XDIVINSA: eXtended DIVersifying INStruction Agent to Mitigate Power Side-Channel Leakage

Thinh H. Pham[1], Ben Marshall[1], Alexander Fell[2], Siew-Kei Lam and Daniel Page[1]

[1] Department of Computer Science, University of Bristol,
{th.pham,ben.marshall,daniel.page}@bristol.ac.uk
[2] Barcelona Supercomputing Center, Spain,
alexander.fell@bsc.es
[3] Nanyang Technological University, Singapore,
assklam@ntu.edu.sg

Abstract. Side-channel analysis (SCA) attacks pose a major threat to embedded systems due to their ease of accessibility. Realising SCA resilient cryptographic algorithms on embedded systems under tight intrinsic constraints, such as low area cost, limited computational ability, etc., is extremely challenging and often not possible. We propose a seamless and effective approach to realise a generic countermeasure against SCA attacks. *XDIVINSA*, an extended diversifying instruction agent, is introduced to realise the countermeasure at the microarchitecture level based on the combining concept of diversified instruction set extension (ISE) and hardware diversification. *XDIVINSA* is developed as a lightweight co-processor that is tightly coupled with a RISC-V processor. The proposed method can be applied to various algorithms without the need for software developers to undertake substantial design efforts hardening their implementations against SCA. *XDIVINSA* has been implemented on the SASEBO G-III board which hosts a Kintex-7 XC7K160T FPGA device for SCA mitigation evaluation. Experimental results based on non-specific t-statistic tests show that our solution can achieve leakage mitigation on the power side channel of different cryptographic kernels, i.e., Speck, ChaCha20, AES, and RSA with an acceptable performance overhead compared to existing countermeasures.

Keywords: Side-Channel Attack, Hiding, Hardware Diversification, Instruction Set Extension, RISC-V

1 Introduction

Embedded systems have become an integral part of modern lives, and hence it is essential that they are secure. Although cryptosystems implemented on embedded systems have been mathematically proven to be secure, they can be deployed in unforeseen adversarial settings and could be vulnerable to physical attacks that exploit side-channel information, such as execution time, power consumption, electromagnetic emission, etc. In 1999, Kocher et al. presented power side-channel attacks to reveal sensitive information by analysing the power consumption measurements [1]. The power side-channel attacks are typically categorised into two types of attacks: Simple Power Analysis (SPA) and Differential Power Analysis (DPA). SPA can reveal sensitive information by observing the power consumption of one single execution of a cryptographic algorithm. In contrast, DPA extracts sensitive information by statistically analysing a large number of power measurements on the same algorithm with different inputs.

Masking and hiding are two widely used countermeasures against power side-channel attacks. Masking countermeasures merge the sensitive information with random shares which are unknown to the attacker, while hiding countermeasures reduce the signal-to-noise ratio of leakage information in observation traces [2]. Hiding countermeasures has a lower implementation and performance cost compared to masking alternatives, but it takes a toll on security [3]. A common way to perform hiding is to use random delays in embedded software to desynchronise side-channel traces. Hiding is not able to prevent an SCA attack completely, yet renders the attack more complex and time-consuming to the point where it is no longer practical [4]. Moreover, there is generally accepted intuition that side-channel resistance requires the combination of several countermeasures in order to be effective [5]. For example, combination of masking with time randomization is a promising method against SCA attacks.

In this work, we investigate the application of hardware diversification and diversifying instructions to harden cryptographic software on an embedded system against SCA attacks. The approach aims at a countermeasure that is transparent to software developers and requires no substantially algorithm-specific change nor experience to implement the countermeasure. Therefore, the countermeasure is able to integrate straightforwardly into other techniques to constitute a multi-layer countermeasure. The main contributions of this paper are as follows:

1. We propose a novel SCA countermeasure which relies on hardware diversification and hiding techniques. Rather than improving the security of hiding techniques, we focus on delivering a generic and "drop in" countermeasure which avoids substantial changes in both hardware and software, hence, minimise overhead incurred to provide an acceptable security level.

2. An approach combining ISE and diversification concepts is introduced. Diversified ISEs are defined with the same logical function as primitive instructions. This enables the countermeasure to be seamlessly applied to existing cryptographic algorithms to facilitate SCA-hardened realisation without incurring an additional cost in code density and memory footprint, nor re-designing the algorithms.

3. The proposed countermeasure employs a non-invasive approach that avoids negative effects on non-security critical applications and causes only a marginal increase in hardware.

4. An empirical SCA evaluation on widely used cryptographic algorithms is conducted. The measurements show that the proposed method reduces power side-channel leakage below the thresholds and can mitigate actual SCA attacks against AES encryption.

The paper is organized as follows: Section 2 discusses related countermeasures and power analysis attacks, while XDIVINSA is presented in Section 3. In Section 4, we provide a detailed discussion of the experimental evaluation and Section 5 concludes the paper. Moreover, the source code of the proposed method is available at https://github.com/scarv/xdivinsa.

2 Related Work

A hiding countermeasure can be generally realised on time dimension and/or amplitude dimension. The former can improve the robustness of cryptographic algorithms against both power analysis and timing-based attacks. Random delays in software are commonly used in the hiding countermeasure against SCA and fault attacks in embedded devices. There are extensive studies on generating various random delay distributions to increase the difficulty against SCA attacks [3, 4]. Due to the fact that a processor is usually idle during the random delay, these countermeasures are still vulnerable to enhanced SCA attacks, namely Sliding Window DPA [6], and elastic alignment based DPA [7]. The works in [2] showed that runtime code polymorphism [8] is more robust against the enhanced SCA attacks compared to inserting delays based on random dummy loops. The code polymorphism countermeasure uses techniques that involve instruction shuffling, randomly selecting instructions, and inserting noise instructions to generate executed code at runtime. However, runtime code generation and inserting additional redundant instructions incur a large performance overhead in embedded processors.

Hardware solutions of SCA-resistant processors can provide a more generic countermeasure for different cryptographic algorithms. However, existing hardware solutions usually require substantial changes to the processor architecture that impacts all programs running on the processor and also induce performance overhead on non-security critical programs. A hardware-based instruction shuffler was proposed in [12] to shuffle independent instructions randomly for protection against side-channel attacks. However, the effect of this countermeasure is dependent on the implementation of the algorithm software. In addition, a secure processor which can protect against side-channel attacks using masking and hiding techniques was proposed in [13]. Besides an independent data path to implement the masking scheme, a pipeline randomizer is introduced to add non-deterministic dummy control and data signals to the processor data path. The authors in [14] presented an SCA-resistant embedded processor based on masking and DPA-resistant logic styles. These approaches result in a significant increase in hardware usage. Interestingly, a hardware-based non-invasive approach was presented in [11]. This countermeasure avoids substantial changes to a processor architecture and effectively reduces timing side-channel leakage. However, the evaluation in this work is solely performed on execution time measurements. The method is based on naive insertion of a delay after performing the operation. Since the operation is executed at the beginning of execution period, a successful side channel attack could be realized by measuring the power/energy consumption during this time. Hence, the countermeasure is still vulnerable to power side-channel attacks. Table 1 summarises the comparison between countermeasures realised on time dimension. *Invasive* and *non-invasive* hardware-based countermeasures refer to those that either require substantial modifications on a base processor's microarchitecture or not.

