A Systematic Comparison of Side-channel Countermeasures for RISC-V-based SoCs

Abolfazl Sajadi* , Nusa Zidaric* , Todor Stefanov* , Nele Mentens*†
*LIACS, Leiden University, Leiden, The Netherlands
† ES&S, COSIC, ESAT, KU Leuven, Leuven, Belgium

***Abstract*—In this study, we conduct a systematic comparison of various countermeasures against power side-channel attacks on RISC-V-based Systems-on-Chip implemented on an FPGA. Focusing on the AES cryptographic algorithm running on a RISC-V Ibex core, we evaluate a range of countermeasures, including software masking, hardware/software noise generation, and dummy instruction insertion. We also examine the impact of unprotected software implementations on the specialized CoCo-Ibex micro-architecture by evaluating the Tiny-AES implementation. Our findings reveal significant trade-offs between security and overhead, offering critical insights for selecting appropriate countermeasures tailored to the specific requirements of resource- and energy-constrained systems.**

***Index Terms*—Power Side-Channel Attacks, RISC-V, FPGA, Countermeasures, Tiny-AES, Ibex**

I. Introduction

Mobile edge computing (MEC) is a modern paradigm aimed at pushing applications, data, and services geographically closer to where they are requested. Edge devices utilize low-power Systems-on-Chip (SoCs) and are limited in resources and energy budgets. These SoCs commonly use a single CPU core, based on architectures such as RISC-V [1] or ARM [2], which can be realized on different implementation platforms. One such platform is a Field Programmable Gate Array (FPGA).

A significant concern for MEC systems is security because of their extensive connection with various nodes and users in the immediate vicinity. Nowadays, the majority of these systems use standardized cryptographic algorithms such as AES [3] in different modes of operation to ensure the confidentiality, integrity, and authenticity. Despite the theoretical robustness of these algorithms, their practical utilization puts them at risk of physical side-channel analysis (SCA) attacks. Furthermore, MEC systems frequently operate in unsupervised and remote locations, allowing adversaries to obtain direct physical access to the computing hardware. SCA exploits physical characteristics, such as the power consumption [4], electromagnetic emanations [5], or timing behavior [6] to obtain secret information like the cryptographic key. Power consumption measurements capture the switching activity of the transistors while the device performs a cryptographic operation. In circuits without protection mechanisms, the switching activity is data-dependent, i.e., it depends on variables such as the plaintext and the key. To increase resilience against such vulnerabilities, various countermeasures have been designed [7]–[9]. The choice of the countermeasure is a trade-off between increasing the security and minimizing overheads in terms of hardware resources, power consumption, and timing.

Therefore, in this work, we systematically compare the effectiveness and overhead of different countermeasures against power side-channel vulnerabilities for RISC-V-based SoCs implemented on an FPGA. To the best of our knowledge, our work is the first to conduct such systematic evaluation and comparison, which is the main novel contribution of the paper. We focus on Tiny-AES [10] software implementations executed on the commonly used Ibex RISC-V CPU core. We evaluate and compare the following countermeasures: software masking, obfuscation using hardware- and software-generated noise, and insertion of dummy instructions. In addition, we analyze Tiny-AES executed on CoCo-Ibex [11] to determine the effect of this specialized Ibex core micro-architecture on unprotected Tiny-AES software implementations.

This paper is organized as follows. Section II briefly discusses some related work and Section III the necessary background to understand our work. Section IV presents our experimental methodology and Section V discusses the experimental results. The conclusions are given in Section VI.

II. Related Work

In the past decades, AES has been subjected to rigorous cryptanalysis and SCA attacks, including power SCA. Early comprehensive works on this topic include [12], describing different power SCA and fault injection on both software and hardware implementations of AES, different leakage models and statistical analyses, and possible countermeasures. Recently published work [13], is focusing on power SCA on FPGA implementations of AES. Correlation Power Analysis (CPA) is particularly efficient, as it requires a smaller number of measured traces [14] than the originally proposed Differential Power Analysis (DPA) [4]. Another line of work is focusing on leakage detection, using methods such as Test Vector Leakage Assessment (TVLA), instead of key recovery attacks such as DPA or CPA. Recent work in [15] performed comparative analysis of leakage detection methods on two RISC-V cores. Among other experiments, they detected leakage from the execution of the Tiny-AES implementation on the Ibex core. Their primary focus was to identify the source of the leakage on the microarchitectual level, e.g., leakage from the ALU, without applying any countermeasures.

Since the first SCA attacks, numerous countermeasures based on different approaches have been designed. In general, the aim of SCA countermeasures is to minimize the dependence of power consumption on intermediate data values and/or executed operations. The countermeasures can be classified into two categories: hiding and masking. Hiding is trying to achieve either equal or random power consumption in each clock cycle. In our work, we focus on the latter, which can be achieved by randomizing execution times or by increasing the noise and thus decreasing the signal-to-noise ratio. Masking, on the other hand, involves concealing intermediate data values with random values (masks). Examples of hiding countermeasures are shuffling the order in which operations are executed, inserting dummy operations [12], on-chip noise generation [8] and the use of secure logic styles [7]. A comprehensive survey of masking schemes can be found in [16]. Examples span from generalized secret sharing ISW masking schemes [17], [18] to Instruction Set Extensions for RISC-V [19]–[21].

