Extending and Accelerating Inner Product Masking with Fault Detection via Instruction Set Extension

Songqiao Cui
ESAT-COSIC
KU Leuven
Leuven, BE
songqiao.cui@esat.kuleuven.be

Geng Luo[*]
School of Computing
National University of Singapore
Singapore, SG
luo.geng@comp.nus.edu.sg

Junhan Bao[*]
Independent Researcher
CN
postbjh@126.com

Josep Balasch[*]
Rambus
NL
jbalasch@rambus.com

Ingrid Verbauwhede
ESAT-COSIC
KU Leuven
Leuven, BE
ingrid.verbauwhede@esat.kuleuven.be

***Abstract*—Inner product masking is a well-studied masking countermeasure against side-channel attacks. IPM-FD further extends the IPM scheme with fault detection capabilities. However, implementing IPM-FD in software especially on embedded devices results in high computational overhead. Therefore, in this work we perform a detailed analysis of all building blocks for IPM-FD scheme and propose a Masked Processing Unit to accelerate all operations, for example multiplication and IPM-FD specific Homogenization. We can then offload these computational extensive operations with dedicated hardware support. With only 4.05% and 4.01% increase in Look-Up Tables and Flip-Flops (Random Number Generator excluded), respectively, compared with baseline `cv32e40p` RISC-V core, we can achieve up to $16.55\times$ speed-up factor with optimal configuration. We then practically evaluate the side-channel security via uni- and bivariate Test Vector Leakage Assessment which exhibits no leakage. Finally, we use two different methods to simulate the injected fault and confirm the fault detection capability of up to $k-1$ faults, with k being the replication factor.**

***Index Terms*—Hardware security, side-channel attack, fault-injection attack, RISC-V, ISE**

I. Introduction

Since the rapid development and deployment of ubiquitous embedded devices, hardware attacks have become a significant threat to the security of these systems. Hardware attacks can be typically categorized into two categories, namely side-channel attacks and fault injection attacks [1]. Side-channel attacks (SCA) are passive, and exploit the fact that sensitive information can be leaked through physical and measurable channels, such as power consumption [2], [3], electromagnetic radiation [4], or timing information [5]. Fault injection attacks (FIA), on the other hand, are active and introduce faults, including voltage or clock glitches, laser injection, and electromagnetic interference into the system. By analyzing the system behavior to the injected faults, the adversaries can recover secret information or bypass security functionalities [6], [7]. It is also shown in literature that side-channel and fault analysis can be combined to perform more powerful attacks [8].

Therefore, hardware security has become a critical area of research and development, with a focus on designing secure and efficient countermeasures against these threats. One of the most widely used and studied countermeasures against SCAs is masking [9]. Contrary to processing the sensitive data directly, masking techniques encode the sensitive data s into random shares. These shares are then processed independently, therefore reducing the side-channel leakage of the original sensitive data. Many masking schemes have been proposed, including Boolean Masking [9], Arithmetic Masking [10], and Inner Product Masking [11], [12], and Direct Sum Masking [13]. Different masking schemes involve different encoding functions and thus consequently different mathematical operations over the shares and security levels. Boolean masking (BM), for example, encoding the sensitive data with Boolean XOR functions, is the most simple and extensively studied masking scheme. Inner product masking (IPM), on the other hand, encodes the sensitive data as the more complex inner product of a public left vector and a secret share vector, providing stronger security guarantees [11].

On the other hand, countermeasures against FIAs can be roughly categorized into two main types, namely prevention and detection [6]. The first type typically involves physical design modifications, such as frequency or voltage regulator to detect anomaly and active shields to prevent laser injection, making it harder for the adversaries to inject faults into the system. To detect the injected faults, countermeasures usually involve redundancy, such as temporal, spatial, or information redundancy [6]. The redundancy can be used to detect discrepancies between system results that may indicate the presence of injected faults and make arbitration decisions to either reset the system or accept the system output based on the comparison results [14].

Direct Sum Masking (DSM) allows for inherent concurrent side-channel and fault injection resistance through redundancy check [13]. Cheng, et al. compare IPM with DSM and extend IPM with fault detection (IPM-FD) capabilities in [15]. Later,

[*] Work completed while affiliated with KU Leuven.

IPM-RED was proposed to extend IPM with robust error detection to also enable concurrent side-channel and fault injection resistance [16].

These countermeasures, when implemented in software, usually involve significant overhead in terms of time and memory usage, which are usually constrained in embedded devices. Earlier works have proposed hardware extensions to offload computational extensive software operations to dedicated hardware circuits. This makes RISC-V a good target base for being open-source and allowing hardware customization. Domain-Oriented Masking, a hardware-oriented approach to prevent glitch-based leakages [17] was integrated into a RISC-V core and create non-linear instructions [18]. Gao et al. [19] propose instructions targeted at first-order BM operations, e.g. BM multiplication and squaring. Lozachmeur and Tisserand propose similar instructions to perform masked operations at higher orders [20]. Cui and Balasch proposed Galois Field multiplier to target and accelerate the lowest level bottleneck and can be applied to different masking schemes [21]. Cheng et al. [22] further categorized masked instructions into computation-based and storage-based groups, emphasizing their impact on leakage reduction.

