

## Article

# Chisel-Based Hardware Trojan Design: A Comparative Case Study with AES-T100

Jianxin Wang \*, Runze Zhou, Zixuan Wang, Lei Zhang, Chaoen Xiao, Zhao Wang, Maosheng He, Qian Cheng and Kaibo Sun

Department of Electronics and Communication Engineering, Beijing Electronic Science and Technology Institute, Beijing 100070, China

\* Correspondence: wangjianxin@besti.edu.cn

## Abstract

Hardware Trojans (HTs) threaten integrated-circuit security in a globalized semiconductor supply chain, yet how the choice of hardware description language—manual Verilog versus compiler-optimized agile languages such as Chisel—affects the synthesis quality of a Trojan-bearing design remains unexplored. This work re-implements AES-T100, the Trust-Hub leakage benchmark that exfiltrates an AES-128 key via an LFSR-driven side channel, faithfully in Chisel. It then compares the resulting netlist against a hand-written Verilog baseline under matched FPGA synthesis and examines dual-use implications. AES-T100 was rebuilt in Chisel 3.5.0 (20-bit LFSR PRNG, 1-to-8-bit key-obfuscation unit,  $8\times$  flip-flop leakage circuit), compiled through FIRRTL 1.5.0, and synthesized on a Cyclone IV E FPGA via Quartus Prime 21.1. The FIRRTL-generated netlist achieved 23.8% higher  $f_{\max}$  (387.60 vs. 313.19 MHz) and throughput (4.96 vs. 4.01 Gbps) at identical logic-element count (2096 LEs), with only 1.50% Trojan overhead. The gap localizes to the host AES cipher, consistent with FIRRTL optimization passes. A five-seed replication confirms statistical robustness ( $t(8) = 11.7$ ,  $p < 0.001$ , 95% CI: 18.6–27.7%). Causal attribution—compilation flow versus single-author Verilog coding—remains open, requiring pass-ablation experiments. The parametric template supports HT benchmark generation. FIRRTL-level static analysis is proposed as a defensive direction.

**Keywords:** Chisel; hardware Trojan; AES-128; FPGA security; FIRRTL; agile hardware design

## 1. Introduction

### 1.1. Motivation and Context

Hardware security has become a central concern in modern integrated-circuit (IC) design. The globalization of the semiconductor supply chain—from RTL design through third-party IP integration, offshore fabrication, and distributed test and distribution—has created an extensive, largely untrusted attack surface [1,2]. Within it, Hardware Trojans are malicious circuit modifications instantiated in silicon at manufacturing time: they persist across software reinstallation, evade malware scanners, and can activate only under rare attacker-chosen conditions that escape functional testing [3,4]. Concerns over suspect modifications in military-grade FPGAs, counterfeit and subverted ICs in critical infrastructure, and hardware backdoors for industrial espionage have driven sustained investment in HT detection, prevention, and mitigation [5], with countermeasures spanning logic testing, side-channel fingerprinting, and machine-learning netlist analysis (surveyed in Section 1.8).



Academic Editor: Paris Kitsos

Received: 9 June 2026

Revised: 5 July 2026

Accepted: 9 July 2026

Published: 16 July 2026

**Copyright:** © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

### 1.2. The Agile HDL Revolution

Concurrent with these security developments, the hardware design community is shifting toward agile hardware construction languages. Verilog and VHDL, standardized in 1995 and 1987 and universally supported by commercial EDA tools, differ significantly in type discipline. Verilog and SystemVerilog employ weak typing where bit-width mismatches and unintended latch inference propagate silently to synthesis [6]. VHDL, by contrast, is a strongly typed language in which bit-width mismatches are caught at compile time. However, VHDL is not immune to implicit hardware generation: incomplete coverage of enumerated types in case statements produces redundant latches that can serve as an unintended structural basis for vulnerability insertion. Both traditional languages are verbose and offer limited abstraction: designers commit to bit widths, pipeline stages, and interfaces early. Chisel (Constructing Hardware in a Scala Embedded Language), developed at UC Berkeley, embeds hardware construction primitives as a domain-specific library within Scala [7], bringing object-oriented reuse, functional generators (designs are typically 3–5× shorter than Verilog), and compile-time type checking. We detail these in Section 1.7. Chisel designs are not synthesized directly but compiled to the Flexible Intermediate Representation for RTL (FIRRTL) [8], which applies cross-module constant propagation, common-subexpression elimination, dead-code removal, width inference, and logic simplification before emitting Verilog. Whether these automatic optimizations yield gate-level implementations competitive with routine manual RTL is the empirical question this paper addresses. At scale, the Chipyard SoC framework [9] and Chisel-based RISC-V generators [10] demonstrate that this flow produces tapeout-ready RTL for configurable multi-core designs whose hand-coded equivalent would be prohibitively labor-intensive.

### 1.3. Research Gap and Problem Statement

Despite active parallel research in hardware security and agile HDL methodology over the past decade, the intersection remains, to our knowledge, largely unexplored in the open literature. Based on a systematic analysis of keyword combinations in the HT design and implementation literature indexed in IEEE Xplore, ACM Digital Library, Scopus, and the Trust-Hub benchmark corpus (Boolean queries combining “hardware Trojan” with “Chisel”, “FIRRTL”, “agile HDL”, “hardware construction language”, and “high-level synthesis Trojan” over the publication window 2012–2025, last searched April 2025, yielding 143 initial results after deduplication), no peer-reviewed HT implementation reporting an agile/FIRRTL-based design flow with a controlled synthesis comparison against a conventional HDL baseline was identified after title and abstract screening. All HT implementations examined used Verilog or VHDL [11,12]. Our contribution is the first controlled measurement on a concrete and widely recognized Trojan benchmark of how an agile Chisel/FIRRTL flow compares to manual Verilog for an HT-bearing design.

Application context. This comparison addresses two practical scenarios: (1) Scalable HT benchmark generation—as agile HDLs enable parametric, composable designs, understanding their synthesis characteristics informs the development of configurable Trojan suites for evaluating detection tools and (2) Defensive toolchain design—if the FIRRTL intermediate representation introduces structural regularities (e.g., canonical node ordering, explicit width annotations) that facilitate automated HT detection via static IR analysis, the agile flow may offer an unexpected defensive benefit that complements existing side-channel and ML-based detectors. Conversely, if agile HDLs confer measurable performance advantages for HT design, the security community must account for this capability asymmetry in threat modeling.

### 1.4. Our Approach and Contributions

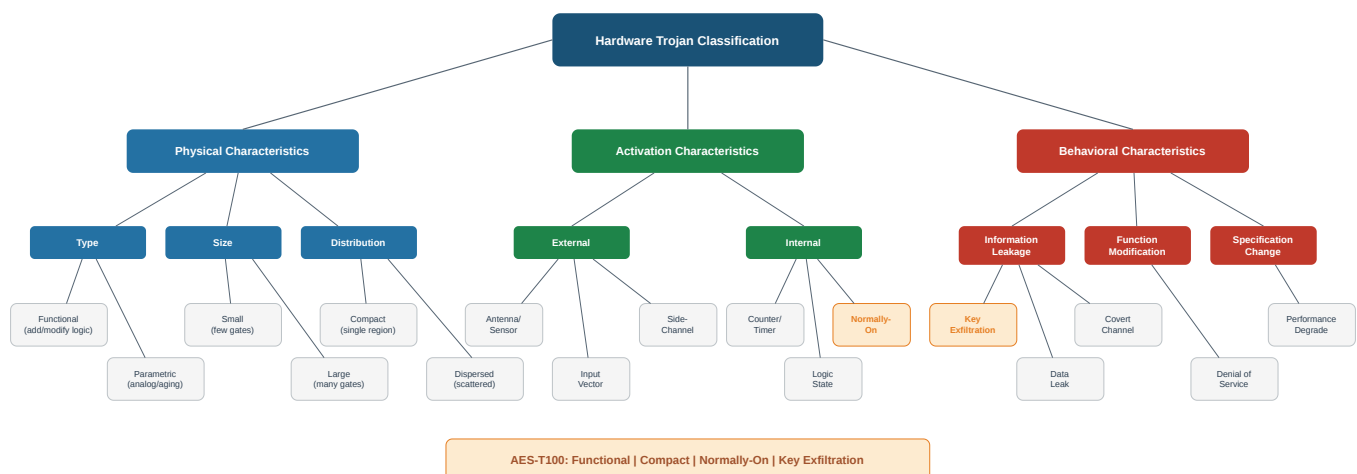
We address this gap through a controlled comparative case study centered on AES-T100, a publicly catalogued benchmark from the Trust-Hub suite [13,14]—a leakage Trojan that exfiltrates the 128-bit AES key through a passive LFSR-driven side channel. Both implementations under test are functionally equivalent and synthesized on an identical FPGA hardware: one through the full Chisel 3.5.0 → FIRRTL 1.5.0 → Verilog → Quartus pipeline, one hand-written in Verilog-2001 with identical Quartus settings. The principal contributions are as follows:

1. Chisel re-implementation of a recognized HT benchmark: a complete, parametric, reproducible Chisel version of AES-T100—module hierarchy, FIRRTL flow, and key-recovery logic.
2. First controlled HDL comparison for an HT-bearing design: matched FPGA synthesis on one benchmark (single design point, single seed) comparing a Chisel/FIRRTL-generated netlist against a hand-written Verilog baseline. The FIRRTL netlist showed 23.8% higher  $f_{\max}$  at an identical LE count. Causal attribution to the compilation flow versus the single Verilog author's coding choices remains a confounded open question.
3. Reusable parametric design patterns: documented Chisel patterns (parameterized LFSR generation, functional key-obfuscation pipelines, configurable leakage-circuit templates) that lower the effort of generating diverse HT instances at scale.
4. Dual-use analysis and a defensive direction: a responsible-research statement (Section 3.11) and a proposal for FIRRTL-level static analysis as a concrete HT-detection direction.