3 Proposed Solution

3.1 XDIVINSA

A hardened processor is studied based on a soft RISC-V processor. Basic approaches to integrate a hardware supported countermeasure into a processor can be invasive or non-invasive. The former can achieve low area cost but induces substantial changes in hardware and depends on a specific microarchitecture. The latter can be implemented in isolated hardware modules (i.e., dedicated IP modules) which can be micro-architecture independent but induce higher area cost and longer latency to access data compared to the former. Our proposed method can be viewed as a non-invasive alternative to provide a drop-in solution which is independent of the soft processor implementation. We introduce XDIVINSA as a co-processor which is tightly coupled with the soft processor as shown in Fig. 1. There is no additional changes to the processor microarchitecture except for the need of a co-processor interface which is assumed to be available in the soft processor. The tightly coupling with the processor enables XDIVINSA to effectively get the source registers, stall the entire processor, and write back the result to the destination register through the co-processor interface. Importantly, the presence of XDIVINSA does not cause negative effects on the soft processor micro-architectural performance (i.e., reducing maximum clock rate). Therefore, the countermeasure does not affect other normal (non-security critical) applications running on the processor.

An ISE executed by XDIVINSA, called the diversified ISE, is introduced to diversify a subset

Table 1: Comparison of countermeasures on time dimension.

Methods	Side Channel	Software based	Hardware based	
			invasive	non-invasive
[3, 4]	power	dummy loops		
[8]	power	shuffling, noise instr.		
[9, 10]	power	enhancing algorithms		
[11]	time			✓
[12, 13]	power		✓	
This work	power			✓

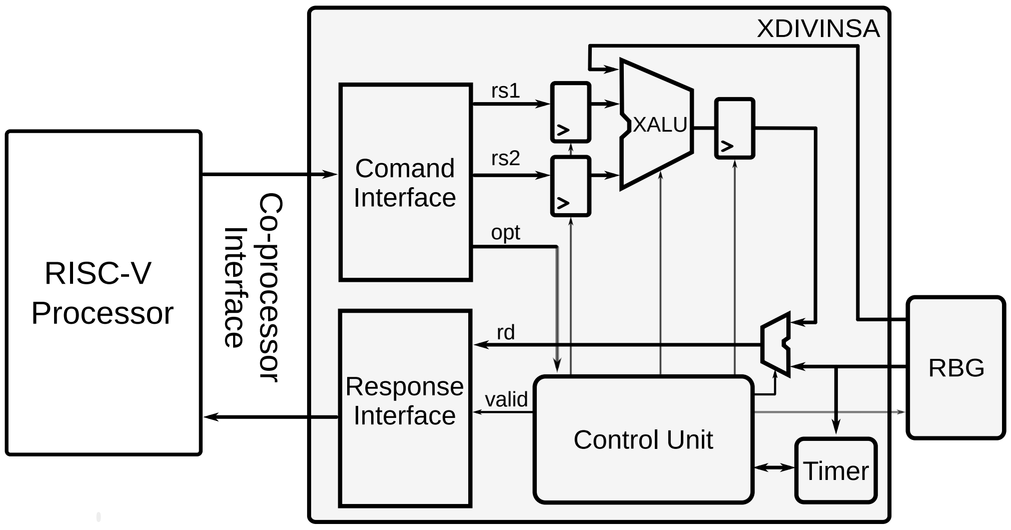

Figure 1: Hardware Diversification on XDIVINSA.

of the arithmetic/logic instructions. We opt for the Addition and Xor instructions because these instructions are commonly used in integer-number-based (e.g., RSA) or finite-field-based (e.g., AES) cryptography, respectively, to handle critical data. It is worth noting that the RSA cryptosystem works on large integer number computation that requires both Addition and Multiplication instructions. We select diversifying the Addition instruction in favour of low area cost. The diversification of executing ISE can typically be implemented in hardware by using multiple versions of the same operation. However, this approach incurs a considerable hardware consumption to implement multiple versions of the operation. Instead, XDIVINSA relies on hiding technique mechanisms to implement diversification on the time dimension. That can reduce the leakage with lower cost compared to the previous alternative. Indeed, only one execution unit is implemented inside the Extended ALU (*XALU*) for each ISE instruction. Every time the instruction is executed, a different delay duration is inserted before and after instruction operation. With this, the measurements of power traces are desynchronised, therefore reducing the leakage observed. In addition, the hardware could also perform other countermeasures, i.e., inserting dummy operations. In this countermeasure, XDIVINSA executes dummy operations on 32-bit random operands during the delay durations resulting in different power profiles of the ISE. Doing so helps to raise the difficulty for an attack templating the diversified ISEs.

To enable different execution times of the ISE, the *Control Unit* loads random bits into a countdown timer to generate the random duration. A larger range for the random duration results in a better de-synchronisation (i.e., more security). However, this also culminates in a longer overall execution time. We define a diversification level L to specify the number of diversifying profiles in time ($n = 2^L$) wherein the execution duration are varied in the range from 1 to n clock cycles. Therefore, the level is equivalent to the number of random bits required to represent the random duration. As our countermeasure requires a relatively small L to achieve the expected leakage mitigation, we empirically choose to support the diversification levels upto 8 that equivalently requires 8 random bits. The *opt* signal determines the operation to be executed on *XALU* and also allows a developer to set an appropriate diversification level as a trade-off between channel leakage and execution time. During the random duration, dummy operations are calculated on 32-bit random numbers. In the middle of the random duration, the *Control Unit* releases the actual operands (*rs1, rs2*) from the registers, and latches the result. After the random duration expiration, the *valid* signal is set to indicate the completion of the operation on XDIVINSA and the valid result is passed to the processor core (refer to Fig. 1). The core is then allowed to continue its execution.