Few prior works focused on ISE proposals target the IPM-FD scheme, which provides simultaneous protection against both SCA and FIA. In particular, IPM-FD relies heavily on IPM building blocks but also requires unique operations such as Homogenization which cannot be efficiently expressed with existing ISEs. Thus our work fills the gap and is the first to focus on designing a dedicated Masked Processing Unit and associated RISC-V ISEs that can be beneficial to both IPM and IPM-FD schemes. Our contributions are summarized as below:

- We propose a Masked Processing Unit (MPU) that natively implements all algorithms from IPM-FD building blocks, including IPM-FD specific homogenization algorithm.
- We propose the first RISC-V ISEs that provide both Side-Channel and Fault Injection resistance and demonstrate the extension unit with `cv32e40p` core and synthesize it on a FPGA and analyze their area costs.
- We benchmark IPM-FD with different configurations and evaluate the performance gain of our extension.
- We perform security evaluations on our designs to confirm both first and second-order side-channel resistance.
- We propose two different methods to simulate fault injection attacks and confirm the fault resistance of our designs.

The remainder of this paper is organized as follows. Section II introduces the IPM and IPM-FD scheme, analyzing their building blocks and components, and main overhead, motivating the need for dedicated hardware support. Section III details the implementation of our extension unit and integration within RISC-V ISA. Section IV evaluates the performance of our extension unit, comparing the software-only implementation with the hardware accelerated unit. We also perform security evaluation of the extension in terms of side-channel leakage assessment and fault injection simulation. Finally, we conclude our paper in Section V.

II. Background

In this section, we introduce Inner Product Masking (IPM) and its fault-detection extension (IPM-FD), define the operation sets and analyze the resulting computational overhead that motivates hardware acceleration.

A. IPM and IPM-FD: Context

Inner Product Masking is a higher-order masking scheme against SCA. It encodes a secret s as the inner product of a public vector $\vec{L} \in \mathcal{K}^n$, which is also refereed as linear code, and a vector of random shares $\vec{R} \in \mathcal{K}^n$, where $s = \langle \vec{L}, \vec{R} \rangle = \sum_{i=1}^{n} L_i \otimes R_i$. We denote $\langle \cdot \rangle$ as inner product, n as the number of shares, $\otimes$ as field multiplication and $\oplus$ as field addition.

In IPM-FD scheme, secret s are replicated k-times into a secret vector $\vec{S} \in \mathcal{K}^k$. Linear code $\vec{L}$ is expanded into a matrix $\boldsymbol{L} \in \mathcal{K}^{k \times n}$ and is constructed as:

$$\boldsymbol{L} = \left(\begin{array}{c|ccc} & L_{1,k+1} & \cdots & L_{1,n} \\ I_k & \vdots & \ddots & \vdots \\ & L_{k,k+1} & \cdots & L_{k,n} \end{array} \right) \in \mathcal{K}^{k \times n} \tag{1}$$

with I_k being an identity matrix. We encode the secret into share vector $\vec{R} \in \mathcal{K}^n$, which is also refereed as *codeword* [15], where $\vec{S} = \boldsymbol{L} \cdot \vec{R}$. We use (n, k) to denote the configuration of IPM-FD, where n denotes the number of shares and k denotes the replication factor. To help better understand the relationship between IPM and IPM-FD, we use $(n{=}3, k{=}2)$ as an example. In such configuration, we have secret vector $\vec{S} = (s, s)$, share vector $\vec{R}{=}(R_1, R_2, R_3)$, and linear code matrix as:

$$\boldsymbol{L} = \begin{pmatrix} 1 & 0 & L_{1,3} \\ 0 & 1 & L_{2,3} \end{pmatrix} \in \mathcal{K}^{2 \times 3} \tag{2}$$

Thus, we have $k{=}2$ ways of reconstructing secret $s = \langle \boldsymbol{L}_1, R \rangle = \langle \boldsymbol{L}_2, R \rangle$, where each reconstruction is a IPM. When omitting 0, each $\boldsymbol{L}_k$ is a valid linear code in IPM. When performing IPM-FD operations, e.g. `IPM-FD-Mult`, the share vector $\vec{R}$ is first split into $k{=}2$ share sets, where $R^1{=}(R_1, R_3)$ and $R^2{=}(R_2, R_3)$. Then the corresponding IPM operation, i.e. `IPM-Mult`, will be performed on each share set independently, resulting in $R'^1{=}(R'_1, R'_3)$ and $R'^2{=}(R'_2, R'_4)$. Note that IPM-FD does not require k sets of independent shares, as shares R_{k+1} to R_n are reused across different reconstructions. Thus to maintain IPM-FD share representation, where different share sets bears the same $n-k$ masks, homogenization is applied to the results to align these masks before merging to a single share vector $\vec{R'} = (R'_1, R^p_2, R'_3)$ where $R^p_2{=}R'_2 \oplus (R'_3 \oplus R'_4) \otimes L_{2,3}$.

This representation integrates fault detection into the masking, by checking share vector s consistency to detect up to $k-1$ faults. Compared to repeating masked computations (e.g., temporal redundancy or simply executing IPM k times), IPM-FD achieves lower computational complexity, making it more efficient for concurrent protection against both SCA and FIA. We define $d = n - k$ as the masking order of the underlining IPM with $N = n - k + 1$ shares in each share set.

This shows that IPM-FD operations heavily relies on IPM building blocks, as we summarize other algorithms from IPM-FD into Tab I. Therefore, accelerating IPM operations benefits both schemes. Note that multiplication with constant is a linear operation in both IPM-FD and IPM, thus they share the same implementation. Subscript L is used to indicate involvement with the linear code L. In IPM-FD routines, the linear code L needs to be selected from the linear code matrix according to share set.