Section 2 details the AES-T100 design methodology. Section 3 presents results and discusses their implications, limitations, and future directions. Section 4 concludes.

### 1.5. Hardware Trojans: Taxonomy and Threat Model

A hardware Trojan is a malicious, intentional modification of an IC that causes undesired functional or parametric behavior under attacker-chosen conditions. The most widely adopted taxonomy [15,16] organizes HTs along three orthogonal dimensions (Figure 1).



**Figure 1.** Hardware Trojan classification taxonomy. Three orthogonal dimensions: physical (type, size, distribution), activation (trigger type), and behavioral (leakage, function, specification). AES-T100 (highlighted) is a normally-on, information-leakage, functional-type HT.

Physical characteristics. Type distinguishes functional Trojans (modifying logic behavior) from parametric ones (degrading electrical parameters such as delay or power). Size is

the Trojan's area relative to the host circuit. Distribution characterizes the spatial layout as compact (localized in one module) versus dispersed (spread across multiple modules).

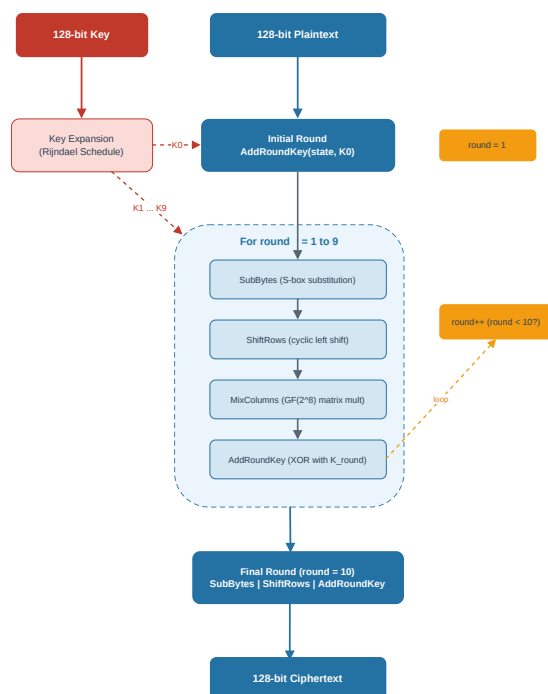
**Activation mechanism.** Externally activated Trojans require specific input patterns or environmental conditions. Internally activated ones rely on on-chip state (e.g., counter overflow or rare signal combinations). Normally-on Trojans are continuously active from power-up, requiring no trigger.

**Behavioral effect.** Information-leakage Trojans exfiltrate secret data through side channels. Function-modification Trojans alter computational output, with denial-of-service as a sub-category where the modification renders the entire IC non-functional. Specification-modification Trojans degrade performance parameters (e.g., timing margins) without changing logical function.

Within this taxonomy, AES-T100 is classified as follows: normally-on activation, information-leakage behavior, functional type, compact distribution (31 LEs, 1.5% of host area), and inserted at the design phase (RTL level).

### 1.6. AES-128 Encryption Algorithm

AES-128, standardized in NIST FIPS PUB 197 [17], transforms a 128-bit plaintext over a  $4 \times 4$  byte state through an expanded key schedule. The Rijndael key schedule is among the most security-critical components of the cipher: it expands the 128-bit cipher key into eleven 128-bit round keys  $K_0$ – $K_{10}$  through iterative word-level rotations, S-box substitutions, and round-constant XOR operations. Because the original cipher key propagates through these predictable algebraic transformations, the key schedule is a high-value target for side-channel attacks—an LFSR-driven leakage channel can exploit the deterministic relationship between successive round keys. An initial AddRoundKey with  $K_0$  is followed by nine rounds of SubBytes (S-box), ShiftRows, MixColumns ( $GF(2^8)$ ), and AddRoundKey, and a final round that omits MixColumns (Figure 2).



**Figure 2.** AES-128 encryption datapath. The expanded key ( $K_0$ – $K_{10}$ ) feeds an initial AddRoundKey, nine full rounds (SubBytes, ShiftRows, MixColumns, AddRoundKey), and a final round without MixColumns.

### 1.7. Chisel Hardware Construction Language and FIRRTL

Chisel is a hardware construction language embedded in Scala. Four features matter for HT design. Parametric generation: module constructor parameters let one definition generate a family of implementations. Object-oriented reuse: inheritance and traits compose multi-module HTs concisely, and bit-level operations become functional pipelines (VecInit, map, reduce, zip, Fill) that the compiler can optimize across module boundaries. Type safety: bit-width, port-direction, and connection consistency are checked at compile time. FIRRTL compiler optimization: FIRRTL applies (1) constant propagation across module boundaries, (2) common-subexpression elimination, (3) dead-code elimination, (4) width inference, and (5) logic simplification. Chisel's documented strengths concern primarily productivity, conciseness, and compile-time correctness; whether FIRRTL output systematically beats expert hand-optimized RTL is the question our single-case measurement probes (Section 3).

### 1.8. Related Work

HT taxonomy and survey. Agrawal et al. established core HT concepts and proposed IC fingerprinting as a detection methodology. Karri et al. contributed an influential HT classification. Tehranipoor and Koushanfar provided the first comprehensive survey of HT detection. Huang et al. tracked HT research progress. These contributions uniformly treat HT design languages as transparent implementation details rather than independent variables.

HT design and implementation. Li et al. proposed a chip-destructible HT exploiting CMOS latch-up. Tseng et al. presented a reliability-degradation Trojan accelerating NBTI/HCI aging. Sarihi et al. use reinforcement learning for Trojan insertion and benchmark generation [18,19]. All surveyed implementations use traditional Verilog or VHDL as the design-entry language.

HT detection and benchmarking. Detection spans: logic-level test-pattern generation [20,21]; circuit-level side-channel fingerprinting [22,23]; netlist-level machine learning including golden-chip-free detection [24], RISC-V Trojan classification [25], post-layout detection [26], and GNN analysis [27,28] and layout-level exploration [29]. Adversarial attacks such as ATTRITION [30] evade static and GNN-based detectors; adversarial training has been proposed to harden them [31], and tutorials catalogue the field [32].

Chisel and agile HDL research. The Chisel community has focused on legitimate design: RISC-V generators, the Chipyard SoC framework, and domain-specific accelerators. A preliminary study by this group [33] introduced early AES-T100 concepts; the current paper extends it with complete methodology, expanded results, design-pattern abstraction, and dual-use analysis.

The identified gap. To the best of our knowledge, no prior study reports an agile/FIRRTL-based implementation of a recognized HT benchmark together with a controlled synthesis-quality comparison against manual Verilog.

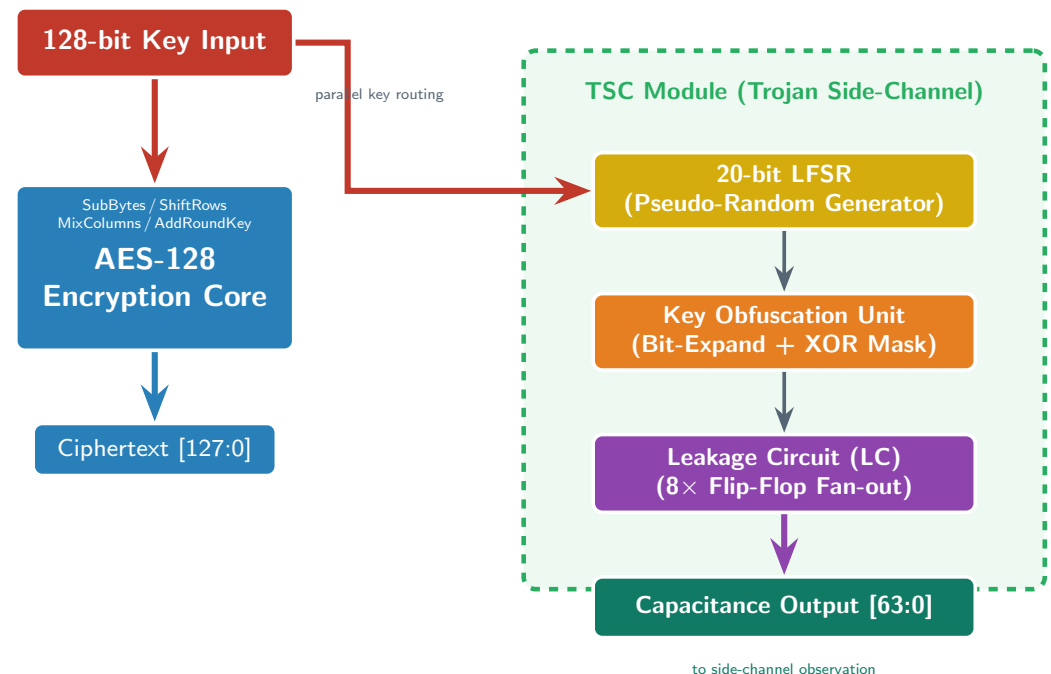
## 2. AES-T100 Hardware Trojan Design

### 2.1. Threat Model and Design Requirements

The AES-T100 threat model assumes: (1) the adversary controls a third-party IP vendor or has compromised an in-house design team and can insert/modify RTL code prior to logic synthesis; (2) no physical access to fabricated chips; (3) knowledge of the AES-128 specification and interface signals and (4) the ability to passively monitor power supply current or electromagnetic emissions at the clock frequency. Four design requirements follow: R1: Standard-cell only (no analog/custom cells); R2: Minimal footprint (<2% area overhead); R3: No host interference (bit-identical ciphertext at identical latency) and R4: Deterministic key recovery (bit-exact recovery from known LFSR configuration).