In contrast to the aforementioned related work, our work systematically evaluates and compares software and hardware countermeasures against power analysis attacks on RISC-V-based SoCs implemented on an FPGA with a focus on the Ibex and CoCo-Ibex RISC-V CPU cores. It provides important insights in the trade-offs between computational resources, delay, power/energy consumption, and SCA resistance.

III. Background

This section lays the foundation for understanding our methodology and the analysis of various countermeasures.

A. The Ibex core

Ibex is an open source 32-bit RISC-V CPU core. It was first introduced as "Zero-riscy" [22] within the PULP platform and has since then contributed to lowRISC, by which it is maintained and further developed. The core is implemented in SystemVerilog and provides support for multiple instruction set extensions: Integer (I,E), Integer Multiplication and Division (M), Compressed (C), and Bit Manipulation (B). It offers optional configurations, e.g., three variants of the Multiplier/Divider Block with different trade-offs between the area and the cycle count for multiplication instructions. We use the Fast Multi-Cycle Multiplier Block executing the `MUL` instruction in 3 cycles. The same long division algorithm is used regardless of the configuration, which requires 37 cycles. However, the first cycle performs a divide-by-zero check. If the check is positive the division terminates in 2 cycles.

B. The Secure-Ibex core

The Ibex CPU core integrates a range of additional security features that can be activated at runtime by using the CPU Control and Status Register `cpuctrl`. The *Data-Independent Timing* feature ensures that the execution time is constant, regardless of whether a branch is taken or not. Additionally, the premature termination of multiplication by zero/one and division by zero is eliminated. The *Dummy Instruction Insertion* feature randomly inserts dummy instructions. The frequency of insertion is controlled by the 3-bit `dummy_instr_mask` in the `cpuctrl` register. At runtime, the next dummy instruction is inserted in the cycle chosen randomly from the interval $[0, \mathrm{D}]$, where $\mathrm{D} = 1 + (\texttt{dummy\_instr\_mask}\|11)_2$.We are particularly interested in the two corner cases that we denote with *-max* and *-min*:

- *Secure-Ibex-max*: to achieve the maximum random insertion frequency, we set $\mathrm{D} = 1 + (00011)_2 = 4$;
- *Secure-Ibex-min*: to achieve the minimum random insertion frequency, we set $\mathrm{D} = 1 + (11111)_2 = 32$.

The inserted instruction is selected among four different instructions: `ADD`, `AND`, `MUL`, and `DIV`, for which the cycles-per-instruction (CPI) are 2, 2, 3, and 37 clock cycles, respectively. Additionally, the source registers for these instructions are selected randomly, and the destination register is set to 0.

C. The CoCo-Ibex core

CoCo-Ibex is relevant to our study as an Ibex CPU core modified for robust execution of masked software implementations. The CoCo tool [11] was employed to identify side-channel leakage sources within the Ibex CPU core netlist. CoCo is a formal verification tool designed to model leakage of masked software implementations in a time-constrained probing model at the gate level. It requires access to the CPU gate-level netlist and simulates the execution of a masked assembly implementation, where a sensitive value is split into two shares using internal randomness. If CoCo detects a recombination of shares within a specific cycle, it reports the gate and cycle as a leakage point. This helps determine whether the leakage originates from the software or the CPU's micro-architectural side effects. To mitigate secret-dependent glitches, gating is applied in the Ibex CPU core, on the one hand consisting of Register Gating to address issues in the register file such as switching wires in the multiplexer tree, glitchy address signals and unintended reads, and on the other hand consisting of Computation Unit Gating to disable always-active computation units and clearing the hidden state in the Load/Store Unit to prevent unintended leakage.

D. Correlation Power Analysis

In CPA, we compute Pearsons's correlation coefficient between the estimated power consumption (based on hypothetical calculations) and the measured power consumption. We focus on the absolute value of the correlation coefficient, as we are interested in the strength of this relationship. For our analysis, we employ the online calculation method from [23], [24], allowing the incorporation of new power traces without recalculations.