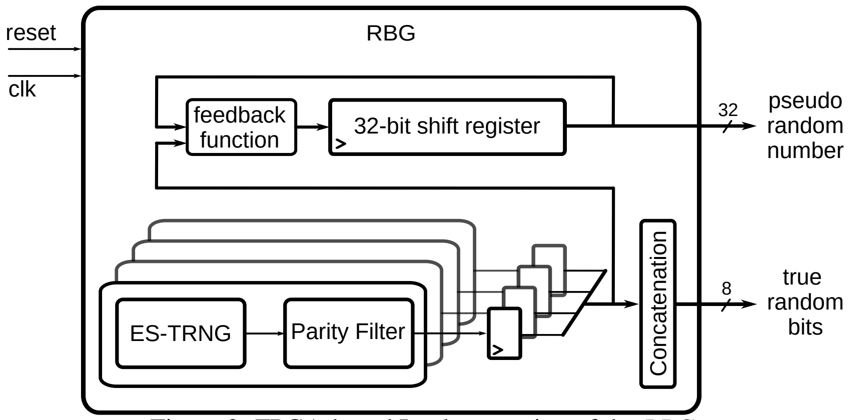

Figure 2: FPGA-based Implementation of the *RBG*.

3.2 Random Bit Generation (RBG)

There are two basic types of random number generators, Pseudo Random Number Generator (PRNG) and True Random Number Generator (TRNG). The former generates random number sequences using a deterministic algorithm according to initial numbers, called seeds. The randomness of the seeds can be governed by a non-deterministic input to obtain the unpredictability of PRNG. The latter has an indeterministic property, required by cryptographic applications, and is based on physical sources such as thermal noise. It normally induces a higher area cost and longer latency for generating large number of random bits compared to PRNG. Practically, in some current embedded processor systems, a TRNG and/or PRNG in available. In this case, our XDIVINSA can reuse these built-in random sources from outside through some inputs for its RBG. Since the basic RISC-V processor lacks random sources, we need to implement an RBG for XDIVINSA.

The RBG generates 8 random bits to define the random execution time and 32-bit random operands for dummy operations. The random duration of the execution is critical for hiding techniques. Hence, high-quality randomness is needed. Moreover, the generation of random duration requires relatively few random bits (i.e., 8 bits) and low throughput. Therefore, we implement a true random generator for the 8 random bits. The dummy operations require 32-bit random numbers with high throughput (i.e., one number per clock cycle) during the random execution time. Therefore, a PRNG is implemented for this function to focus on area overhead and throughput over the randomness quality. Fig. 2 shows the implementation of our RBG.

There are many TRNG designs for FPGA platforms in the literature which either use timing jitter [15, 16] or metastability [17] as a noise source. We implemented our true RBG (TRBG) based on ES-TRNG [15] which relies on timing jitter of ring oscillators implemented on the FPGA fabric. ES-TRNG is designed to generate one single raw random bit. To enhance statistical and security characteristics of the ES-TRNG, a third-order parity filter is used for post-processing the raw random bit. This implementation is certified to pass the NIST SP 800-22 statistical test suites [18] and is compliant with the AIS-31 standard [19]. To produce multiple random bits, apart from generating each bit sequentially, an alternative is to generate them in parallel. This approach results in a considerable area cost, but has an increased throughput and reduces the correlation between the bits. Our TRBG is practically implemented with four 1-bit ES-TRNGs in parallel and one shift register. The shift register sequentially concatenates 4 random bits to generate 8 random bits.

For the PRNG, a simple 32-bit linear feedback shift register (LFSR) is used. The LFSR can generate the maximum length sequence of $2^{32} - 1$ numbers. The selection of a feedback function determines the maximum length pseudo random sequence and the appropriate function described in [20] is used. The LFSR is updated every clock cycle, and the state cannot be accessed by the soft processor. Although the PRNG could produce uniform random numbers, the output is deterministic and can be exploited by attackers. An input from our true TRBG seeds non-deterministic values to the PRNG to enhance its security.

3.3 ISE

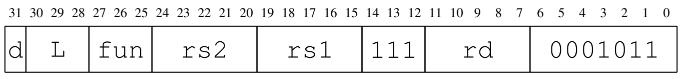

Figure 3: Encoding of a custom RISC-V instruction for the diversified ISE.

We realise the diversified ISE using the *custom-0* encoding space [21], which is reserved for custom instructions. Specifically, the ISE is encoded per Fig. 3. The d bit is set to specify the diversified ISE. L and *fun* (3 bits for each) determine the diversification level and operation, respectively, for the ISE. *fun* can be assigned to 0, 1, and 2 to perform diversified Addition, diversified Xor and RBG reading, respectively. The other values of *fun* are reserved. 5-bit *rd*, *r1* and *r2* are encoded for the destination, the first and the second source registers, respectively, in the processor's register file. Three bits, 12, 13, 14, are set to enable the processor to fetch two source registers and write the computed result back to the destination register.

3.4 Instruction Substitution

Since the function of the diversified ISE is equivalent to the function of the corresponding normal instruction, the presence of the ISE can be abstracted from algorithm implementation at software level. Realising an SCA countermeasure for a cryptographic algorithm using the proposed method can be straightforwardly done in an automatic manner through an instruction substitution sequence as follows: Firstly, a cryptographic implementation is compiled into assembly codes. Then, a script (i.e., python) is employed to substitute the normal instructions in the assembly codes by the corresponding diversified ISE. Finally, the revised codes are compiled and linked to an executable file by using a customised binary tool (i.e., GNU Binutils). The customised tool is modified from RISC-V GNU Binutils to support the diversified ISE. This approach avoids incurring any cost in software, and relieve developers from the need to implement cryptographic algorithms in a new language or with side-channel security considerations. Currently, the substitution uses a naive find-and-replace mechanism. In future work, the substitution will be investigated with other objective functions (e.g., to reduce execution time and to increase obfuscation level).

3.5 Implementation on FPGA

We employ Xilinx Vivado 2019.1 to implement the evaluated systems on Kintex-7 XC7K160T FPGA device; default synthesis settings are used, with no effort invested in synthesis or post-implementation optimisation. In this work, we investigate the integration of XDIVINSA for two different and popular RISC-V processor core's implementations: a) Rocket Chip for a high-performance profile and b) PicoRV32 for a low-cost profile. Both of them explicitly support a similar co-processor interface. Table 2 reports the hardware resource utilization of XDIVINSA and RBG compared to the typical Rocket Chip and PicoRV processor. The RBG uses 96 slices, consisting of 15 slides for PRNG and 81 slides for TRBG, which are more than double of that consumed by XDIVINSA. The implementation of XDIVINSA consumes only 39 slices. The slices used by both XDIVINSA and RBG occupy only 3.9% of the total slices of the entire Rocket Chip system. For the low-area profile system, the number of slices used by XDIVINSA and RBG accounts for 16.7% of the total slices of the PicoRV system.

Table 2: Hardware Resource Utilization.