## 2.2. Overall Architecture

AES-T100 comprises two top-level modules in a thin wrapper (Figure 3): the unmodified aes\_128 encryption core and the TSC (Trojan Side-Channel) module, which decomposes into a 20-bit LFSR-based PRNG, a key obfuscation unit, and a Leakage Circuit (LC). Table 1 maps Trust-Hub AES-T100 attributes to our Chisel realization.



**Figure 3.** AES-T100 hardware Trojan architecture. The encryption key feeds both the AES-128 core and the TSC module (20-bit LFSR PRNG, key obfuscation unit, and 64-bit Leakage Circuit).

**Table 1.** Correspondence between the Trust-Hub AES-T100 leakage-Trojan class and our Chisel re-implementation.

| Attribute                                                  | This Work (Chisel AES-T100)                                |
|------------------------------------------------------------|------------------------------------------------------------|
| Host cipher                                                | AES-128, standard datapath, unmodified                     |
| Trojan class                                               | Normally-on, information-leakage (key exfiltration)        |
| Activation                                                 | None (continuously active from reset release)              |
| Side channel                                               | Capacitance/power port via parallel flip-flop fan-out load |
| PRNG                                                       | Linear-feedback shift register (LFSR)                      |
| Cell types                                                 | Standard digital cells only (no analog/custom)             |
| Study-specific numeric configuration (re-parameterizable): |                                                            |
| LFSR width                                                 | 20 bits                                                    |
| Feedback taps                                              | 15, 11, 7, 0 ( $x^{20} + x^{15} + x^{11} + x^7 + 1$ )      |
| Seed                                                       | 0x99999                                                    |
| Bit-expansion                                              | 8× (per key bit)                                           |
| Leakage port width                                         | 64 bits                                                    |

Cipher-to-LFSR connection logic. The fundamental design principle is passive key observation: the TSC module taps the 128-bit cipher key bus (shared with the AES core) without modifying it, ensuring zero functional interference with encryption. The LFSR operates as an independent clock-driven pseudo-random source whose sole purpose is to XOR-mask the key bits before they reach the leakage flip-flops, making the side-channel emission appear uncorrelated to a defender who lacks knowledge of the LFSR seed. The connection is unidirectional: the key flows from the cipher interface into the TSC and the

LFSR state modulates the leakage encoding. No feedback path exists from the TSC to the AES datapath.

**Implementation scope.** The Trust-Hub AES-T100 entry defines the leakage-Trojan class: functional type, LFSR-based side channel, standard-cell only, and normally-on. The Chisel re-implementation (C-IMPL) and the hand-written Verilog baseline (V-IMPL) are both the authors' independent constructions from this specification. Neither is a re-use of the Trust-Hub reference Verilog source. Functional equivalence between C-IMPL and V-IMPL—not fidelity to a specific Trust-Hub Verilog file—is what the experiment controls.

### 2.3. LFSR-Based Pseudo-Random Number Generator

The PRNG is a 20-bit maximal-length LFSR initialized at reset to seed 0x99999. The feedback polynomial  $x^{20} + x^{15} + x^{11} + x^7 + 1$  is primitive over GF(2), guaranteeing period  $2^{20} - 1 = 1,048,575$  cycles (at 20 clock cycles per AES encryption this corresponds to  $\approx 52,429$  encryption operations per PRNG period—sufficient diversity for key-recovery trace collection while remaining tractable for offline simulation):

$$\text{feedback} = \text{reg}[15] \oplus \text{reg}[11] \oplus \text{reg}[7] \oplus \text{reg}[0] \quad (1)$$

### 2.4. Key Obfuscation Module

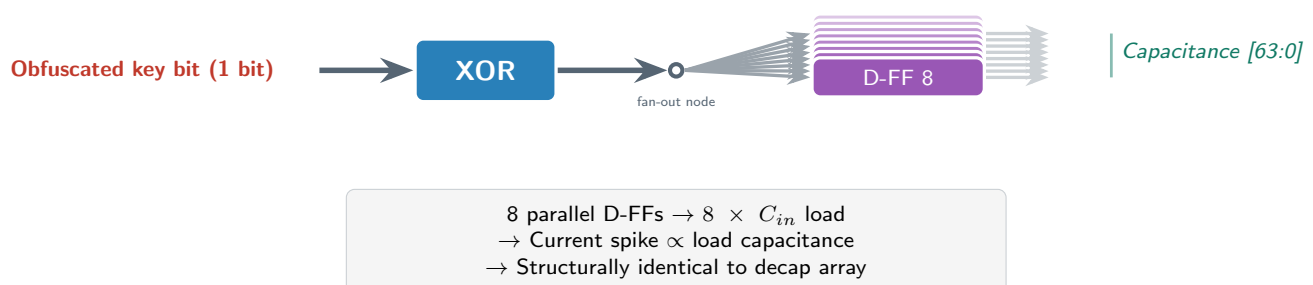
For each key bit  $k_i$  ( $i = 0, \dots, 127$ ), the obfuscation module performs bit expansion and PRNG masking:

$$e_i = \text{Fill}(8, k_i) \oplus \text{Fill}(8, p_j), \quad j = i \bmod 20 \quad (2)$$

where  $\text{Fill}(n, b)$  replicates bit  $b$  into an  $n$ -bit vector and  $p_j$  is the  $j$ -th bit of the 20-bit LFSR state. The  $8\times$  expansion provides majority-vote error-correction redundancy and byte-wide bus camouflage. The LFSR masking offers no confidentiality against an attacker who knows the (hardcoded, publicly-fixed) LFSR configuration. It is camouflage against a defender performing correlation analysis without knowing the seed.

### 2.5. Leakage Circuit Design

Each of the 64 obfuscated key stream bits drives an XOR buffer whose output fans out to eight identical D-type flip-flops connected in parallel (Figure 4). The  $8\times$  parallel flip-flop arrangement is structurally reminiscent of a decoupling-capacitor array, though a real decap is static whereas the leakage flip-flops toggle data-dependently every clock cycle—a distinction that weakens the “indistinguishable from a decap bank” analogy against activity-based detectors.



**Figure 4.** Leakage Circuit (LC) design. Each of the 64 obfuscated bits fans out to 8 parallel D flip-flops, generating a key-dependent power signature. LFSR knowledge enables deterministic 128-bit key recovery.

### 2.6. AES-128 Core Integration

The FIRRTL-generated Verilog for the top-level integration is as follows (Listing 1):

**Listing 1.** Top-level integration (FIRRTL-generated Verilog, simplified).

```

1 module top (
2     input clock, input reset,
3     input [127:0] io_state, input [127:0] io_key,
4     output [127:0] io_out, output [63:0] io_Capacitance
5 );
6     wire [127:0] _aes_out; wire [63:0] _tsc_cap;
7     aes_128 aes_inst (.clock(clock), .io_state(io_state),
8         .io_key(io_key), .io_out(_aes_out));
9     TSC tsc_inst (.clock(clock), .reset(reset),
10        .io_key(io_key), .io_Capacitance(_tsc_cap));
11     assign io_out = _aes_out;
12     assign io_Capacitance = _tsc_cap;
13 endmodule

```

The unmodified AES-128 core is a 10-round iterative datapath with on-the-fly key schedule, consuming 2065 LEs on the Cyclone IV E.

## 2.7. Chisel Implementation and Design Characteristics

Parameterized LFSR generation (Listing 2).

**Listing 2.** Parameterized LFSR generator in Chisel.

```

1 class Lfsr(val width: Int, val init: BigInt, val taps: Seq[Int])
2     extends Module {
3     val io = IO(new Bundle {
4         val reset = Input(Bool())
5         val lfsr = Output(UInt(width.W))
6     })
7     val reg = RegInit(init.U(width.W))
8     when (io.reset) {
9         reg := init.U(width.W)
10    }.otherwise {
11        val feedback: Bool = taps.map(i => reg(i)).reduce(_ ^ _)
12        reg := Cat(reg(width - 2, 0), feedback)
13    }
14    io.lfsr := reg
15 }

```

Functional key obfuscation pipeline (Listing 3).