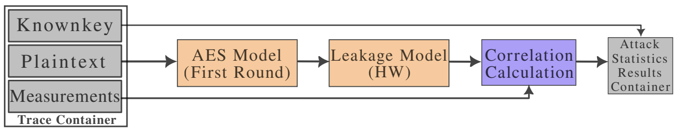


Fig. 1. Flowchart of Correlation Power Analysis [23]

Fig. 1 illustrates the data flow of CPA. The system utilizes online updates during the computation of attack statistics to monitor the progress as new traces are incorporated. Data

TABLE I. Implemented countermeasures

Type of Countermeasure	**Hardware**	**Software**
Algorithm-agnostic	Secure-Ibex	Secure-Ibex
	HW-NG	SW-NG
Algorithm-specific	CoCo-Ibex	Mask-AES

from the 'Trace Container' might undergo preprocessing or be selectively sampled from a specific window of data rather than using the entire trace range. The outcomes are recorded in the 'Attack Statistics Container', which logs results after a certain number of traces and computes metrics such as Partial Guessing Entropy (PGE) in the current state [23], [24].

IV. Experimental Methodology

Our study aims to systematically compare different countermeasures against power side-channel attacks when a specific cryptographic algorithm is executed on a RISC-V-based SoC implemented on an FPGA. We choose AES as the target algorithm because it is widely used and well known, and many countermeasures against side-channel analysis on AES have been proposed in the past decades. We select and compare the various types of countermeasures, shown in Table I, classifying them in two ways: (a) algorithm-agnostic vs. algorithm-specific, and (b) modifying/extending the hardware vs. modifying/extending the software.

In the algorithm-agnostic category, we analyze the impact of the security features of the Secure-Ibex core and evaluate their implementation overhead. These security features utilize specific hardware modules to alter the software execution time. Therefore, in Table I, the Secure-Ibex security features are classified as hardware and software countermeasures. Additionally, we analyze the effect of generating noise within the RISC-V-based SoC implemented on an FPGA by utilizing hardware (HW-NG) and software (SW-NG) techniques. In hardware, we generate noise using a specific Noise Generation module (Section IV-A3), which utilizes Shift Register LUTs (SRL) to randomize the power consumption. In software, we generate noise through redundant computation with different secret data, i.e., we mirror the S-box computation in AES with a different key and the same plaintext (Section IV-A4). Although mirroring the S-box is algorithm-specific, noise generation in software (SW-NG) through redundant computation is an algorithm-agnostic technique in general, thus we include it in both categories (the dashed line in Table I).

In the algorithm-specific hardware category, we use CoCo-Ibex, which helps reduce exploitable leakage in algorithms with protections based on secret sharing [25]. The original work on CoCo [11] focuses on test cases using various masking approaches, and presents results for S-box assembly implementations executed on CoCo-Ibex. We want to analyze whether CoCo-Ibex has any effect on unprotected algorithms, thus we use Tiny-AES as the cryptographic algorithm. Finally, we analyze a masked version of Tiny-AES using boolean masking [26],which we nickname Mask-AES. This algorithm-specific software countermeasure follows the description in [12].

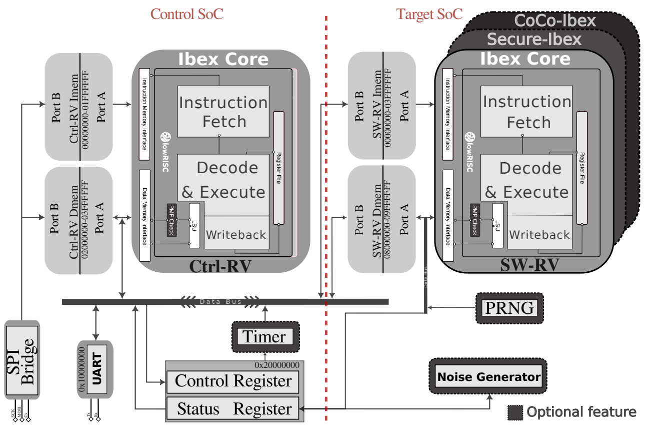


Fig. 2. Block Diagram of the Experimental Platform

A. *The Experimental Platform - Components and Features*

To conduct our experiments and gather actual data, such as timing performance and power consumption, we have developed an experimental platform. This platform enables side-channel analysis (CPA) of cryptographic software running on a general-purpose CPU. In our experiments, we use the platform for a systematic comparison of different countermeasures (Table I) against CPA on the Tiny-AES software running on the RISC-V CPU cores described in Section III. Fig. 2 shows our platform, which is implemented on an FPGA. In the platform, we have two parts: Control SoC and Target SoC. The Control SoC part contains an Ibex core (Ctrl-RV) and two 32-bit registers (Control Reg and Status Reg) used to control and monitor the Target SoC part. This part contains a target RISC-V CPU core (SW-RV) executing the cryptographic software. Before the platform synthesis, we select either the Ibex core, the Secure-Ibex core, or the CoCo-Ibex core as the target CPU core. Additionally, the Target SoC part includes optional units such as a Pseudo Random Number Generator (PRNG) and Noise Generator module. We use the Target SoC to run and analyze the cryptographic software, while the Control SoC is used to manage all data input/output and the activation/deactivation of components in the Target SoC. Such an approach increases the platform's flexibility and enables the analysis of different Target SoC configurations.