	XDIVINSA	PRNG	TRBG	Rocket Chip	PicoRV
Slices	39	15	81	3486	809
DSPs	0	0	0	8	0

4 Experimental Evaluation

4.1 Experimental Setup

The proposed solution has been implemented and evaluated on SASEBO G-III (i.e., SAKURA-X) board [22]. The setup consists of four main components, i.e., the host computer, SAKURA-X board, amplifier, and PicoScope. The host computer interfaces to the SAKURA-X board via a USB-UART connection to provide data input and commands to the system on the board being evaluated. The secured soft processor is implemented on the cryptographic FPGA device of SAKURA-X. The board has a connector to a passive probe for measuring the consumed power of the FPGA device. The power signals probed from the SAKURA-X board is fed to an amplifier to enhance the SNR of measured signals before being passed as inputs to an oscilloscope. The oscilloscope is set to capture power traces at the sampling rate of 250 MHz. The acquired power traces are sent to the host computer for leakage analysis.

The executions of the PicoRV32 processor has lower switching noise and, therefore, generates clearer leakage in power traces (due to its simple architecture and multiple cycle execution) compared to the Rocket Chip (a 5-stages pipeline processor). Hence, the PicoRV32 processor is chosen as the worse case for leakage evaluation. The system runs the benchmarks on the secured 32-bit PicoRV processor core with a system clock of 50 MHz.

4.2 Benchmark Functions

Power side-channel leakage mitigation is evaluated with a set of cryptographic kernels. Our benchmark includes widely used cryptographic kernels as follows:

1. Speck is an (Addition-Rotation-Xor) ARX-based family of lightweight block ciphers [23]. Speck-64/128, referring to the variant of Speck characterized by a 64-bit block, a 128-bit key, and 27 rounds, is investigated in our evaluation. The round function of Speck-64/128 uses only bitwise Xor, modular addition, and rotations. The Speck encryption function is included in our benchmarks.

2. ChaCha20, an ARX based stream cipher, is deployed in many application domains [24]. This kind of cipher is easy to protect against timing side-channel attacks. However, other side-channel protections, i.e., power or EM are very costly. The Chacha20 block function is included in our benchmarks.

3. The Advanced Encryption Standard (AES), original name Rijndael, was standardised by the National Institute of Standards and Technology in 2001 [25]. AES-128, referring to a 128-bit key variant, encryption function is implemented using pre-computed S-box for our benchmarks.

4. Modular exponentiation (*modExp*) function used in RSA cryptosystem to encrypt or decrypt a message [26] is also included in the benchmarks. The *modExp* function is straightforwardly implemented using the square-multiply algorithm with long integer numbers. We apply the Coarsely Integrated Operand Scanning (CIOS) Montgomery algorithm [27] (i.e., Montgomery reduction and Montgomery multiplication) for *modExp* calculation.

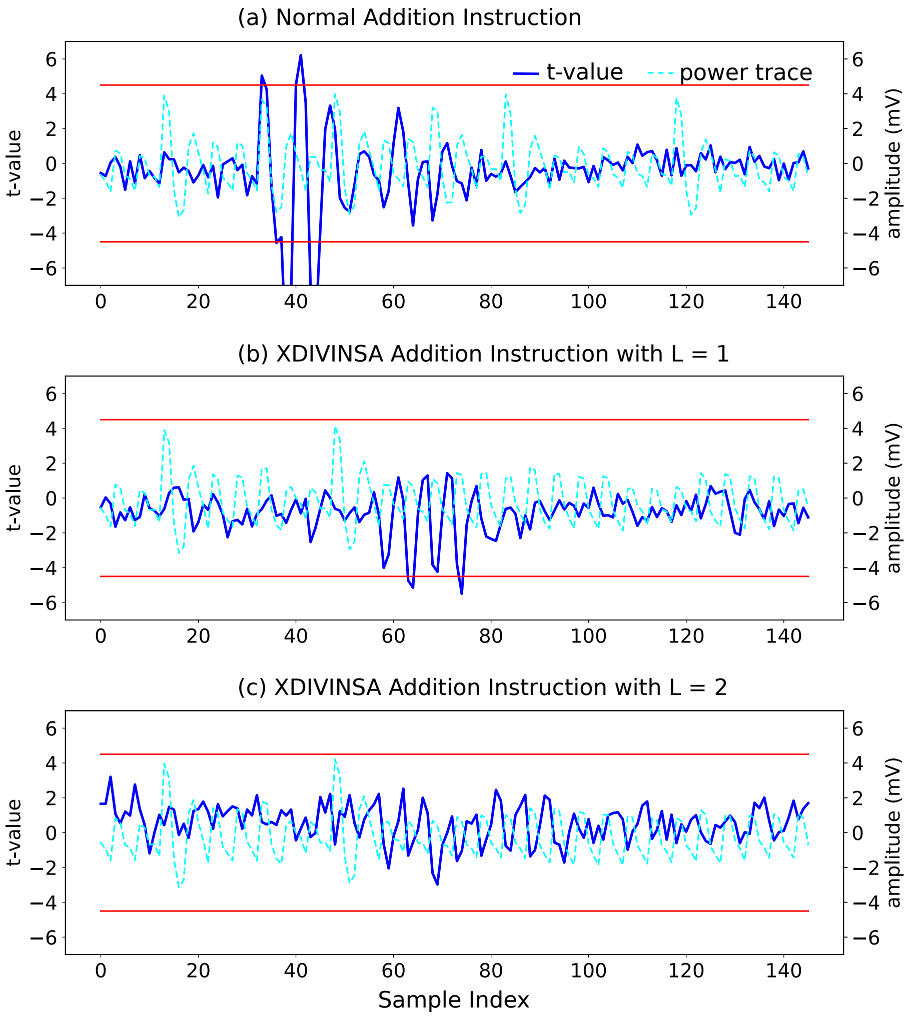

Figure 4: Non-specific leakage detection test on Addition instructions.

4.3 Evaluation Results

In order to evaluate and verify leakage mitigation, a range of methodologies exist: Side-channel Vulnerability Factor [28], Signal Available to Attacker [29], and the Welch's t-test [30] based Test Vector Leakage Assessment [31], can, for example, be used to assess whether leakage occurs in the power traces. A main advantage of such approaches is their non-specific nature, which helps to abstract the evaluation of leakage mitigation countermeasures from performing a battery of side-channel attacks. We adopt fixed versus random t-tests for evaluating the leakage. The two sets of measurements are employed to calculate t-test statistics. If the t-test values are greater than the threshold $T = \pm 4.5$, the null hypothesis "the implementation has no leakage" is rejected with the confidence $< 99.999\%$. Otherwise, it corroborates that the information leakage on power side channel of the measurements is not distinguishable. The leakage evaluations are performed with two detail levels: a) a fine level shows the leakage detection on the power traces of a diversified instruction. At this level, it can be clearly seen how the countermeasure reduces the leakage. b) a coarse level illustrates the observed leakage of each benchmark function. It provides a leakage detection on very long traces to evaluate the leakage mitigation results of the countermeasure compared to the unprotected versions.