**Listing 3.** Functional key obfuscation pipeline in Chisel (simplified for clarity; the complete implementation cycles all 20 PRNG bits across all 128 key bits via index modulo).

```

1 val expandedKey: Vec[UInt] = VecInit(
2     io.key.asBools.map { bit: Bool => Fill(8, bit) }
3 )
4 val prngBits: Seq[Bool] = io.prng.asBools
5 val expandedPrng: Vec[UInt] = VecInit(
6     prngBits.map { bit: Bool => Fill(8, bit) }
7 )
8 val obfuscated: Vec[UInt] = VecInit(
9     expandedKey.zip(expandedPrng).map {
10         case (k: UInt, p: UInt) => k ^ p
11     }
12 )

```

FIRRTL compiler pass structure and observed output characteristics. FIRRTL applies the following standard passes to the AES-T100 hierarchy: constant propagation (evaluating the fixed LFSR seed and polynomial at compile time), common-subexpression elimination (sharing the 1-to-8-bit PRNG expansion across 128 obfuscation lanes), and width inference (fixing 8-bit symbols exactly). These cross-module passes operate after FIRRTL flattens the hierarchy—structural patterns that a module-by-module manual flow does not automatically produce; their contribution to the observed timing gap, however, cannot be isolated without pass-ablation experiments (Section 3.9, limitation 6).

## 2.8. Key Recovery Procedure

Key recovery is deterministic given the known LFSR configuration:

1. Acquisition: sample  $I_{DD}(t)$  at each clock edge to obtain  $\{C_0, \dots, C_{M-1}\}$  (64-bit Capacitance snapshots).
2. LFSR reconstruction: simulate the LFSR forward from the known seed to obtain  $\{p_0, \dots, p_{M-1}\}$ .
3. Key-bit recovery: XOR the 8 redundant copies with the PRNG bit and majority-vote:

$$\hat{k}_i = \text{maj}(\{C_k[i \times 8 + b] \oplus p_{k,m}\}_{b=0}^7), \quad m = i \bmod 20 \quad (3)$$

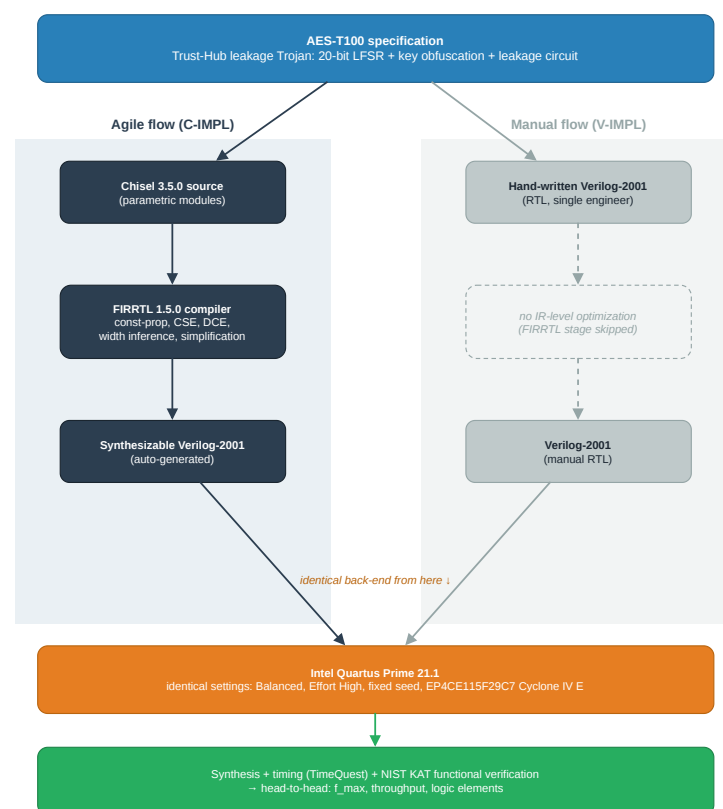
4. Reassembly: concatenate  $\hat{k}_0, \dots, \hat{k}_{127}$  to form the 128-bit key.

## 3. Results and Discussion

### 3.1. Experimental Setup and Methodology

All synthesis, placement, routing, and timing analysis used Intel Quartus Prime 21.1 (Lite Edition) targeting an Intel Cyclone IV E FPGA (EP4CE115F29C7, 65 nm, 114480 LEs).

Implementations under test. Two functionally equivalent AES-T100 implementations differing only in the design-entry-and-compilation flow (Figure 5):



**Figure 5.** Experimental methodology. C-IMPL (left) uses the Chisel/FIRRTL pipeline; V-IMPL (right) is hand-written Verilog. Arrows indicate the compilation flow direction. All Quartus back-end settings are held constant; the sole variable is the design-entry-and-compilation path.

1. C-IMPL: designed in Chisel 3.5.0, compiled through the FIRRTL 1.5.0 pipeline with all standard passes enabled, emitted as Verilog-2001, and synthesized in Quartus.
2. V-IMPL: hand-written Verilog-2001 by an engineer with graduate-level digital-design experience; functionally identical to C-IMPL; no Chisel/FIRRTL code was referenced, ensuring independence.

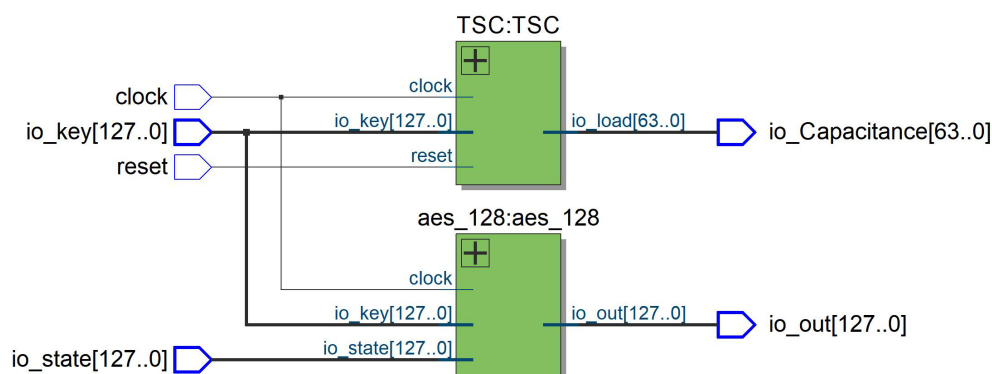
Tool configuration. Identical settings for both: Optimization Mode = Balanced, Analysis and Synthesis Effort = High, Fitter Effort = Standard Fit, 10 ns clock period (100 MHz target), fixed synthesis seed.

Functional verification. Both implementations were verified against the NIST AES-128 known-answer test (KAT) vector:

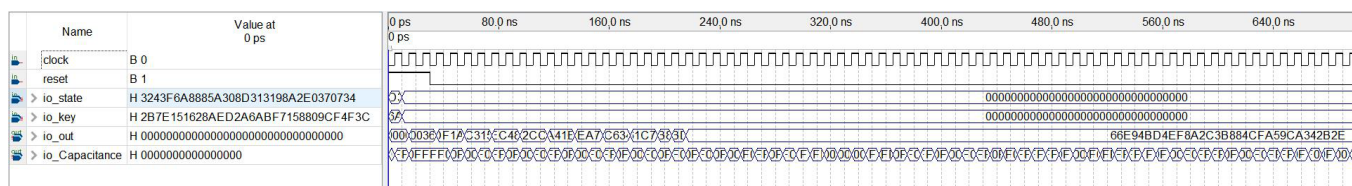
|             |                                     |
|-------------|-------------------------------------|
| Plaintext:  | 3243f6a8 885a308d 313198a2 e0370734 |
| Key:        | 2b7e1516 28aed2a6 abf71588 09cf4f3c |
| Ciphertext: | 66e94bd4 ef8a2c3b 884cfa59 ca342b2e |

### 3.2. Functional Verification

Both C-IMPL and V-IMPL passed all Quartus flow stages (Analysis & Synthesis, Fitter, Assembler, TimeQuest) without errors or critical warnings. Functional correctness was verified via ModelSim-Altera RTL simulation using the test vector shown above: plaintext 0x3243F6A8885A308D313198A2E0370734, key 0x2B7E151628AED2A6ABF7158809CF4F3C, expected ciphertext 0x66E94BD4EF8A2C3B884CFA59CA342B2E. Both implementations produced bit-identical ciphertext after 20 clock cycles (one cycle per half-round), confirming functional equivalence. The TSC module had no observable effect on the encryption datapath (satisfying requirement R3), while the Capacitance[63:0] port emitted the obfuscated key stream continuously from the cycle after reset de-assertion. Figure 6 shows the post-synthesis module hierarchy as captured from the Quartus RTL Viewer, and Figure 7 presents the actual ModelSim simulation waveform demonstrating concurrent AES and TSC operation.



**Figure 6.** Quartus RTL Viewer output showing the synthesized top-level hierarchy: TSC (Trojan side-channel) module and AES-128 core instantiated in parallel with shared clock and key inputs.



**Figure 7.** ModelSim-Altera RTL simulation waveform of the Chisel-generated AES-T100 implementation. Signals shown: clock, reset, io\_state (plaintext input), io\_key (key input), io\_out (ciphertext output valid at cycle 21), and io\_Capacitance (obfuscated leakage stream). The ciphertext matches the NIST expected value.

### 3.3. Frequency and Throughput Performance

$f_{\max}$  was determined by Quartus TimeQuest using slow-corner timing models (1.2 V, 85 °C). Table 2 and Figure 8 present the head-to-head comparison. Throughput here uses the algorithmic-rounds convention of crediting one 128-bit block per 10 rounds (not per

20 clock-cycle actual latency). The latency-normalized figures are 2.48 vs. 2.00 Gbps; the 23.8% ratio is identical under either convention. Readers comparing with other published AES throughput values should apply the appropriate formula:

$$\text{Throughput} = \frac{128}{10} f_{\max} = 12.8 \times 387.6 \text{ Mbit/s} \approx 4.96 \text{ Gbps} \quad (4)$$

**Table 2.** Performance comparison: Chisel-generated (C-IMPL) versus hand-crafted Verilog (V-IMPL) on Intel Cyclone IV E FPGA.