TABLE I
IPM-FD ROUTINES AS SIMPLE COMPOSITIONS

IPM-FD Routine	**IPM calls**	**Extra step**
`IPM-FD-Mask`$_L$	$k \times$ `IPM-Mask`$_L$	—
`IPM-FD-Unmask`$_L$	$k \times$ `IPM-Unmask`$_L$	—
`IPM-FD-Mult`$_L$	$k \times$ `IPM-Mult`$_L$	$(k-1) \times$ `IPM-FD-Homo`
`IPM-FD-Square`$_L$	$k \times$ `IPM-Square`$_L$	$(k-1) \times$ `IPM-FD-Homo`
`IPM-FD-Refresh`$_L$	$k \times$ `IPM-Refresh`$_L$	—
`IPM-FD-Homo`$_L$	IPM-FD only	—
`IPM-FD-ConstMult`	Same as `IPM-ConstMult`	—

B. Overhead Analysis

IPM domain operations are computationally intensive in software, especially for embedded devices. Specifically for Advanced Encryption Standard (AES) on which we will focus from now in this work, these building blocks operate over Galois Field $\mathbb{F}_{2^8}$. The cost of each IPM operation in terms of basic operation in field operations is summarized in Tab II. In AES field, the addition $\oplus$ can be efficiently implemented using XOR instruction that is supported on any devices. However, field multiplication $\otimes$ implementation involves several instructions and thus considerable time overhead. As mentioned earlier in Tab I, in IPM-FD, each routine applies its corresponding IPM operation to k share sets, yielding more overhead. This overhead increases latency and energy consumption in software-only implementations, highlighting the need for dedicated hardware support, which will be demonstrated in the next section.

TABLE II
OPERATION COUNTS FOR CORE IPM ALGORITHMS (IN TERMS OF N)

IPM Routine	**#** $\bigotimes$	**#** $\bigoplus$
`IPM-Mask`$_L$	$N-1$	$N-1$
`IPM-Unmask`$_L$	$N-1$	$N-1$
`IPM-Mult`$_L$	$3N^2-N$	$2N^2-2N$
`IPM-Refresh`$_L$	$N-1$	$N-1$
`IPM-Square`$_L$	$2N-1$	0
`IPM-ConstMult`	N	0
`IPM-FD-Homo`	$N-1$	$2N-2$

III. IMPLEMENTATION

In this section, we first implement a Random Number Generator unit to provide fresh randomness, essential for operations in the Masked Processing Unit, as will be detailed next. We further demonstrate the integration of the unit into a RISC-V core and the proposed customized instructions.

A. Random Number Generator

We implement and adapt a hybrid Random Number Generator (*h*RNG) module from [23] to provide fresh randomness to the masked operations, e.g. `IPM-Mult` and `IPM-Refresh`. The *h*RNG module consists of a Multi Ring Oscillator (MURO) based true RNG (*t*RNG) and a Trivium based pseudo RNG *p*RNG [24]. The MURO *t*RNG consists of five single AND oscillators, the output of which are XORed together and sampled at 50kHz. The output of the *t*RNG is used as seed to the *p*RNG, which can generate randomness at higher frequency (clock frequency in our case). The *p*RNG is configured to output 8 random bits per clock cycle at a cost of around 4 Look-up-tables (LUTs) per bit when implemented in FPGA [24]. To evaluate the *h*RNG, 1 million output bits were collected and passed the NIST Statistical Test Suite.

B. Masked Processing Unit

The Masked Processing Unit (MPU) is implemented to execute algorithms within the IPM(-FD) domain, the schematic of which is depicted in Fig. 1. Our MPU design can be configured to execute IPM domain operations only when setting $k = 1$. From now on, we set $k>1$ and the unit is configured to execute IPM-FD domain algorithms.

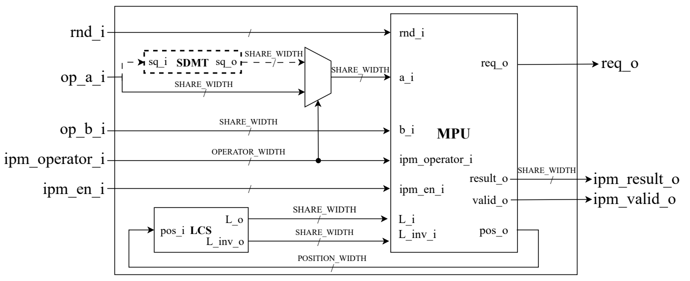

Fig. 1. MPU schematic

1) Trade-offs Between the Hardware Resources and Latency: Building on the work [21], which presents an efficient hardware approach for field multiplication in $\mathbb{F}_{2^8}$, our implementation integrates this advanced field multiplier to perform the field multiplications within the IPM(-FD) domain.

As detailed in Table II, most algorithms in the IPM domain require $N-1$ field multiplications with one noticeably exception of `IPM-Mult` requiring $3N^2 - N$ field multiplications. Therefore, it is impractical to complete `IPM-Mult` in one clock cycle as it incurs too much area overhead. According to [11], `IPM-Mult` can be seen as a matrix calculation of size (n, n) with n^2 elements. Hence, when considering processing each element individually in one clock cycle, we can effectively lower the field multiplications required at once. It is noteworthy that the randomness used for element i, j is the same for element j, i. Thus, we propose a coordination traversal strategy specifically for `IPM-Mult`$_L$, as depicted in Fig.2, where only one random byte needs to be stored. As a consequence, the computation of `IPM-Mult`$_L$ can be distributed across N^2 cycles, with each cycle involving three field multiplications. This approach effectively lowers the number of required field multipliers through time multiplexing.