Fig. 4 and Fig. 5 show the leakage detection of the Addition and Xor instructions, respectively.

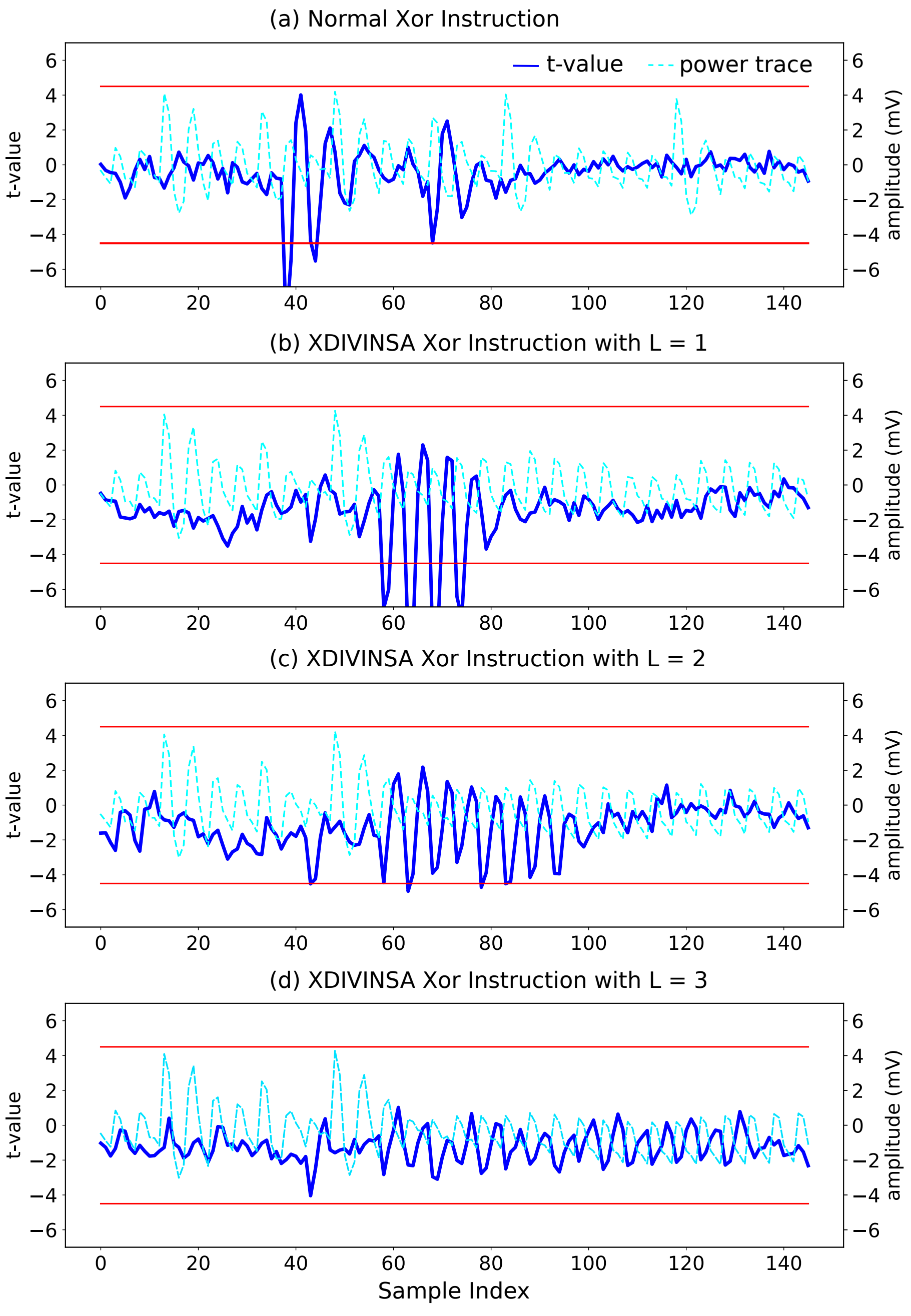

Figure 5: Non-specific leakage detection test on Xor instructions.

The figures illustrate the power trace average and the t-test results of normal instructions and diversified ISEs with different levels L running on the evaluated system with a PicoRV core. Two red horizontal lines on the t-value sub-figures denote the threshold $T = \pm 4.5$. If the t-test values are below the threshold, it corroborates that the information leakage on the power traces is not distinguishable. NOP instructions are inserted before and after the evaluated instructions to isolate their leakage. We collect 100,000 power traces for each t-test evaluation. As can be observed, the normal instruction clearly expose leakage on t-value traces. Interestingly, the effect of diversifying execution in time before performing instructions' operation in the XDIVINSA flattens the amplitude peaks in the average power trace, equivalently spreads and reduces the leakage in t-value traces. Larger spreading diversification (i.e., larger L) makes the leakage smaller and spreading wider. The leakage of the diversified Addition and Xor instructions of which L is set greater than 2 and 3, respectively, is below the threshold.

Fig. 6 shows the results of the proposed countermeasure applied to the cryptographic kernels compared to their unprotected implementation. Since collecting very long traces (i.e., greater than 17,000, 60,000, 70,000, and 170,000 samples per trace for Speck, Chacha20, AES, and RSA, respectively) leads to a large memory required and longer acquisition and processing time, the number of traces collected for each t-test evaluation is set to 10,000 traces. In each benchmark, the plot above shows the t- test trace of the unprotected version, while the plot below represents the protected version with XDIVINSA. The leakage can be clearly seen in the unprotected t-test traces. The leakage of the protected versions is reduced below the threshold at different L levels (i.e., 3, 3, 3, and 4 for Speck, Chacha20, AES, and RSA, respectively).

4.4 Leakage Recovery and Power Side-Channel Attacks

We evaluate further the security of the countermeasure against actual attacks in terms of the number of traces required to succeed. Without loss of generality, the AES encryption case study is chosen for the evaluation. Firstly, a well-known first order correlation-based differential power analysis attack (CPA), used in [12, 8], is performed against both unprotected and protected implementations. The Hamming weight is used to estimate the power consumption model. The attack computes the sample estimation of PearsonâĂŹs correlation coefficients between the measured power traces and the model for each possible hypothetical value of the involved key part. The hypothetical value which results in the maximal correlation value is guessed as the key. If the guessed key matches the correct key, the attack is successful. We choose to target the first-round substitution operation of the first key-byte because in the countermeasure, this operation has the least protection compared to that of the subsequence substitutions. Only one diversified ISE is performed before this operations while an increased number of diversified ISE are executed through the iteration of AES rounds providing more protection.