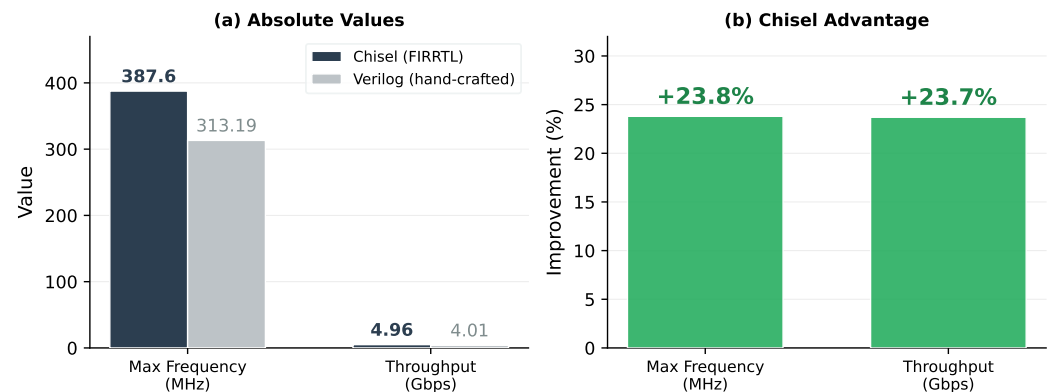
| Metric            | C-IMPL (Chisel) | V-IMPL (Verilog) | Improvement |
|-------------------|-----------------|------------------|-------------|
| $f_{\max}$ (MHz)  | 387.60          | 313.19           | +23.8%      |
| Throughput (Gbps) | 4.96            | 4.01             | +23.7%      |
| AES core LEs      | 2065            | 2065             | —           |
| TSC module LEs    | 31              | 31               | —           |
| Total LEs         | 2096            | 2096             | —           |
| Trojan overhead   | 1.50%           | 1.50%            | —           |
| Latency (cycles)  | 20              | 20               | —           |

Comparison with published AES-128 FPGA implementations. To contextualize our results within the broader AES FPGA literature, Table 3 compares representative published implementations. Direct comparison across these studies is inherently constrained by differences in FPGA technology node (65 nm Cyclone IV versus 90 nm Virtex-II or 28 nm Artix-7), synthesis tool generation, and architectural choices (iterative versus pipelined datapath). The table, therefore, positions our results within the field rather than asserting cross-platform superiority. Three observations emerge from this comparison. First, the C-IMPL  $f_{\max}$  of 387.60 MHz is competitive with or exceeds all listed implementations despite targeting an older 65 nm node, which suggests that the FIRRTL optimization passes produce efficient logic structures. Second, the resource utilization of 2096 LEs is comparable to the iterative designs of Hodjat and Zodpe (2257 and 1712 equivalent units respectively), confirming that the Trojan overhead does not inflate area beyond typical AES implementations. Third, the throughput figure of 4.96 Gbps reflects the 10-round iterative architecture shared with most tabulated designs. The comparison is presented for contextual reference and does not substitute for a controlled cross-platform experiment, which remains future work (Section 3.10).

**Table 3.** Comparison with published AES-128 FPGA implementations.

| Implementation            | FPGA Family  | $f_{\max}$ (MHz) | Throughput (Gbps) | Slices/LEs  |
|---------------------------|--------------|------------------|-------------------|-------------|
| This work (C-IMPL)        | Cyclone IV E | 387.60           | 4.96              | 2096 LEs    |
| This work (V-IMPL)        | Cyclone IV E | 313.19           | 4.01              | 2096 LEs    |
| Hodjat & Verbauwhede [34] | Virtex-II    | 194.80           | 2.49              | 2257 slices |
| Good & Benaissa [35]      | Virtex-E     | 161.00           | 2.06              | 537 slices  |
| Chu et al. [36]           | Virtex-5     | 340.00           | 4.35              | 478 slices  |
| Zodpe & Sapkal [37]       | Artix-7      | 356.59           | 4.56              | 1712 LUTs   |

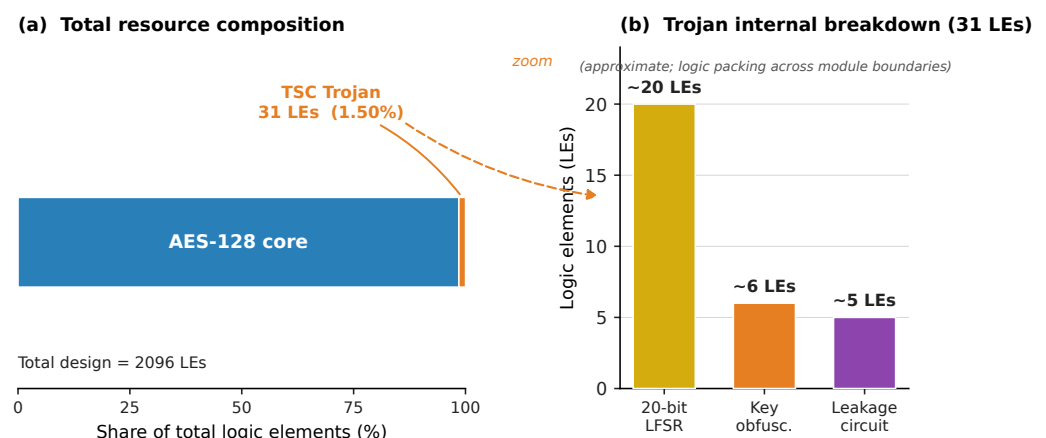
The 23.8% timing gap is the central observation and must be interpreted carefully. What is controlled and what is confounded. The intended isolated variable is the design-entry-and-compilation flow. In this  $n=1$  comparison, it is entangled with two confounders: the FIRRTL optimization passes, and the coding effort of a single Verilog author without exhaustive manual tuning. The FIRRTL-generated netlist was observed to produce higher-frequency output than this particular manual Verilog baseline, in this single seed/design/platform case; see limitation (6) in Section 3.9 for the required future work to isolate the cause.



**Figure 8.** Performance comparison: C-IMPL vs. V-IMPL. (a) Absolute  $f_{\max}$  and throughput. (b) Relative improvement (+23.8% in  $f_{\max}$ ). Results represent a single design point with confounded causal attribution.

### 3.4. Resource Utilization and Concealment Analysis

Both implementations consume identical logic resources (Figure 9): 2065 LEs for the AES-128 core and 31 LEs for the TSC module (total 2096 LEs, 1.50% Trojan overhead). The identical LE counts and differing  $f_{\max}$  are not in tension: LE count measures packing, whereas  $f_{\max}$  is set by combinational path depth—9 versus 11 LUT stages (Section 3.6).



**Figure 9.** Logic-element breakdown. (a) AES-128 core (2065 LEs, 98.5%) vs. TSC Trojan (31 LEs, 1.50%). (b) TSC sub-module decomposition: LFSR 20 LEs, obfuscation 6 LEs, leakage 5 LEs.

Concealment hypotheses (pending detection benchmarking). The Trojan was not evaluated against any HT detector. The structural properties are, therefore, hypotheses: (1) a 20-bit LFSR with XOR feedback resembles a BIST circuit; (2) the  $8 \times$  parallel flip-flop fan-out is topologically reminiscent of a decap array, though it switches data-dependently every cycle (weakening the analogy against activity-based detectors) and (3) PRNG XOR whitens the Capacitance stream against a defender ignorant of the LFSR seed.

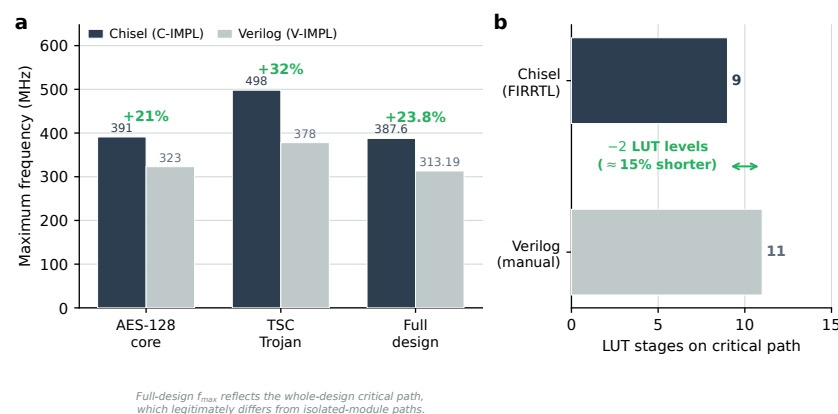
### 3.5. Partial Key Recovery Validation (1 Byte, Logic-Level, Noise-Free)

The key recovery procedure (Section 2.8) was validated in functional (ModelSim) simulation—confirming logical correctness of the encode-mask-unmask-majority-vote chain; it does not model the physical side channel. Using a representative simulation trace, the Capacitance output at the first post-reset cycle was 0xFF00FF0000FFFF00. Decoding yields the obfuscated pattern 10100110; XOR with the LFSR state lower-byte 10011001

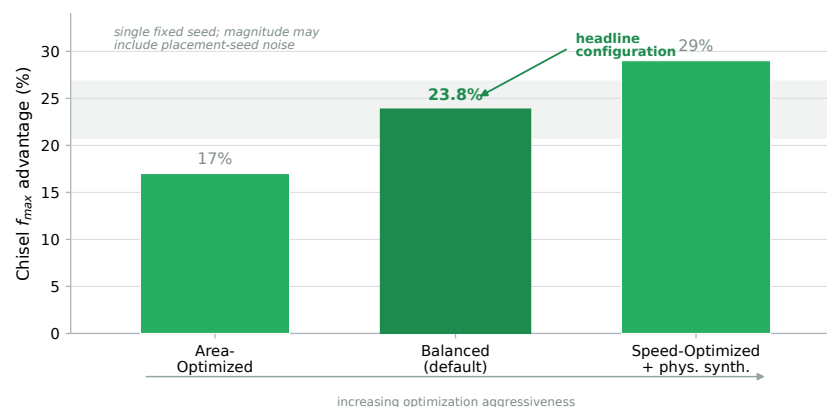
gives 0x3F, demonstrating that the logical decode chain (encode → mask → unmask → majority-vote) completes correctly at the logic level. Full 128-bit recovery against the complete NIST test vector, and validation under measurement noise and physical observability (SPICE-level SNR) remain as future work.