1) Control SoC Features: The Serial Peripheral Interface (SPI) protocol is utilized to initially program the Ctrl-RV. In this process, the SPI module, depicted in Fig. 2, receives memory data and places it in the specified locations. Subsequently, Ctrl-RV establishes a communication channel with a host computer using UART for the purpose of processing commands. Receiving commands from the host, the Ctrl-RV can load the target core (SW-RV) instruction and data memories, and activate the core by setting a flag in the Control Reg. The Ctrl-RV monitors the status of SW-RV by reading the Status Reg.

2) Target SoC Features: This experimental platform is specifically designed to function as a collaborative tool that establishes suitable triggers for the accurate acquisition of

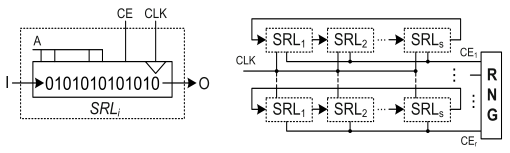


Fig. 3. Noise generation based on SRLs [8]

power (or other) traces, i.e., a cooperative target. For the trigger signal, we use one bit in Status Reg. The SW-RV can set this bit to make the trigger signal high or reset it to make the trigger signal low.

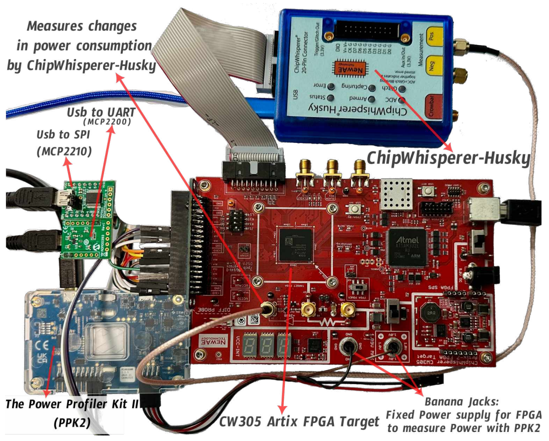
3) Noise Generation Module [HW-NG]: This module, introduced in [8], can be used to generate Gaussian noise and it utilizes SRLs. SRLs are highly effective for generating noise because they increase the toggle rate, which significantly impacts the power consumption of a gate in SRAM-based FPGA devices. By configuring LUTs as shift registers (SRLs), cascading them, and initializing them with alternating toggle bit patterns as depicted in Fig. 3, we can efficiently produce high noise levels. The enable signals CE of the cascaded SRLs are connected to a random number generator, thereby creating an effective noise source.

4) Noise Generating Software [SW-NG]: The S-box computation in AES running on SW-RV is mirrored with the same plaintext but a different key in Ctrl-RV to observe the effect of this noise technique. Mirroring the computation can be done by creating a specific hardware module, but in our platform, we have the ability to reuse Crtl-RV as such module.

B. The Experimental Platform - Hardware Setup

The experimental platform, described in Section IV-A, is implemented on the hardware setup, shown in Fig. 4, that includes the CW305 target board, featuring an XC7A100T Artix-7 FPGA. This board is designed for power analysis and fault injection attacks on cryptographic algorithms in FPGAs [23]. We use a 50 MHz input clock frequency for our implementation. To upload the control program executed by Ctrl-RV from the host computer, we use the MCP2210 chip (a USB-to-SPI Master converter) to connect the host with the Control SoC. Similarly, to establish the communication channel for sending commands from the host to the Control SoC, we use the MCP2200 chip (a USB-to-UART converter). The plaintexts and ciphertexts processed by SW-RV in the Target SoC are transmitted via this communication channel as well.

In order to capture power traces of the Target SoC, we employ the ChipWhisperer-Husky [27]SCA platform [23] shown in Fig. 4. We utilize the highest capturing frequency available, i.e., 200 MHz, which is four times higher than the 50 MHz clock frequency of our platform, in order to obtain the highest level of accuracy in the power traces. To capture each trace, we first send a randomly generated plaintext from the host computer through the UART to the platform. The Ctrl-RV places it in the SW-RV data memory and waits for the start signal from the ChipWhisperer-Husky, indicating it is ready to capture a trace. Upon receiving this signal, Ctrl-RV sets


Measures changes in power consumption by ChipWhisperer-Husky
Usb to UART (MCP2200)
Usb to SPI (MCP2210)
ChipWhisperer-Husky
The Power Profiler Kit II (PPK2)
CW305 Artix FPGA Target
Banana Jacks: Fixed Power supply for FPGA to measure Power with PPK2
Fig. 4. Hardware Setup for our Experimental Platform

the second bit of Control Reg to activate the SW-RV core, which then performs AES encryption. The attack targets the S-box computation in the first round of AES. Before executing the first round, SW-RV sets the trigger signal high, initiating the trace capturing on the ChipWhisperer-Husky, and sets the signal low at the end of the S-box computation. The ciphertext is stored in the data memory of SW-RV and sent by Ctrl-RV to the host computer via the UART for verification. We use the trigger signal to activate the HW-NG module or SW-NG execution when noise have to be generated.