Fig. 7.a illustrates the result of the CPA attack on the unprotected implementation. The correct key clearly distinguishes from all the other hypothetical values as soon as more than 4,000 traces are used for the attack. This result validates the experimental setup and the choice of the Hamming weight model targeted operation used in the CPA. Fig. 7.b and 7.c show the results of the CPA attack on the protected implementation using XDIVINSA with L = 1, and 3, respectively. Even if XDIVINSA is set with the smallest diversification level, the countermeasure can mitigate the attack. In case of $L = 3$, where the leakage is compressed below the threshold, the correlation value of the correct key is clearly indistinguishable from the values of other keys.

Lastly, we have conducted well-known trace alignment attacks, i.e., sliding window integration (SWI) [6] and elastic alignment (EA) [7] to recover the leakage hiding by the countermeasure before performing the CPA. Fig. 8.b and 8.c show the results of the combined attack using SWI and EA, respectively, on the protected implementation. The sub-figures on the left-hand side present the result of the countermeasure with $L = 1$. It can be seen that the SWI attack recovers the leakage and results in a successful attack with more than 16,000 traces required. The EA attack fails to have the similar result of the SWI attack. More importantly, when the implementation is protected with $L = 3$ shown in the right-hand side sub-figures, both of these attacks are unsuccessful.

Table 3: Hardware Cost Comparison.

	LUTs	FFs
Hardware Shuffler [12]	278	131
XDIVINSA	121	76

Table 4: Performance Cost Comparison.

	Speck	ChaCha	AES
Hardware Shuffler [12]	-	-	$\approx 1\times$
Code Polymorphism [8]	-	-	$2.3\times$
PicoRV-XDIVINSA	$1.1\times$	$1.2\times$	$1.1\times$

4.5 Comparison with existing countermeasures

This sub-section provides the comparison between the proposed method and existing hiding based countermeasures in terms of area and performance overheads. We considered the 16-blocks hardware shuffler presented in [32] and the runtime code polymorphism countermeasure in [8] for comparisons. Table 3 reports the hardware usage of XDIVINSA and the hardware shuffler. The results listed exclude the overhead of RBG in both implementations. XDIVINSA consumes about half of the Look-up Tables (LUTs) and Flip-Flops (FFs) compared to the hardware shuffler.

Table 4 reports the performance overhead of the countermeasures. The overhead is represented by a ratio between the execution time of the protected and unprotected implementations. For the proposed countermeasure, diversification levels are assigned as in Section 4.3 so that the security of the countermeasure is satisfied while measuring its performance. The hardware shuffler causes a negligible performance overhead to protect AES encryptions, while the countermeasure using XDIVINSA introduces an overhead of $1.1\times$. The runtime code polymorphism in software has an overhead of $2.3\times$ for the AES. In addition, the proposed countermeasure also induces small performance overheads of $1.1\times$ and $1.2\times$ on Speck and Chacha20, respectively. Importantly, the proposed countermeasure does not induce any overheads of code density and memory footprint, and avoids requiring changes in software to realise the countermeasure. None of the existing countermeasures offers these advantages.

5 Conclusion

We have proposed XDIVINSA to harden cryptographic software against SCA attacks. The proposed solution employs diversified ISE with hardware diversification to provide a hardware-based countermeasure avoiding modification in software. This enable the software to benefit from the countermeasures automatically and without user intervention. Experimental results from the benchmarks consisting of widely used ciphers proved that our solution achieves leakage mitigation on the power side-channel. The proposed hardware incurs a negligible increased area cost compared to the base processor. The protected software using our method does not suffer any increase in terms of code density and memory-footprint but induces a small performance overhead compared to its unprotected version. We show that the proposed solution is generic and can be applied to various cryptographic algorithms without requiring any changes to them.

Acknowledgement

This work has been supported in part by EPSRC via grant EP/R012288/1, under the RISE (http://www.ukrise.org) programme.

References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology — CRYPTO' 99*, M. Wiener, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[2] K. M. Abdellatif, D. Couroussé, O. Potin, and P. Jaillon, "Filtering-based CPA: A Successful Side-channel Attack Against Desynchronization Countermeasures," in *Proceedings of the Fourth Workshop on Cryptography and Security in Computing Systems*, ser. CS2 '17. New York, NY, USA: ACM, 2017, pp. 29–32.

[3] J.-S. Coron and I. Kizhvatov, "Analysis and Improvement of the Random Delay Countermeasure of CHES 2009," in *Cryptographic Hardware and Embedded Systems, CHES 2010*, S. Mangard and F.-X. Standaert, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 95–109.

[4] M. Tunstall and O. Benoit, "Efficient Use of Random Delays in Embedded Software," in *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, D. Sauveron, K. Markantonakis, A. Bilas, and J.-J. Quisquater, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 27–38.

[5] F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard, "The World Is Not Enough: Another Look on Second-Order DPA," in *Advances in Cryptology - ASIACRYPT 2010*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 112–129.

[6] C. Clavier, J.-S. Coron, and N. Dabbous, "Differential Power Analysis in the Presence of Hardware Countermeasures," in *Cryptographic Hardware and Embedded Systems — CHES 2000*, Ç. K. Koç and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 252–263.

[7] J. G. J. van Woudenberg, M. F. Witteman, and B. Bakker, "Improving Differential Power Analysis by Elastic Alignment," in *Topics in Cryptology – CT-RSA 2011*, A. Kiayias, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 104–119.

[8] D. Couroussé, T. Barry, B. Robisson, P. Jaillon, O. Potin, and J.-L. Lanet, "Runtime Code Polymorphism as a Protection Against Side Channel Attacks," in *Information Security Theory and Practice*. Cham: Springer International Publishing, 2016, pp. 136–152.

[9] Á. Kiss, J. Krämer, P. Rauzy, and J.-P. Seifert, "Algorithmic Countermeasures Against Fault Attacks and Power Analysis for RSA-CRT," in *Constructive Side-Channel Analysis and Secure Design*, F.-X. Standaert and E. Oswald, Eds. Springer International Publishing, 2016, pp. 111–129.

[10] H. Kim, Y. Choi, D. Choi, and J. Ha, "A secure exponentiation algorithm resistant to a combined attack on rsa implementation," *International Journal of Computer Mathematics*, vol. 93, no. 2, pp. 258–272, 2016.

