art methods. Experimental results show that mitigation came with a comparatively small cost in terms of synthesized design area. Finally, I created communication wrappers to integrate the mitigated IP cores into an existing security engine firmware.

References

[1] J. Daemen and V. Rijmen, *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002.

[2] D. Mcgrew and J. Viega, *The Galois/Counter Mode of Operation (GCM)*, 2005. [Online]. Available: https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf

[3] L. Martin, "Xts: A mode of aes for encrypting hard disks," *IEEE Security & Privacy*, vol. 8, no. 3, pp. 68–69, 2010.

[4] R. L. Rivest, A. Shamir, and L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, 1977. [Online]. Available: https://people.csail.mit.edu/rivest/Rsapaper.pdf

[5] M. Amara and A. Siad, "Elliptic curve cryptography and its applications," in *International Workshop on Systems, Signal Processing and their Applications, WOSSPA*, 2011, pp. 247–250.

[6] A. Jayasena and P. Mishra, "Scalable detection of hardware trojans using atpg-based activation of rare events," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 12, p. 4450–4462, Jun 2023.

[7] A. Ayalasomayajula, N. Farzana Dipu, M. M. Tehranipoor, and F. Farahmandi, "Automatic asset identification for assertion-based soc security verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 10, pp. 3264–3277, 2024.

[8] A. Jayasena, K. Rani, and P. Mishra, "Efficient finite state machine encoding for defending against laser fault injection attacks," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, 2022, pp. 247–254.