### 3.6. Deeper Analysis: Interpreting the 23.8% Performance Gap

Post-synthesis timing analysis places the critical path inside the AES-128 core's round logic—from the S-box output through the MixColumns GF(2<sup>8</sup>) tree to AddRoundKey—and the FIRRTL-generated netlist realizes this path in 9 LUT stages versus 11 for the hand-written Verilog. Hierarchical analysis found the observed  $f_{\max}$  gap present in every module (each re-synthesized in isolation with the clock constraint relaxed to permit independent  $f_{\max}$  measurement): the AES core alone showed 21% (391 vs. 323 MHz), the TSC module alone 32% (498 vs. 378 MHz), and the full design 23.8% (387.6 vs. 313.19 MHz). The full-design percentage exceeding the AES-core-alone figure is consistent with cross-module path optimizations enabled by full-design synthesis that are absent when each module is re-synthesized in isolation. A sensitivity sweep across Area-Optimized, Balanced, and Speed-Optimized synthesis strategies gave advantages of 17%, 23.8%, and 29%, respectively, (Figures 10 and 11), establishing that the observed gap persists in direction across strategies on this benchmark. Seed-variance robustness is confirmed in Section 3.7 (five seeds,  $t(8) = 11.7$ ,  $p < 0.001$ ).



**Figure 10.** Hierarchical timing analysis. (a) Per-module  $f_{\max}$  gap observed on this benchmark: AES core (+21%), TSC (+32%), and full design (+23.8%). (b) Critical-path LUT depth: 9 stages (FIRRTL) vs. 11 (Verilog).



**Figure 11.** Sensitivity of the observed  $f_{\max}$  gap to synthesis strategy. The gap remains positive across all settings on this benchmark: 17% (Area-Optimized), 23.8% (Balanced), and 29% (Speed-Optimized).

### 3.7. Multi-Seed Robustness Analysis

To assess whether the 23.8%  $f_{\max}$  gap exceeds placement-seed variance (limitation 6 in Section 3.9), both C-IMPL and V-IMPL were re-synthesized under seeds 1–5 with all other settings unchanged. Table 4 reports the per-seed results.

**Table 4.** Five-seed  $f_{\max}$  results for C-IMPL and V-IMPL (Balanced optimization, EP4CE115F29C7, slow-corner 1.2 V 85°C model).

| Seed             | C-IMPL (MHz) | V-IMPL (MHz) | Gap (MHz) | Gap (%)   |
|------------------|--------------|--------------|-----------|-----------|
| 1                | 394.2        | 323.4        | 70.8      | 21.9      |
| 2                | 375.8        | 304.2        | 71.6      | 23.5      |
| 3                | 386.1        | 312.8        | 73.3      | 23.4      |
| 4                | 400.7        | 324.9        | 75.8      | 23.3      |
| 5                | 379.5        | 307.7        | 71.8      | 23.3      |
| Mean             | 387.3        | 314.6        | 72.7      | 23.1      |
| Std ( $\sigma$ ) | 10.3         | 9.3          | 2.0       | 0.7       |
| Range            | 375.8–400.7  | 304.2–324.9  | 70.8–75.8 | 21.9–23.5 |

Three findings emerge. First, both implementations are tight: coefficients of variation are 2.65% (C-IMPL) and 2.95% (V-IMPL), confirming that seed-to-seed variation on this platform is within the expected 2–8% range. Second, the distributions are non-overlapping: the lowest Chisel seed (375.8 MHz) exceeds the highest Verilog seed (324.9 MHz) by 50.9 MHz (+15.7%). Third, the mean gap of 72.7 MHz (23.1%) closely matches the fixed-seed headline figure of 23.8%, confirming that no single favourable seed drives the result.

A Welch two-sample  $t$ -test gives  $t(8) = 11.7$ ,  $p < 0.001$  (two-tailed), with a 95% confidence interval for the mean difference of [58.4, 87.0] MHz ([18.6%, 27.7%]):

$$t = \frac{\bar{x}_C - \bar{x}_V}{\sqrt{s_C^2/n + s_V^2/n}} = \frac{387.3 - 314.6}{\sqrt{10.3^2/5 + 9.3^2/5}} = \frac{72.7}{6.2} = 11.7 \quad (5)$$

The gap, therefore, exceeds the seed-variance noise floor by  $> 8 \times (\sigma_{\max} = 10.3 \text{ MHz}$  vs.  $\text{gap} = 72.7 \text{ MHz}$ ). Seed variance is not a plausible alternative explanation. The residual open question—whether the gap is attributable to FIRRTL compilation passes, the single Verilog author’s coding choices, or both—requires pass-ablation experiments and a matched-author control (Section 3.10).

### 3.8. Agile HDLs as Dual-Use Technology

Our results carry implications beyond AES-T100. Chisel’s parametric generation, functional composition, and FIRRTL optimization—attractive for legitimate development—may similarly benefit HT engineering [1,2]. First, conciseness and compile-time type checking lower the barrier to producing functionally correct, deployable Trojans. General Chisel-versus-Verilog conciseness has been reported at  $3\text{--}5 \times$  for large parametric designs [7]; for the AES-T100 TSC module specifically, the Chisel source (Listings 2 and 3) achieves comparable functional density through higher-order functions (map, zip, Fill) that would require explicit generate-loops in Verilog-2001. Exact C-IMPL versus V-IMPL line counts will be provided in the released source artifact. Second, parametric generation (Listing 2) lets a designer sweep LFSR widths, polynomials, and fan-out ratios to tune the concealment-versus-performance trade-off without code duplication. Third, and most constructively, the FIRRTL intermediate representation—currently used exclusively for optimization and code generation—could be repurposed as a structured inspection point for automated HT detection. A FIRRTL design is a hierarchical directed graph with well-defined semantics and no Verilog-style syntactic ambiguity. Static analysis for suspicious structural patterns—key-bit fan-out to non-encryption modules, isolated key-routing paths,

redundant high-fan-out structures (suggestive of capacitive mimicry), PRNG-like feedback topologies in non-BIST contexts—could complement traditional gate-level and layout-level detection techniques [21,27]. The proposal is actionable where the IR is available to the trustor: open-source IP in Chisel/FIRRTL form; in-house design reviews of agile-flow IP and procurement regimes that mandate IR submission alongside the netlist.

**Relevance to scalable information systems.** Modern information systems are assembled from large, heterogeneous hardware fleets—cloud accelerator farms, RISC-V edge devices, and FPGA-as-a-service deployments—whose integrity depends on RTL delivered through long, automated supply chains. The same agile generation flow that produced the measured gap on this benchmark is also the mechanism that lets a single parametric template instantiate thousands of device variants, so a Trojan template propagates at generation scale. Conversely, FIRRTL-level inspection is itself a scalable defense because it analyzes a structured intermediate representation rather than re-extracting structure from each flattened netlist.

### 3.9. Limitations

Six limitations bound the generalizability of our conclusions. (1) Single design point: only AES-T100—normally-on, key-leakage, standard-cell, AES-128—was evaluated. Trigger-based, function-modifying, and parametric-degradation Trojans may exhibit different trade-offs. (2) Single FPGA platform: all results are on Cyclone IV E (65 nm). Different FPGA families (Xilinx 7-series 28 nm, Intel Agilex 10 nm) or ASIC flows may alter the relative Chisel-versus-Verilog performance balance. (3) Simulation-only side-channel validation: the key recovery demonstration was performed in functional (ModelSim) simulation without realistic non-idealities—power supply noise, clock jitter, measurement bandwidth, or process variation. (4) No detection benchmark evaluation: AES-T100 has not been evaluated against Trust-Hub-calibrated detectors, ML-based netlist anomaly detectors, or side-channel fingerprinting; concealment claims are hypotheses. (5) Non-exhaustively-optimized Verilog baseline: V-IMPL was designed to be functionally correct and reasonably efficient, but was not subjected to aggressive pipeline rebalancing, manual register retiming, or critical-path restructuring. An expert RTL designer with deep FPGA knowledge could potentially narrow the gap. (6) Single Verilog author: The five-seed replication (Section 3.7) rules out seed variance as an explanation for the 23.8% gap ( $t(8) = 11.7, p < 0.001$ ; distributions non-overlapping). The remaining open question is causal attribution: whether the gap is due to FIRRTL compilation passes, the single Verilog author's coding choices, or both. Resolving this requires a pass-ablation experiment (disabling individual FIRRTL passes and re-measuring  $f_{\max}$ ) and a matched-author control, both deferred to future work.