[11] T. H. Pham, A. Fell, A. K. Biswas, S. Lam, and N. Veeranna, "CIDPro: Custom Instructions for Dynamic Program Diversification," in *28th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2018, pp. 224–229.

[12] A. G. Bayrak, N. Velickovic, P. Ienne, and W. Burleson, "An architecture-independent instruction shuffler to protect against side-channel attacks," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 20:1–20:19, Jan. 2012.

[13] F. Bruguier, P. Benoit, L. Torres, L. Barthe, M. Bourree, and V. Lomne, "Cost-effective design strategies for securing embedded processors," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 1, pp. 60–72, Jan 2016.

[14] S. Tillich, M. Kirschbaum, and A. Szekely, "SCA-resistant Embedded Processors: The Next Generation," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: ACM, 2010, pp. 211–220.

[15] B. Yang, V. RoÅ¿ic, M. Grujic, N. Mentens, and I. Verbauwhede, "ES-TRNG: A High-throughput, Low-area True Random Number Generator based on Edge Sampling," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 3, pp. 267–292, 2018.

[16] V. Rozic, B. Yang, W. Dehaene, and I. Verbauwhede, "Highly efficient entropy extraction for true random number generators on FPGAs," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[17] I. Vasyltsov, E. Hambardzumyan, Y.-S. Kim, and B. Karpinskyy, "Fast Digital TRNG Based on Metastable Ring Oscillator," in *Cryptographic Hardware and Embedded Systems – CHES 2008*, E. Oswald and P. Rohatgi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 164–180.

[18] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, "NIST special publication 800-22 revision 1a," National Institute of Standards and Technology, 2010.

[19] W. Killmann and W. Schindler, "A Proposal for: Functionality classes for random number generators," BSI, Bonn, 2011.

[20] M. George and P. Alfke, "Linear Feedback Shift Registers in Virtex Devices." [Online]. Available: www.xilinx.com/support/documentation/application_notes/xapp210.pdf

[21] "The RISC-V instruction set manual," Tech. Rep. Volume I: User-Level ISA (Version 20190608-Base-Ratified), 2019. [Online]. Available: http://riscv.org/specifications/

[22] SAKURA-X, 2013. [Online]. Available: http://www.risec.aist.go.jp/project/sasebo/download/SASEBO-GIII_Spec_v1_1_English.pdf

[23] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers, "The simon and speck lightweight block ciphers." IEEE, 2015, pp. 1–6.

[24] D. J. Bernstein, "Chacha, a variant of salsa20," in *Workshop Record of SASC: The State of the Art of Stream Ciphers*, 2008.

[25] NIST, "Announcing the ADVANCED ENCRYPTION STANDARD (AES)," ed-eral Information Processing Standards Publication, n. 197, Nov. 26, 2001.

[26] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[27] C. Kaya Koc, T. Acar, and B. S. Kaliski, "Analyzing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33, June 1996.

[28] J. Demme, R. Martin, A. Waksman, and S. Sethumadhavan, "Side-channel vulnerability factor: a metric for measuring information leakage," in *International Symposium on Computer Architecture (ISCA)*, 2012, pp. 106–117.

[29] R. Callan, A. Zajić, and M. Prvulovic, "A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014, pp. 242–254.

[30] B. Welch, "The generalization of "student's" problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1-2, pp. 28–35, 1947.

[31] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," in *NIST non-invasive attack testing workshop*, vol. 7, 2011, pp. 115–136.

[32] W. Diehl, A. Abdulgadir, J.-P. Kaps, and K. Gaj, "Side-channel resistant soft core processor for lightweight block ciphers." IEEE, 2017, pp. 1–8.

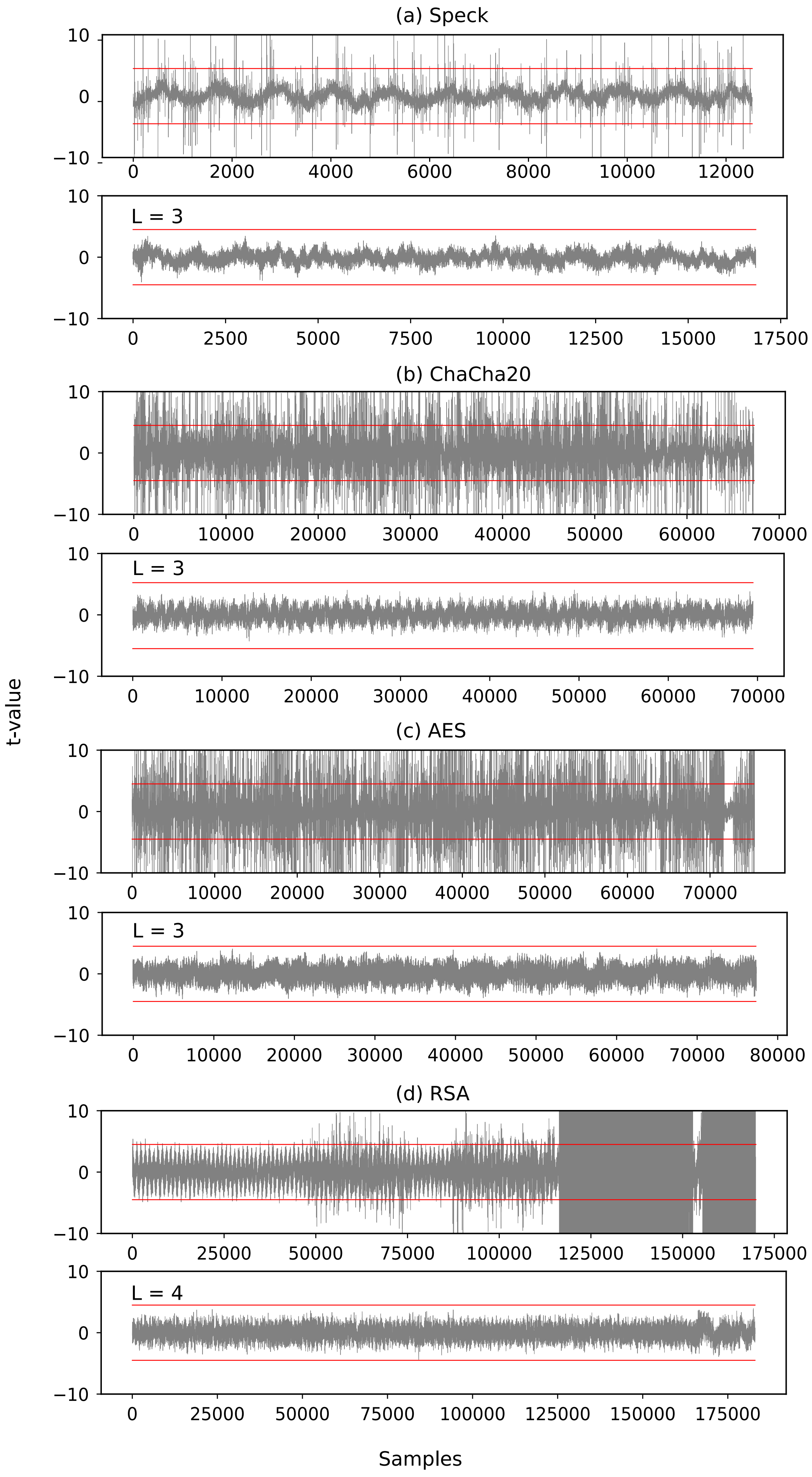

Figure 6: Leakage detection test on the unprotected version and protected version applied *XDI-VINSA* of the cryptographic kernels.