As for `IPM-Square`$_L$, which requires $2N-1$ field multiplications, a dedicated Squared Data Mapping Table (SDMT) is utilized. The SDMT directly maps each byte to its corresponding field squared result, outputting N field squared results from N input bytes directly. The MPU then leverages these results

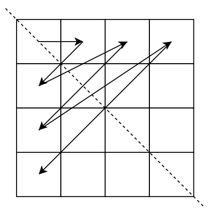

Fig. 2. Illustration of the Symmetric Coordination Traversal Strategy in a (4,4) Matrix

to execute the remaining $N-1$ field multiplications with minimized computation overhead. To achieve a higher throughput across a broad range of operations while optimizing hardware utilization, we set the optimal number of field multipliers as $\max(3, N)$.

This configuration ensures an effective balance between hardware utilization and throughput, achieving the optimal performance for our architecture. N is of course not unlimited, from practical point of view, N is limited at 4 as we work with 32-bit RISC-V ISA and each general purpose register can hold at most 4 shares. We summarize the multiplier usage of each operation in Tab. III.

TABLE III
PEAK MULTIPLIERS/CYCLE AND TOTAL CYCLES PER OPERATION AND CORRESPONDING INSTRUCTION .

Algorithm	$\otimes$/cycle	Cycles	Instruction
`IPM-FD-Mask`$_L$	$N-1$	$N-1/1$	ipm.fd.mask rd rs1
`IPM-FD-Unmask`$_L$	$N-1$	1	ipm.fd.unmask rd rs1
`IPM-FD-Mult`$_L$	3	N^2	ipm.fd.mult rd rs1, rs2
`IPM-FD-Refresh`$_L$	$N-1$	$N-1/1$	ipm.fd.refresh rd rs1
`IPM-FD-Square`$_L$	$N-1$	1	ipm.fd.square rd rs1
`IPM-FD-Homo`	$N-1$	1	ipm.fd.homo rd rs1, rs2
`IPM-ConstMult`	N	1	ipm.mult.const rd rs1, imme

2) Operation Logic: The MPU accepts two operands $\vec{A}, \vec{B}$ of N shares and produces $\vec{C}$ of the same size, though not all operations expect two inputs, such as `IPM-FD-Mask`, `IPM-FD-Unmask` and `IPM-FD-Square`$_L$. We hard-code the optimal linear codes $\boldsymbol{L}$ from [15] in a sub-module called Linear Code Storage (LCS) within the MPU. For each algorithm involving interaction with linear codes, an index *pos* is used to serve as internal counter to address the correct corresponding linear code L_{pos} from the LCS module, and updated accordingly. We outline how the MPU steers data-path resources (field multipliers, SDMT, scratch registers) for each algorithm. For clarity and consistency in demonstration, we use $N{=}4$ in the rest of the subsection.

a) ***Masking:*** `IPM-FD-Mask`$_L$ masks the first byte of $\vec{A}$ into N shares with $N{-}1$ fresh random masks and the associated code vector $\boldsymbol{L}_{pos}$. In IPM-FD, the first share set consumes $N{-}1$ random bytes thus requires $N{-}1$ cycles to finish, while subsequent sets reuse the same masks and complete in one cycle.

b) ***Unmasking*** *:* `IPM-FD-Unmask`$_L$ reconstructs the secret in one cycle by computing $\langle \boldsymbol{L}_{pos}, \vec{A} \rangle$ with the result written to $C[0]$. The k share sets are unmasked independently, resulting in k results for later consistency check.

c) ***Multiplication*** *:* `IPM-FD-Mult`$_L$ is an N^2 cycle operation that updates $\vec{C}$ in place while iterating two indices (i, j) over the share grid as shown in Fig. 3. The algorithm runs independently on each of the k share sets (k runs total, each consuming N^2 cycles). The first share set is selected as reference and apply $(k{-}1)$ `IPM-FD-Homo` steps to align masks across the k results.

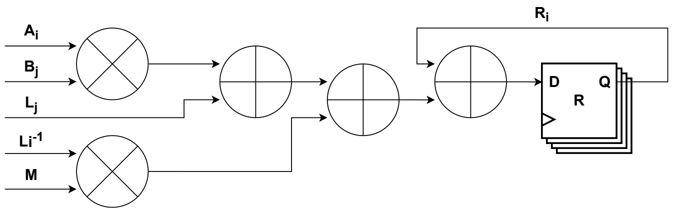

Fig. 3. Block Diagram of `IPM-FD-Mult`$_L$

d) ***Homogenization*** *:* `IPM-FD-Homo` aligns the masking of two share sets by adjusting $\vec{B}$ to match $\vec{A}$ (Fig. 4). It does not use $\vec{L}$ and yields a uniformly masked output for downstream checks.