To measure the power consumption of the Target SoC, we utilize the Power Profiler Kit II (PPK2), developed by Nordic Semiconductor, that is shown in Fig. 4. This is because ChipWhisperer-Husky uses AC-coupling, thus it only measures changes in the power consumption and not a direct DC offset. The resolution of this kit ranges from 100 nA to 1 mA, depending on the measuring range, and it offers a sampling speed of 100 kS/s. The CW305 board has a banana jack as an input power source, allowing to measure the power of the FPGA component on the board. According to the PPK2 specifications, our resolution is set to 50 uA. To enhance the precision, we decrease the clock frequency on the CW305 board to the minimum frequency of 630 kHz and measure the average power over 1000 consecutive measurements.

V. Experimental Results

In this section, we present and discuss the experimental results obtained by applying our methodology presented in Section IV.

A. Security Analysis of Target SoCs with Countermeasures

After the capturing of power traces, the next step is the security analysis of our Target SoC running Tiny-AES and implementing the countermeasures against power side-channel attacks shown in Table I. For this purpose, we use CPA [28], illustrated in Fig. 1, to try to reveal the secret key. To reduce the analysis time and number of traces needed to find the secret key successfully, we place the trigger signal in the first S-box of the first round of the Tiny-AES implementation.

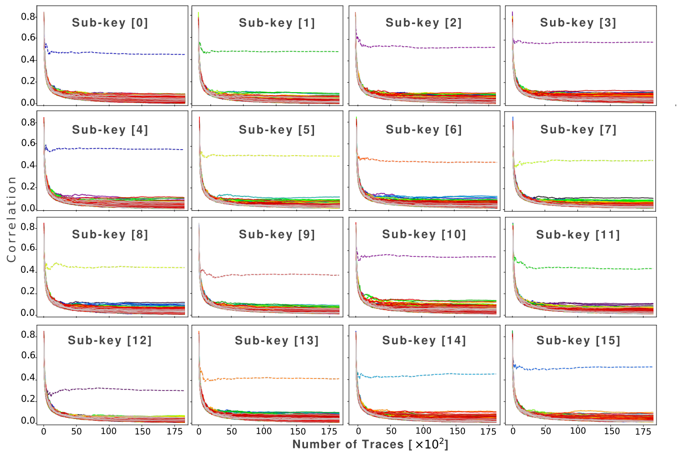


Fig. 5. Corr. vs #Traces for Tiny-AES on Ibex (unprotected)

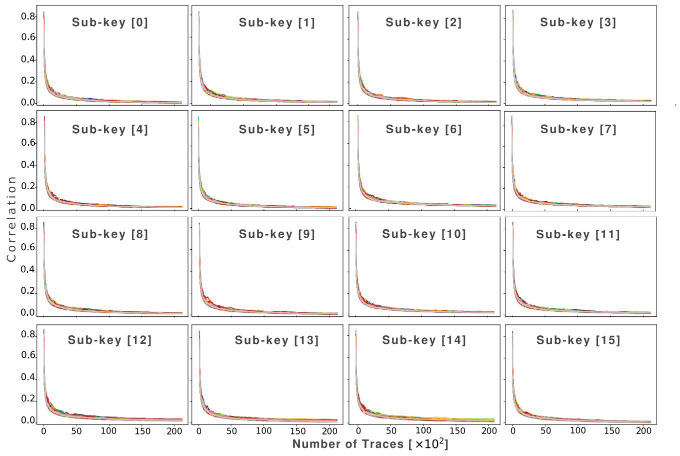


Fig. 6. Corr. vs #Traces for Software Mask-AES on Ibex

1) ***Tiny-AES on Ibex:*** This Target SoC configuration is our baseline because the AES algorithm is unprotected. We analyze 18K traces and Fig. 5 shows the correlation versus the number of traces for each sub-key when the Target SoC executes Tiny-AES on the SW-RV core, where a sub-key in AES is a smaller part of the key that is analyzed individually using a divide and conquer approach instead of analyzing the full key at once. The correct sub-key is indicated with a dashed line and each candidate sub-key is shown in a different color. The results indicate that to achieve a highly distinguishable correlation for all sub-keys in this baseline configuration, we need approximately 250 traces per sub-key to successfully reveal the full secret key.

2) ***Mask-AES on Ibex:*** In this Target SoC configuration, we examine the software masked version of Tiny-AES and increase the number of traces to 21K. Fig. 6 shows the correlation versus the number of traces for all sub-keys. In order to execute the Mask-AES software implementation, we activate the PRNG module in our experimental platform. The results indicate that, with our setup and using Power Side-Channel attacks, we cannot find any distinguishable correlation for any sub-key, thus the full secret key is not revealed.