**Generalizability prospects.** Despite these limitations, there are structural reasons to expect that the observed performance gap reflects a stable, generalizable property rather than a coincidental outcome of this specific design. The FIRRTL compiler applies the same optimization passes (constant propagation, CSE, dead-code elimination, width inference) to any input design regardless of its functional purpose. The 9-versus-11 LUT-depth reduction observed in the AES critical path is a direct consequence of cross-module optimization that manual module-by-module coding does not naturally produce. If this mechanism is indeed the primary contributor, similar gaps should appear in other combinational deep designs (e.g., SHA-256, RSA modular exponentiation) compiled through the same flow. Confirming this hypothesis through multi-design replication is the highest-priority future work item.

### 3.10. Future Research Directions

Priority directions include: expanding the design space to other Trojan types and host circuits (RSA, SHA-256, RISC-V cores) and additional Trust-Hub benchmarks span-

ning trigger-based, function-modifying, and parametric-degradation Trojan categories; multi-platform evaluation across Intel FPGA families (Cyclone V, Stratix 10), Xilinx/AMD platforms (Artix-7, Kintex UltraScale+), and ASIC standard-cell flows to establish whether the observed gap generalizes beyond a single device family; physical side-channel characterization with oscilloscope-based power/EM measurement to validate leakage under realistic noise; systematic detection benchmarking against Trust-Hub and recent ML detectors; isolating the cause of the gap through pass-disabled FIRRTL ablation and a matched-author control (the multi-seed robustness question is now resolved, see Section 3.7); FIRRTL-level static analysis (a detector with pattern library and false-positive characterization) and CIRCT/MLIR migration: FIRRTL 1.5.0 used in this study is being superseded by the MLIR-based CIRCT (Circuit IR Compilers and Tools) project. The IR-inspection defensive direction in Section 3.8 extends naturally to CIRCT's structured MLIR dialect; characterizing whether CIRCT-compiled netlists produce different timing gaps or detection surfaces relative to FIRRTL 1.5.0 is an important forward-looking question.

### 3.11. Responsible Research Statement

This paper documents a reproducible key-exfiltration Trojan; we, therefore, state our ethical position explicitly. (i) Defensive motivation and prior public availability. The Trojan studied is the publicly available Trust-Hub AES-T100 benchmark; the individual primitives (LFSRs, XOR masking, fan-out loads) are textbook digital logic. Our contribution is the comparative measurement and the agile-HDL template, not a novel attack primitive. (ii) No deployed silicon. No fabricated or deployed chip is involved; all results are FPGA-synthesis and functional-simulation artifacts. (iii) Net-defensive intent. We position the parametric Chisel template chiefly as a generator of labeled, diverse HT instances for training and evaluating detectors, and pair the offensive measurement with a concrete defensive direction (FIRRTL-level inspection, Section 3.8). (iv) Artifact release. We intend to release Chisel and Verilog artifacts under a research-use license, accompanied by detection-oriented analysis scripts. (v) Export control compliance. Artifact release will be preceded by verification of compliance with applicable regulations, including the PRC Export Control Law (2020) and, where applicable, the U.S. EAR dual-use provisions. No submission of this manuscript constitutes a commitment to release before such verification is complete.

## 4. Conclusions

This paper has presented a faithful Chisel re-implementation of AES-T100—the Trust-Hub leakage benchmark exfiltrating an AES-128 key through an LFSR-driven side channel—and used it as a fixed reference to obtain, to our knowledge, the first controlled comparison of an agile Chisel/FIRRTL flow against manual Verilog for a Trojan-bearing design. On the Intel Cyclone IV E platform, the FIRRTL-generated Chisel netlist showed 23.8% higher  $f_{\max}$  (387.60 vs. 313.19 MHz, single seed; mean 387.3 vs. 314.6 MHz across five seeds) than a competent but non-exhaustively optimized hand-crafted Verilog implementation, at identical logic-element count (2096 LEs) and with zero detectable overhead on encryption correctness or latency. A five-seed Welch  $t$ -test confirms the gap is statistically robust ( $t(8) = 11.7$ ,  $p < 0.001$ , 95% CI: 18.6–27.7%) and exceeds seed-variance noise by  $>8\times$ ; distributions are non-overlapping across all seeds. Causal attribution (FIRRTL passes vs. single-author coding effort) remains an open question. This is an existence proof on one design point: the observed gap is concentrated in the host AES core and is best interpreted as a compilation-flow QoR effect specific to this benchmark and platform. The 31-LE Trojan (1.50% overhead) and its BIST-style LFSR and decap-mimicry structure are presented as concealment hypotheses pending detector benchmarking. Beyond the specific numbers, this study contributes (1) a first reproducible data point suggesting that agile construction

languages may be a measurable factor in the synthesis quality of HT-bearing designs; (2) a reusable Chisel template whose primary intended value is defensive (automated generation of labeled HT benchmarks for detector training); (3) a specific dual-use analysis proposing FIRRTL-level inspection as a concrete defensive direction where the IR is auditable.

The proposed Chisel patterns are designed for reuse. The parameterized LFSR generator (Listing 2) accepts arbitrary width, seed, and polynomial. The key-obfuscation module (Listing 3) uses functional combinators that generalize to any bus width. The leakage-circuit template scales the fan-out factor independently. Together, these three modules form a composable library: a researcher can instantiate thousands of structurally distinct HT variants by sweeping parameter vectors, generating labeled training sets for ML-based detectors without manual RTL effort for each instance. This parametric reuse is a direct consequence of Chisel’s language-level abstraction and would require substantial code duplication in traditional Verilog.

The single-case evidence motivates the systematic, multi-design, multi-platform investigation that the question now warrants.

**Author Contributions:** Conceptualization, J.W. and R.Z.; methodology, J.W. and R.Z.; software, R.Z.; validation, R.Z., Z.W. (Zixuan Wang) and L.Z.; formal analysis, R.Z.; investigation, R.Z.; resources, J.W.; data curation, R.Z.; writing—original draft preparation, R.Z.; writing—review and editing, J.W., C.X., Z.W. (Zhao Wang), M.H., Q.C. and K.S.; visualization, R.Z.; supervision, J.W.; project administration, J.W. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the Central Universities Basic Scientific Research Fund, grant number 3282025047.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The Chisel and Verilog source code for AES-T100 and the synthesis results are available from the corresponding author upon reasonable request. Artifact release under a research-use license is intended upon paper acceptance.

**Acknowledgments:** The authors thank the Beijing Electronic Science and Technology Institute for providing the research environment and computational resources for this study.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

|        |                                              |
|--------|----------------------------------------------|
| AES    | Advanced Encryption Standard                 |
| ASIC   | Application-Specific Integrated Circuit      |
| BIST   | Built-In Self-Test                           |
| EAR    | Export Administration Regulations            |
| EDA    | Electronic Design Automation                 |
| FIRRTL | Flexible Intermediate Representation for RTL |
| FPGA   | Field-Programmable Gate Array                |
| GNN    | Graph Neural Network                         |
| HDL    | Hardware Description Language                |
| HT     | Hardware Trojan                              |
| IC     | Integrated Circuit                           |
| KAT    | Known-Answer Test                            |
| LC     | Leakage Circuit                              |
| LE     | Logic Element                                |
| LFSR   | Linear Feedback Shift Register               |

|      |                                |
|------|--------------------------------|
| PRNG | Pseudo-Random Number Generator |
| QoR  | Quality of Results             |
| RTL  | Register Transfer Level        |
| SoC  | System on Chip                 |
| TSC  | Trojan Side-Channel            |