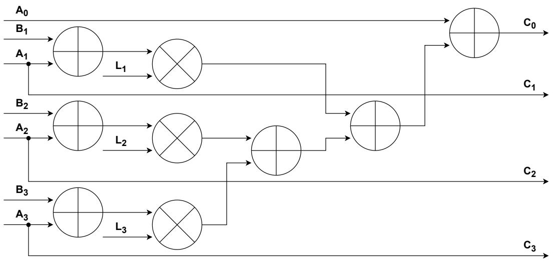

Fig. 4. Block Diagram of `IPM-FD-Homo`$_L$

e) ***Squaring*** *:* `IPM-Square`$_L$ utilizes SDMT to calculate squaring more efficiently than calling `IPMult`$_L$ with identical inputs. Similar to `IPM-FD-Mult`$_L$, k share sets are calculated independently and then homogenized.

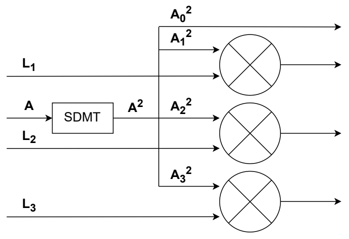

Fig. 5. Block Diagram of `IPM-Square`$_L$

f) ***Refresh*** *:* `IPM-FD-Refresh`$_L$ adds a vector orthogonal to $\boldsymbol{L}_{pos}$ to re-share $\vec{A}$ while preserving the encoded secret. Similarly to `IPM-FD-Mask`$_L$, the first share set consumes $N{-}1$ random words thus require $N{-}1$ cycles, while subsequent sets reuse the masks and complete in one cycle.

g) ***Multiplication with a Constant*** *:* The logic is identical in IPM and IPM-FD scheme as it performs multiplication with constant ($B[0]$) to each share. Homogenization is not required at the end of the algorithm.

C. RISC-V Integration

We select `cv32e40p` formerly known as RI5CY, as our baseline RISC-V core. `cv32e40p` is a compact and in-order 32-bit RISC-V core with a 4-stage pipeline and we configure the core to support RV32IMC ISA subset [25]. We integrate the MPU in the Execution (EX) stage of the pipeline. The randomness required by the unit is supplied from the *h*RNG implicitly. We also propose new ISEs and summarize the new instructions corresponding to the algorithms that the MPU supports along with their latency in III. For configuration with k=1, our ISEs can be essentially utilized to perform IPM algorithms, with only one linear code $\vec{L}$. As mentioned before, our MPU supports both single-cycle instruction as well as multi-cycle instructions. Thus, the pipeline is stalled whenever required.

IV. Evaluation

In this section, we first evaluate the hardware overhead of our MPU extension, followed by a side-channel security analysis. We conclude this section with two different types of simulation of injected fault to the platform to test the fault detection capability.

A. FPGA Evaluation

We synthesize our MPU with different configurations target at Artix 7 FPGA and summaries the hardware utilization in Tab. IV. Row 1 shows the hardware utilization of the default `cv32e40p` core, while row 2 shows that of `cv32e40p` and our *h*RNG module. As Tab. IV shows, the Look-up-tables (LUTs) and Flip-Flops (FFs) utilization increase as n increases when k is constant. However, when n is constant, we can see decrease in both LUTs and FFs when increasing k. This is due to the fact that in such case, N is essentially decreasing and thus our MPU is less complex, inducing fewer hardware utilization. We also observe a slight increase in F7 MUX primitive when $n = 5$, suggesting more complex logic with 7-input are synthesized. We do not observe increase in other FPGA primitives. Our implementation does not affect the critical path of the original design, thus has no impact on the maximum frequency.

TABLE IV
Hardware Resources Utilization of IPM-FD of different configurations

k	n	N	LUTs	FFs	F7 Mux
base (`cv32e40p`)			7986	2744	443
base+*h*RNG			8101	3113	443
2	2	1	8116	3214	443
	3	2	8220	3212	443
	4	3	8324	3230	443
	5	4	8438	3238	464
3	3	1	8136	3216	443
	4	2	8216	3214	443
	5	3	8390	3231	453

B. Performance Evaluation

In this subsection, we evaluate the performance for different configuration of IPM-FD. Table V summarizes the clock cycles required to perform one encryption of AES-128 for different configurations of IPM-FD. We focus on cases where $k > 1$ as faults can only be detected when k is at least 2. The reference column shows the results of C-implementation from [15] ported to our RISC-V ISA platform. The reference design stores the IPM-FD share vector into a $1 - D$ n array, and first unpacks into a $2 - D$ $k \times N$ array when performing IPM-FD operations, i.e., IPM-FD-Squaring, IPM-FD-Multiplication and IPM-FD-Homogenization. These IPM-FD operations then call corresponding IPM operations k times on the share sets. We then replace these software routines with our dedicated instructions in our evaluation of hardware column. Reference design uses iteration index to keep track of which share set to process, while we use internal counter *pos* to record as specified in III-B2. The speedup factor is calculated by dividing the clock cycles of reference design with our hardware implementation.

From Tab. V we can see a clear trend that the required clock cycles increases linearly with the number of shares N. This trend does not hold for our hardware implementations, where the overhead only increases subtly. This is due to the fact that with our implementation, instruction counts of core IPM-FD algorithms, remain constant for different n or N, and only scale with k. Thus, we observe higher speedup factor when increasing n while keeping k constant. We can achieve upto $16.55\times$ and $12.61\times$ speedup factor for the configuration of $n = 5, k = 2$ and $n = 5, k = 3$, respectively.