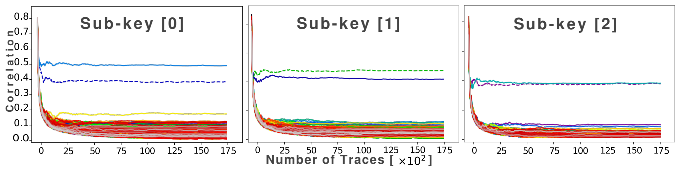


Fig. 7. Corr. vs #Traces for Tiny-AES on Ibex with SW-NG

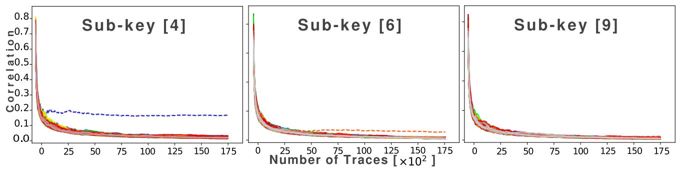


Fig. 8. Corr. vs #Traces for Tiny-AES on Ibex with HW-NG

3) ***Tiny-AES on Ibex with SW-NG:*** In this configuration, we generate software noise by mirroring the S-box computation in the Ctrl-RV using the same plaintext but a different key. According to the correlation equation in [23], no new correlation will be observed if the key in the mirroring technique is changed randomly for each measurement. This is because the CPA method calculates the correlation across all measurements and only a fixed key will produce a consistent correlation. When using a fixed and incorrect key in the mirrored S-box computation, two high correlations are observed in the results: one for the correct key and another for the incorrect key, as shown in Fig. 7. For some sub-keys, such as sub-key [0], the incorrect key (blue solid curve) shows a higher correlation. In other cases, like sub-key [2], the correct (dashed curve) and incorrect (solid curve) sub-keys exhibit nearly equal correlations. Despite these cases, the correct full key can still be identified using brute force attacks over the key candidates.

4) ***Tiny-AES on Ibex with HW-NG:*** In this configuration, we implement a noise generation module using SRLs, as shown in Fig. 3, by using as much as possible resources on the FPGA to construct a big noise generator. Fig. 8 shows results for three different cases of sub-keys. In some cases, we observed a high and distinguishable correlation, such as for sub-key [4], although this correlation is lower than when running Ibex without generating noise, and we need more traces to get it (at least 400 traces). In other cases, such as sub-keys [6] and [9], we find low correlation, and it is not distinguishable in some of them. However, the correlation increases with the number of traces, indicating that we can bypass this countermeasure by simply increasing the number of traces.

5) ***Tiny-AES on Secure-Ibex:*** In this Target SoC configuration, we use Secure-Ibex by activating the two security features described in Section III-B for the entire AES encryption process. This helps change the location of the S-box computation over time, making the attack more challenging. We analyze the two corner cases of randomly inserted dummy instructions described in Section III-B, namely *Secure-Ibex-min* and *Secure-Ibex-max*, using 18K traces. In both cases, we do not find any high or distinguishable correlation with the

TABLE II. Timing, Energy, and Power of Target SoC Configurations

Parameter	Ibex	Mask-AES	Secure-Ibex[a]		SW-NG	HW-NG	CoCo-Ibex
			max	min			
#Cycles	11559	16993	59315	16348	11559	11559	11559
Time (ms)	18.35	26.36	94.15	25.95	18.35	18.35	18.348
Power (mW)	31.42	31.42	33.86	33.86	31.47	37.41	31.45
Energy (μJ)	576.48	828	3188	878.65	577.41	686.39	577.04

[a] Average

TABLE III. Overhead of SoCs with Countermeasures

		LUTs	FFs	Code Size	Time
Mask-AES		-	-	2.03	1.47
Secure-Ibex	max	1.03	1.036	-	5.13
	min				1.41
HW-NG		3.98	-	-	-
SW-NG		-	-	1.34[a]	-
CoCo-Ibex		1.4	1.52	-	-

[a] Controller code size overhead.

correct key in any sub-key for our setup.

*6) **Tiny-AES on CoCo-Ibex**:* In this Target SoC configuration, we analyze Tiny-AES running on the CoCo-Ibex core to determine whether this core has any effect on leakage in unmasked AES. As expected, we obtain the same results as with the standard Ibex core, discussed in Section V-A1, indicating that CoCo-Ibex does not affect unmasked AES. Indeed, the CoCo-Ibex core is designed to effectively reduce leakage only in masked cryptographic algorithms based on secret sharing.

B. Overhead Analysis of Target SoCs with Countermeasures

Table II shows the performance and power/energy consumption of the six Target SoC configurations discussed in Section V-A. The numbers in Column 2 correspond to the baseline configuration *Tiny-AES on Ibex* where no countermeasures are implemented. The other columns correspond to the configurations with different countermeasures. Comparison of the numbers in Table II shows that some countermeasures can introduce significant overhead in terms of clock cycles, execution time, and power/energy consumption that is unavoidable when we want to enhance the security of a system. In particular, Secure-Ibex shows a wide range of overhead (Columns 4 and 5) depending on the activation of the security features, suggesting that careful tuning is necessary to balance the security with the performance and energy consumption.