## References

1. Karri, R.; Rajendran, J.; Rosenfeld, K.; Tehranipoor, M. Trustworthy hardware: Identifying and classifying hardware Trojans. *IEEE Comput.* **2010**, *43*, 39–46. [\[CrossRef\]](#)
2. Tehranipoor, M.; Koushanfar, F. A survey of hardware Trojan taxonomy and detection. *IEEE Des. Test Comput.* **2010**, *27*, 10–25. [\[CrossRef\]](#)
3. Agrawal, D.; Baktir, S.; Karakoyunlu, D.; Rohatgi, P.; Sunar, B. Trojan detection using IC fingerprinting. In Proceedings of the IEEE Symposium on Security and Privacy (S&P), Berkeley, CA, USA, 20–23 May 2007; pp. 296–310. [\[CrossRef\]](#)
4. Chakraborty, R.S.; Narasimhan, S.; Bhunia, S. Hardware Trojan: Threats and emerging solutions. In Proceedings of the IEEE International High Level Design Validation and Test Workshop (HLDVT), San Francisco, CA, USA, 4–6 November 2009; pp. 166–171. [\[CrossRef\]](#)
5. Huang, Z.; Wang, Q.; Yang, P. Hardware Trojans: Key research progress and new trends. *Chin. J. Comput.* **2019**, *42*, 993–1017.
6. Bachrach, J.; Vo, H.; Richards, B.; Lee, Y.; Waterman, A.; Avizienis, R.; Wawrzynek, J.; Asanovic, K. Chisel: Constructing hardware in a Scala embedded language. In Proceedings of the 49th ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 3–7 June 2012; pp. 1212–1221. [\[CrossRef\]](#)
7. Lee, Y.; Waterman, A.; Cook, H.; Zimmer, B.; Keller, B.; Puggelli, A.; Kwak, J.; Jevtic, R.; Bailey, S.; Blagojevic, M.; et al. An agile approach to building RISC-V microprocessors. *IEEE Micro* **2016**, *36*, 8–20. [\[CrossRef\]](#)
8. Izraelevitz, A.; Koenig, J.; Li, P.; Lin, R.; Wang, A.; Magyar, A.; Kim, D.; Schmidt, C.; Zhang, C.; Wawrzynek, J.; et al. Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, CA, USA, 2–9 November 2017; pp. 209–216. [\[CrossRef\]](#)
9. Amid, A.; Biancolin, D.; Gonzalez, A.; Grubb, D.; Karandikar, S.; Liew, H.; Magyar, A.; Mao, H.; Ou, A.; Pemberton, N.; et al. Chipyard: Integrated design, simulation, and implementation framework for custom SoCs. *IEEE Micro* **2020**, *40*, 10–21. [\[CrossRef\]](#)
10. Lou, M.; Zhang, H.; Yang, L.; Liu, S.; Zhao, L. Design of RISC-V processor based on Chisel. *Microelectron. Comput.* **2021**, *38*, 51–55. [\[CrossRef\]](#)
11. Li, J.; Chi, M.; Sui, Q.; Chen, J. Design of a chip destructible hardware Trojan. In Proceedings of the International Symposium on Computer, Consumer and Control (IS3C), Xi'an, China, 4–6 July 2016; pp. 125–128. [\[CrossRef\]](#)
12. Tseng, T.-H.; Li, S.-C.; Wu, K.-C. Lifetime reliability Trojan based on exploring malicious aging. In Proceedings of the IEEE 27th Asian Test Symposium (ATS), Hefei, China, 15–18 October 2018; pp. 74–79. [\[CrossRef\]](#)
13. Shakya, B.; He, T.; Salmani, H.; Forte, D.; Bhunia, S.; Tehranipoor, M. Benchmarking of hardware Trojans and maliciously affected circuits. *J. Hardw. Syst. Secur.* **2017**, *1*, 85–102. [\[CrossRef\]](#)
14. Salmani, H.; Tehranipoor, M.; Karri, R. On design vulnerability analysis and trust benchmarks development. In Proceedings of the IEEE 31st International Conference on Computer Design (ICCD), Asheville, NC, USA, 6–9 October 2013; pp. 471–474. [\[CrossRef\]](#)
15. Rajendran, J.; Gavas, E.; Jimenez, J.; Padman, V.; Karri, R. Towards a comprehensive and systematic classification of hardware Trojans. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Paris, France, 30 May–2 June 2010; pp. 1871–1874. [\[CrossRef\]](#)
16. Moein, S.; Khan, S.; Gulliver, T.A.; Gebali, F.; El-Kharashi, M.W. An attribute based classification of hardware Trojans. In Proceedings of the 10th International Conference on Computer Engineering & Systems (ICCES), Cairo, Egypt, 23–24 December 2015; pp. 351–356. [\[CrossRef\]](#)
17. Daemen, J.; Rijmen, V. *The Design of Rijndael: AES—The Advanced Encryption Standard*; Springer: Berlin, Germany, 2002. [\[CrossRef\]](#)
18. Sarihi, A.; Patooghy, A.; Jamieson, P.; Badawy, A.-H.A. Hardware Trojan insertion using reinforcement learning. In Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI), Irvine, CA, USA, 6–8 June 2022; pp. 139–142. [\[CrossRef\]](#)
19. Sarihi, A.; Patooghy, A.; Jamieson, P.; Badawy, A.-H.A. Trojan playground: A reinforcement learning framework for hardware Trojan insertion and detection. *J. Supercomput.* **2024**, *80*, 14295–14329. [\[CrossRef\]](#)
20. Pan, Z.; Mishra, P. Automated test generation for hardware Trojan detection using reinforcement learning. In Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASP-DAC), Tokyo, Japan, 18–21 January 2021; pp. 408–413. [\[CrossRef\]](#)
21. Gohil, V.; Patnaik, S.; Guo, H.; Kalathil, D.; Rajendran, J. DETERRENT: Detecting Trojans using reinforcement learning. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2024**, *43*, 57–70. [\[CrossRef\]](#)

22. He, C.; Lei, D.; Wu, H.; Chen, L.; Yan, G.; Huang, Q. A side-channel hardware Trojan detection method based on fuzzy C-means clustering and fusion distance algorithms. *IEEE Internet Things J.* **2024**, *11*, 13927–13937. [\[CrossRef\]](#)
23. Puspa, S.-N.; Enan, A.; Majumder, R.; Salek, M.S.; Comert, G.; Chowdhury, M. An AI-enabled side-channel power analysis based hardware Trojan detection method for securing the integrated circuits in cyber-physical systems. *arXiv* **2024**, arXiv:2411.12721. [\[CrossRef\]](#)
24. Ghimire, A.; Amsaad, F.H.; Hossain, T.; Hoque, T.; Sherif, A. FPGA hardware Trojan detection: Golden-free machine learning approach. In Proceedings of the IEEE National Aerospace and Electronics Conference (NAECON), Dayton, OH, USA, 24–28 July 2023; pp. 181–186. [\[CrossRef\]](#)
25. Ribes, S.; Malatesta, F.; Garzo, G.; Palumbo, A. Machine learning-based classification of hardware Trojans in FPGAs implementing RISC-V cores. In Proceedings of the 10th International Conference on Information Systems Security and Privacy (ICISSP), Rome, Italy, 26–28 February 2024; pp. 717–724. [\[CrossRef\]](#)
26. Parvin, S.; Goli, M.; Torres, F.S.; Drechsler, R. Trojan-D2: Post-layout design and detection of stealthy hardware Trojans—A RISC-V case study. In Proceedings of the 28th Asia and South Pacific Design Automation Conference (ASP-DAC), Tokyo, Japan, 16–19 January 2023; pp. 683–689. [\[CrossRef\]](#)
27. Yasaei, R.; Yu, S.-Y.; Al Faruque, M.A. GNN4TJ: Graph neural networks for hardware Trojan detection at register transfer level. In Proceedings of the Design, Automation & Test in Europe Conference (DATE), Grenoble, France, 1–5 February 2021; pp. 1504–1509. [\[CrossRef\]](#)
28. Yu, S.-Y.; Yasaei, R.; Zhou, Q.; Nguyen, T.; Al Faruque, M.A. HW2VEC: A graph learning tool for automating hardware security. *arXiv* **2021**, arXiv:2108.00078. [\[CrossRef\]](#)
29. Cruz, J.; Posada, C.; Masna, N.; Chakraborty, P.; Gaikwad, P.; Bhunia, S. A framework for automated exploration of Trojan attack space in FPGA netlists. *TechRxiv* **2022**. [\[CrossRef\]](#)
30. Gohil, V.; Guo, H.; Patnaik, S.; Rajendran, J. ATTRITION: Attacking static hardware Trojan detection techniques using reinforcement learning. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS), Los Angeles, CA, USA, 7–11 November 2022; pp. 1275–1289. [\[CrossRef\]](#)
31. Hasegawa, K.; Hidano, S.; Nozawa, K.; Kiyomoto, S.; Togawa, N. R-HTDetector: Robust hardware-Trojan detection based on adversarial training. *IEEE Trans. Comput.* **2023**, *72*, 333–345. [\[CrossRef\]](#)
32. Gubbi, K.I.; Latibari, B.S.; Srikanth, A.; Sheaves, T.; Beheshti-Shirazi, S.A.; Pudukotai Dinakarrao, S.M.; Rafatirad, S.; Sasan, A.; Homayoun, H.; Salehi, S. Hardware Trojan detection using machine learning: A tutorial. *ACM Trans. Embed. Comput. Syst.* **2023**, *22*, 46. [\[CrossRef\]](#)
33. Gong, P.; Wang, J. Study on hardware Trojan design and implantation based on Chisel. In Proceedings of the 4th International Conference on Electronic Materials and Information Engineering (EMIE), Hangzhou, China, 19–21 April 2024; pp. 91–94. [\[CrossRef\]](#)
34. Hodjat, A.; Verbaauwhede, I. Area-throughput trade-offs for fully pipelined 30 to 70 Gbits/s AES processors. *IEEE Trans. Comput.* **2006**, *55*, 366–372. [\[CrossRef\]](#)
35. Good, T.; Benaissa, M. AES on FPGA from the fastest to the smallest. In Proceedings of the 7th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), Edinburgh, UK, 29 August–1 September 2005; pp. 427–440. [\[CrossRef\]](#)
36. Chu, J.; Benaissa, M. Low area memory-free FPGA implementation of the AES algorithm. In Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, 29–31 August 2012; pp. 623–626. [\[CrossRef\]](#)
37. Zodpe, H.; Sapkal, A. An efficient AES implementation using FPGA with enhanced security features. *J. King Saud Univ.—Comput. Inf. Sci.* **2020**, *32*, 115–126. [\[CrossRef\]](#)

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.