TABLE V
Performance Evaluation of IPM-FD of different flavor

k	n	N	Reference	Hardware	Speedup factor
2	2	1	14050146	5831178	2.41x
	3	2	38086758	6180969	6.16x
	4	3	72052988	6600158	10.92x
	5	4	115956861	7005807	16.55x
3	3	1	19793310	7534293	2.63x
	4	2	56188261	7905107	7.11x
	5	3	106488797	8443037	12.61x

C. Side-channel security evaluation

In this subsection, we evaluate the side-channel security of our IPM-FD implementations using the Test Vector Leakage Assessment (TVLA) methodology [26]. We measure the instantaneous power consumption on a ChipWhisperer CW305 evaluation board with the `cv32e40p` core running at 50 Mhz, using a PicoScope 6000E series oscilloscope sampling at 1.25GS/s. The TVLA test is essentially a Welch's (two-tailed) t-test for each sample in the collected traces. We follow the standard fixed-vs-random approach, collecting two sets of power traces: one with fixed plaintext and the other with randomly generated plaintext. The resulting t-score vector determines whether there exhibits leakage in the implementation. A common chosen threshold value in literature is ± 4.5 for uni-variate tests, corresponding to 99.999% confidence value.

To begin with, we start with the configuration of $n = 3, k = 2$ which means $N = 2$ and thus first-order IPM, we perform uni-variate TVLA on the implementations. We first perform a sanity check on our setup by turning off the *h*RNG which essentially disabling the masking. Fig. 6 (up) shows the TVLA result with RNG off. There is clear leakage as maximum t-score exceeds the ± 4.5 threshold in multiple instances with only 50k traces. With RNG enabled, we can see from Fig. 6 (middle and bottom), our IPM-FD-Multiplication and IPM-FD-Square implementations exhibit no sign of leakage after 1 million traces. We then continue our evaluation with configuration of $n = 4, k = 2$, corresponding to second-order IPM, with uni- and multi-variate TVLA tests. Fig. 7 shows no leakage in the uni-variate TVLA test for both IPM-FD-Multiplication and Square implementation. Moreover, IPM with $N = 3$ shares shows no second-order leakage in the bi-variate TVLA test as well, as shown in the bottom left triangle in Fig. 8. We also perform the test with RNG off and observe second-order leakage, as depicted in the top right triangle in Fig. 8. According to [27], the threshold for TVLA tests involving large number of test points, e.g. bi-variate tests, can be slightly increased to eliminate false-positive leakage and we set as ± 5.

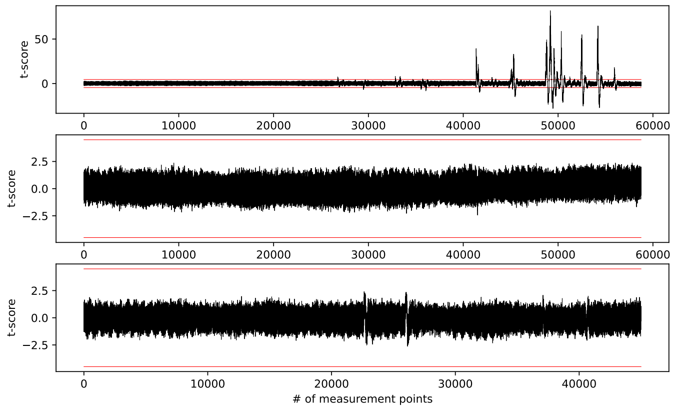

Fig. 6. Uni-variate TVLA result for $n = 3, k = 2$: IPM-FD-Multiplication with RNG off with 50k traces (top), IPM-FD-Multiplication with RNG on with 1M traces (middle) and IPM-Square with RNG on with 1M traces (bottom).

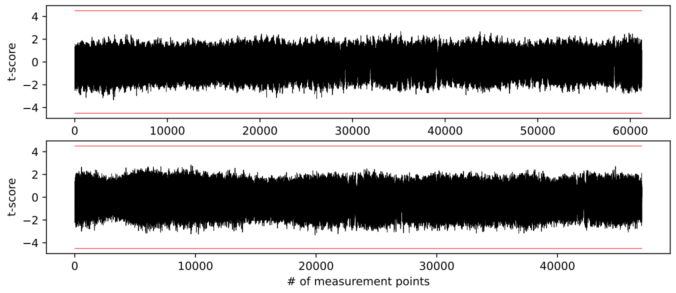

Fig. 7. Uni-variate TVLA result for $n = 4, k = 2$: IPM-FD-Multiplication (top), and IPM-FD-Square (bottom), both with RNG on and 1M traces.

Finally, we conclude our side-channel evaluation by performing an additional uni-variate TVLA test on configuration of $n = 4, k = 3$, where corresponds to, again, first-order IPM. Fig. 9 confirms that with 1M traces, both algorithms exhibit no sign of leakage.

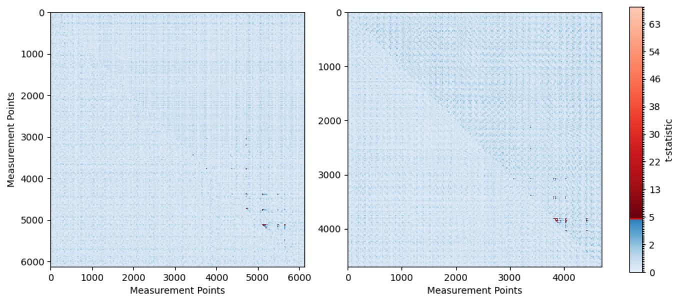

Fig. 8. Bi-variate TVLA result for $n = 4, k = 2$: IPM-FD-Multiplication (left), and IPM-FD-Square (right). Bottom left triangle: RNG on with 1M traces. Top right triangle: RNG off with 200k traces.