Table III shows the overhead of each configuration with countermeasure (listed in Column 1) compared to the baseline configuration in terms of hardware resources (Columns 2 and 3), code size, and execution time. It can be seen that the HW-NG countermeasure exhibits a large overhead (3.98x) in terms of utilized LUTs in the FPGA implementation while no overhead is present in terms of code size and execution time. This is because the noise is generated by a large hardware module using SRLs that runs in parallel with the Tiny-AES algorithm on the Ibex core. The Mask-AES countermeasure shows the largest code size overhead (2.03x) because the masking is implemented in software, wheres the Secure-Ibex countermeasure exhibits significant and variable time overhead (in the range of 1.41x to 5.13x) due to the variable frequency at which dummy instructions are inserted.

Finally, Fig. 9 visually summarizes the security analysis results (Section V-A) and the overhead analysis results by qualitative comparison of all Target SoC configurations in terms of Attack Success (security) and Hardware/CodeSize/Time/Energy overhead introduces by the corresponding countermeasure. For example, comparing the Mask-

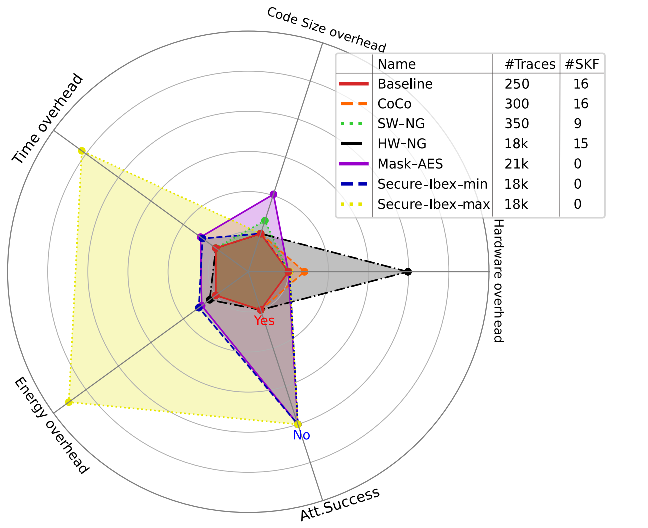


Fig. 9. Qualitative Comparison of Target SoC Configurations in Terms of Security and Overhead

AES (purple area) and Secure-Ibex-max (yellow area) configurations, it can be seen that the countermeasures in both configurations are equally effective in terms of security because we are not able to perform a successful attack. However, Secure-Ibex-max exhibits significantly larger Time and Energy overhead. In Fig. 9, $\#SKF$ shows the number of sub-keys found with a given number of traces denoted as $\#Traces$. For example, the Baseline and CoCo-Ibex configurations require around 250 traces to retrieve all 16 sub-keys, thus the attack is successful which is marked with "Yes" in the figure.

VI. Conclusions

We have systematically evaluated various hardware and software countermeasures against power side-channel attacks when running the Tiny-AES cryptographic algorithm on RISC-V-based SoCs implemented on an FPGA. Our experiments show that the unprotected Tiny-AES running on the Ibex RISC-V core is vulnerable, while different countermeasures offer varying trade-offs between security and overhead. Some countermeasures, such as noise generation, can be bypassed with exhaustive search or an extended number of power traces. Others, like adding dummy instructions and software masking, provide greater resilience but introduce an overhead in different areas. These findings underscore the critical need to balance security with performance, power/energy consumption, and resource utilization when selecting countermeasures, particularly for resource-constrained systems.

Acknowledgment

This work was funded by the Dutch Research Council (NWO) through the PROACT project (NWA.1215.18.014).

References

[1] P. Arul, N. Abirami, S. Sayeekumar, P. Vanmathi, and V. Annapoorani, "Implementation of risc-v instruction set architecture for edge iot computing platform," in *2024 Fourth International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, pp. 1–6, IEEE, 2024.

[2] T. Gomes, S. Pinto, T. Gomes, A. Tavares, and J. Cabral, "Towards an fpga-based edge device for the internet of things," in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pp. 1–4, 2015.

[3] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1999.

[4] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology–CRYPTO'99, LNCS, Vol.1666*, pp. 388–397, Springer Berlin Heidelberg, 1999.

[5] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *Cryptographic Hardware and Embedded Systems — CHES 2001* (Ç. K. Koç, D. Naccache, and C. Paar, eds.), (Berlin, Heidelberg), pp. 251–261, Springer Berlin Heidelberg, 2001.

[6] P. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology–CRYPTO'96, LNCS, Vol.1109*, pp. 104–113, Springer Berlin Heidelberg, 1996.