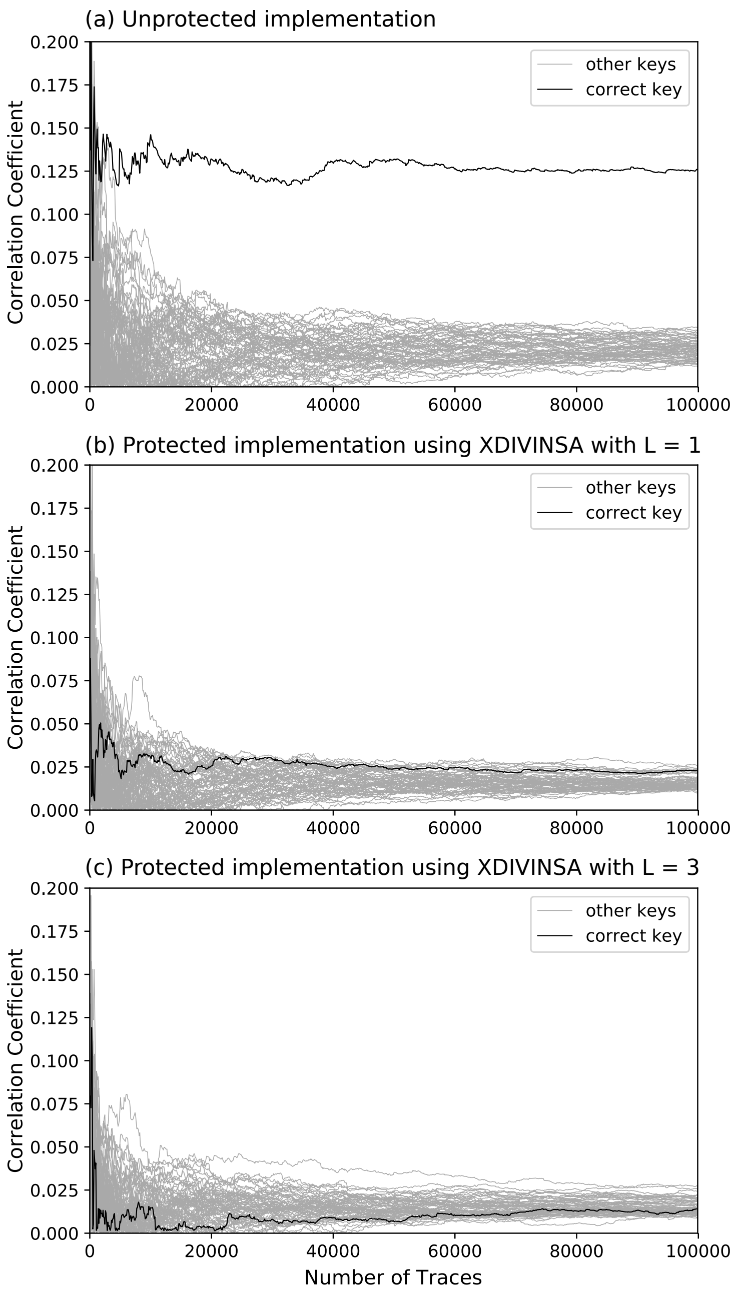

Figure 7: Correlation-based differential power analysis attack against both unprotected and protected implementations.

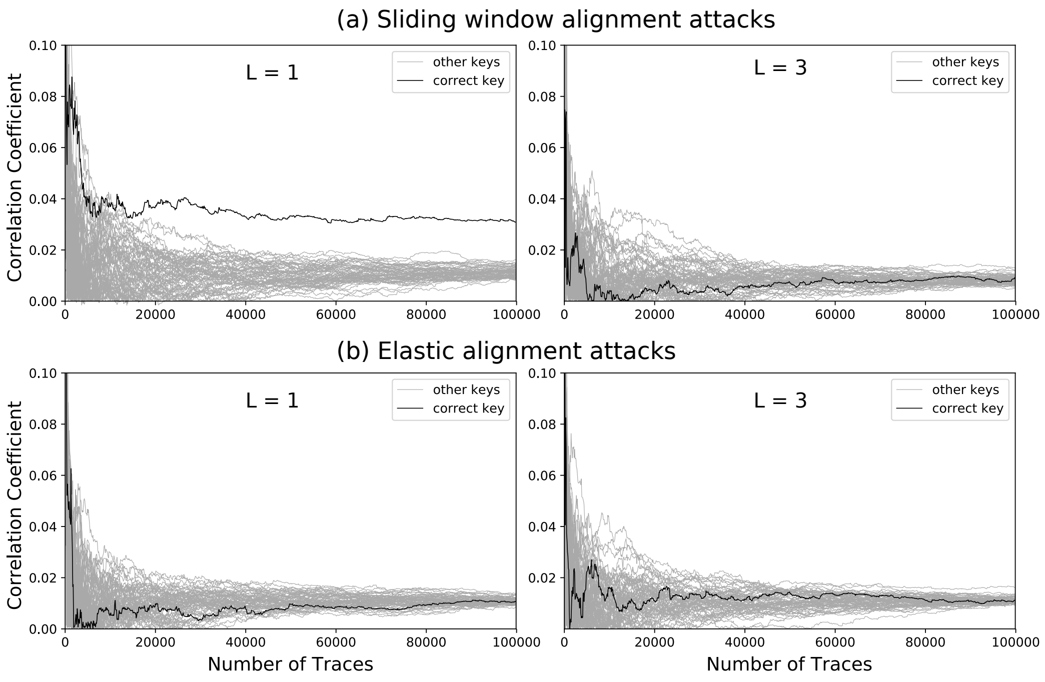

Figure 8: Combining Leakage Recovery and CPA attacks against the protected implementations.