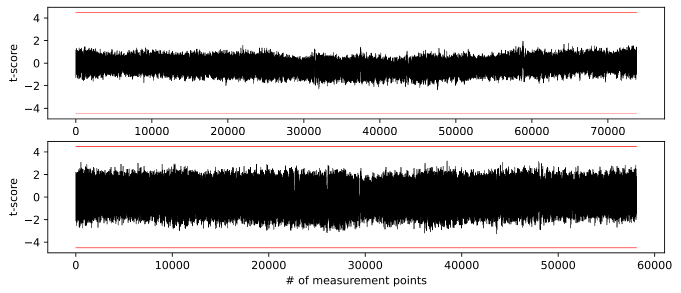

Fig. 9. Uni-variate TVLA result for $n = 4, k = 3$: IPM-FD-Multiplication (up), and IPM-Square (bottom), both with RNG on and 1M traces.

D. Fault-detection evaluation

To evaluate the fault detection capability, we employ two methods to simulate the fault attacks.

```
    share[0] ^= f0;
    share[1] ^= f1;
    share[2] ^= f2;
```

Fig. 10. Pseudo-code for injecting fault in software routine with $n = 3$

We first simulate the fault injection by flipping the codeword during the execution in software as shown in Fig 10. This code snippet can be inserted at arbitrarily and the faults f_i can be chosen freely. Thus we consider this to be a very powerful simulation. According to [15], tempering one share in the codeword is considered as one fault. For configuration with $n = 3, k = 2$, IPM-FD can detect up to $k-1 = 1$ injected faults in worst case which means carefully selected injected faults. We first simulate injecting one fault by setting $f_1 = 0$ and $f_2 = 0$ or simply commenting out that statements, we confirm that IPM-FD can detect any faults with any selected value f_0. The same also applies to any selected value f_1 or f_2. We then simulate two faults by setting $f_2 = 0$ and randomly select fault value f_0 and f_1. When $f_0 = f_1$, the faulted codeword turns into another valid codeword, thus IPM-FD cannot detect the injected fault, confirming its capability of detecting $k-1$ fault in worst case.

Second, we implement a dummy saboteur module besides the MPU in the EX stage of the `cv32e40p` core. The output result of MPU is fed into the saboteur module. We propose one more instruction **fi rs1**. Instruction **fi** enables the saboteur module and applies a mask (fault) stored in **rs1** register to the MPU result. The saboteur replaces the original MPU output with the faulted output and essentially injects faults into the pipeline. This fault can be arbitrarily injected through the execution of the algorithm. By this way, we can simulate a more powerful hardware injected fault with one instruction. Again we confirm that our MPU provides the same level of fault resistance compared to reference software-only implementations.

V. Conclusion

In this work, we target and analyze the IPM-FD and IPM building blocks After comparing the trade-off between hardware resources and latency incurred by each algorithm, we then propose a MPU extension unit to execute all operations in hardware to improve efficiency of such scheme in software. We adapt the reference design with our hardware extension and given the hardware nature of supporting parallelism, our design can achieve more noticeable performance gain with higher masking order n and constant k. Finally, we evaluate with first side-channel evaluation and fault injection simulation to confirm our design can help provide resistence against both attacks. To our knowledge, this is the first RISC-V ISE-based work to address both SCA and FIA simultaneously.

References

[1] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[2] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology - CRYPTO '99*, ser. LNCS, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 388–397.

[3] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Cryptographic Hardware and Embedded Systems - CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3156. Springer, 2004, pp. 16–29. [Online]. Available: https://doi.org/10.1007/978-3-540-28632-5_2

[4] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, ser. Lecture Notes in Computer Science, Ç. K. Koç, D. Naccache, and C. Paar, Eds., vol. 2162. Springer, 2001, pp. 251–261. [Online]. Available: https://doi.org/10.1007/3-540-44709-1_21

[5] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, ser. Lecture Notes in Computer Science, N. Koblitz, Ed., vol. 1109. Springer, 1996, pp. 104–113. [Online]. Available: https://doi.org/10.1007/3-540-68697-5_9

[6] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proc. IEEE*, vol. 94, no. 2, pp. 370–382, 2006. [Online]. Available: https://doi.org/10.1109/JPROC.2005.862424

[7] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults (extended abstract)," in *EUROCRYPT '97*, ser. LNCS, vol. 1233. Springer, 1997, pp. 37–51.