[7] K. Tiri and I. Verbauwhede, "Secure logic synthesis," in *Field Programmable Logic and Application* (J. Becker, M. Platzner, and S. Vernalde, eds.), (Berlin, Heidelberg), pp. 1052–1056, Springer Berlin Heidelberg, 2004.

[8] T. Güneysu and A. Moradi, "Generic side-channel countermeasures for reconfigurable devices," in *Cryptographic Hardware and Embedded Systems – CHES 2011* (B. Preneel and T. Takagi, eds.), (Berlin, Heidelberg), pp. 33–48, Springer Berlin Heidelberg, 2011.

[9] L. Goubin and A. Martinelli, "Protecting aes with shamir's secret sharing scheme," in *Cryptographic Hardware and Embedded Systems – CHES 2011* (B. Preneel and T. Takagi, eds.), pp. 79–94, Springer Berlin Heidelberg, 2011.

[10] kokke, "tiny-aes-c (small portable aes128/192/256 in c)." GitHub repository, 2024. Available at: https://github.com/kokke/tiny-AES-c, Accessed: August 14, 2024.

[11] B. Gigerl, V. Hadzic, R. Primas, S. Mangard, and R. Bloem, "Coco:{Co-Design} and {Co-Verification} of masked software implementations on {CPUs}," in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 1469–1468, 2021.

[12] S. Mangard, M. Oswald, and T. Popp, *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007. XXIII, 337 S.

[13] M. Tehranipoor, N. Nalla Anandakumar, and F. Farahmandi, *Power Analysis Attacks on AES*, pp. 137–161. Cham: Springer International Publishing, 2023.

[14] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Cryptographic Hardware and Embedded Systems - CHES 2004* (M. Joye and J.-J. Quisquater, eds.), (Berlin, Heidelberg), pp. 16–29, Springer Berlin Heidelberg, 2004.

[15] K. Miteloudi, A. Adhikary, N. van Drueten, L. Batina, and I. Buhan, "Plan your defense: A comparative analysis of leakage detection methods on RISC-v cores." Cryptology ePrint Archive, Paper 2024/423, 2024. https://eprint.iacr.org/2024/423.

[16] N. Müller, T. Moos, and A. Moradi, "Low-latency hardware masking of prince," in *Constructive Side-Channel Analysis and Secure Design* (S. Bhasin and F. De Santis, eds.), (Cham), pp. 148–167, Springer International Publishing, 2021.

[17] M. Rivain and E. Prouff, "Provably secure higher-order masking of aes," in *Cryptographic Hardware and Embedded Systems, CHES 2010* (S. Mangard and F.-X. Standaert, eds.), (Berlin, Heidelberg), pp. 413–427, Springer Berlin Heidelberg, 2010.

[18] J. Balasch, B. Gierlichs, V. Grosso, O. Reparaz, and F.-X. Standaert, ""on the cost of lazy engineering for masked software implementations"." Cryptology ePrint Archive, Paper 2014/413, 2014. https://eprint.iacr.org/2014/413.

[19] S. Cui and J. Balasch, "Efficient software masking of aes through instruction set extensions," in *2023 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–6, 2023.

[20] S. Gao, J. Großschädl, B. Marshall, D. Page, T. Pham, and F. Regazzoni, "An instruction set extension to support software-based masking," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, p. 283–325, Aug. 2021.

[21] S. Cui and J. Balasch, "Efficient software masking of aes through instruction set extensions," in *2023 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–6, 2023.

[22] P. Davide Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, and L. Benini, "Slow and steady wins the race? a comparison of ultra-low-power risc-v cores for internet-of-things applications," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pp. 1–8, 2017.

[23] C. O'flynn and Z. Chen, "Chipwhisperer: An open-source platform for hardware embedded security research," in *Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers 5*, pp. 243–260, Springer, 2014.

[24] J. van Woudenberg and C. O'Flynn, *The Hardware Hacking Handbook*. No Starch Press, 2021.

[25] H. Gross, D. Schaffenrath, and S. Mangard, "Higher-order side-channel protected implementations of keccak." Cryptology ePrint Archive, Paper 2017/395, 2017. https://eprint.iacr.org/2017/395.

[26] CENSUS, "masked-aes-c (proof-of-concept c implementation of aes with masking technique to prevent side-channel analysis attacks)," 2024. Available at: https://github.com/CENSUS/masked-aes-c, Accessed: August 14, 2024.

[27] NewAE Technology Inc., "Chipwhisperer-husky - newae hardware product documentation," 2024. Available at: https://rtfm.newae.com/Capture/ChipWhisperer-Husky/, Accessed: August 14, 2024.

[28] NewAE Technology Inc., "Chipwhisperer jupyter course." GitHub repository, 2024. Available at: https://tinyurl.com/Cw-CPA-Algo, Accessed: August 14, 2024.