[8] C. Clavier, B. Feix, G. Gagnerot, and M. Roussellet, "Passive and active combined attacks on aes???combining fault attacks and side channel analysis," in *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2010, Santa Barbara, California, USA, 21 August 2010*, L. Breveglieri, M. Joye, I. Koren, D. Naccache, and I. Verbauwhede, Eds. IEEE Computer Society, 2010, pp. 10–19. [Online]. Available: https://doi.org/10.1109/FDTC.2010.17

[9] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, ser. Lecture Notes in Computer Science, vol. 1666. Springer, 1999, pp. 398–412. [Online]. Available: https://doi.org/10.1007/3-540-48405-1_26

[10] L. Goubin, "A sound method for switching between boolean and arithmetic masking," in *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, ser. Lecture Notes in Computer Science, Ç. K. Koç, D. Naccache, and C. Paar, Eds., vol. 2162. Springer, 2001, pp. 3–15. [Online]. Available: https://doi.org/10.1007/3-540-44709-1_2

[11] J. Balasch, S. Faust, B. Gierlichs, C. Paglialonga, and F. Standaert, "Consolidating inner product masking," in *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, ser. Lecture Notes in Computer Science, T. Takagi and T. Peyrin, Eds., vol. 10624. Springer, 2017, pp. 724–754. [Online]. Available: https://doi.org/10.1007/978-3-319-70694-8_25

[12] J. Balasch, S. Faust, and B. Gierlichs, "Inner product masking revisited," *IACR Cryptol. ePrint Arch.*, p. 105, 2015. [Online]. Available: http://eprint.iacr.org/2015/105

[13] J. Bringer, C. Carlet, H. Chabanne, S. Guilley, and H. Maghrebi, "Orthogonal direct sum masking - A smartcard friendly computation paradigm in a code, with builtin protection against side-channel and fault attacks," in *Information Security Theory and Practice. Securing the Internet of Things - 8th IFIP WG 11.2 International Workshop, WISTP 2014, Heraklion, Crete, Greece, June 30 - July 2, 2014. Proceedings*, ser. Lecture Notes in Computer Science, D. Naccache and D. Sauveron, Eds., vol. 8501. Springer, 2014, pp. 40–56. [Online]. Available: https://doi.org/10.1007/978-3-662-43826-8_4

[14] M. Pflanz, K. Walther, C. Galke, and H. T. Vierhaus, "On-line error detection and correction in storage elements with cross-parity check," in *8th IEEE International On-Line Testing Workshop (IOLTW 2002), 8-10 July 2002, Isle of Bendor, France*. IEEE Computer Society, 2002, pp. 69–73. [Online]. Available: https://doi.org/10.1109/OLT.2002.1030186

[15] W. Cheng, C. Carlet, K. Goli, J. Danger, and S. Guilley, "Detecting faults in inner-product masking scheme - IPM-FD: IPM with fault detection," in *Proceedings of 8th International Workshop on Security Proofs for Embedded Systems, PROOFS 2019, colocated with CHES 2018, Atlanta, GA, USA, August 24, 2019*, ser. Kalpa Publications in Computing, K. Heydemann, U. Kühne, and L. Li, Eds., vol. 11. EasyChair, 2019, pp. 17–32. [Online]. Available: https://doi.org/10.29007/fv2n

[16] O. Keren and I. Polian, "IPM-RED: combining higher-order masking with robust error detection," *J. Cryptogr. Eng.*, vol. 11, no. 2, pp. 147–160, 2021. [Online]. Available: https://doi.org/10.1007/s13389-020-00229-4

[17] H. Groß, S. Mangard, and T. Korak, "Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order," in *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, B. Bilgin, S. Nikova, and V. Rijmen, Eds. ACM, 2016, p. 3. [Online]. Available: https://doi.org/10.1145/2996366.2996426

[18] P. Kiaei and P. Schaumont, "Domain-oriented masked instruction set architecture for RISC-V," *IACR Cryptol. ePrint Arch.*, p. 465, 2020. [Online]. Available: https://eprint.iacr.org/2020/465

[19] S. Gao, J. Großschädl, B. Marshall, D. Page, T. H. Pham, and F. Regazzoni, "An instruction set extension to support software-based masking," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 4, pp. 283–325, 2021. [Online]. Available: https://doi.org/10.46586/tches.v2021.i4.283-325

[20] F. Lozachmeur and A. Tisserand, "A RISC-V instruction set extension for flexible hardware/software protection of cryptosystems masked at high orders," in *66th IEEE International Midwest Symposium on Circuits and Systems, MWSCAS 2023, Tempe, AZ, USA, August 6-9, 2023*. IEEE, 2023, pp. 360–364. [Online]. Available: https://doi.org/10.1109/MWSCAS57524.2023.10405991

[21] S. Cui and J. Balasch, "Efficient software masking of AES through instruction set extensions," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023*. IEEE, 2023, pp. 1–6. [Online]. Available: https://doi.org/10.23919/DATE56975.2023.10137150

[22] H. Cheng, D. Page, and W. Wang, "eliminate: a leakage-focused ISE for masked implementation," *IACR Trans. Cryptogr. Hardw. Embed.*

Syst., vol. 2024, no. 2, pp. 329–358, 2024. [Online]. Available: https://doi.org/10.46586/tches.v2024.i2.329-358

[23] S. Cui and J. Balasch, "Configurable loop shuffling via instruction set extensions," in *35th IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2024, Hong Kong, July 24-26, 2024*. IEEE, 2024, pp. 45–53. [Online]. Available: https://doi.org/10.1109/ASAP61560.2024.00021

[24] G. Cassiers, L. Masure, C. Momin, T. Moos, A. Moradi, and F.-X. Standaert, "Randomness generation for secure hardware masking - unrolled trivium to the rescue," Cryptology ePrint Archive, Paper 2023/1134, 2023, https://eprint.iacr.org/2023/1134. [Online]. Available: https://eprint.iacr.org/2023/1134

[25] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. Gurkaynak, and L. Benini, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," Feb. 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7864441

[26] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side channel resistance validation," http://csrc.nist.gov/newsevents/non-invasive-attack-testing-workshop/08Goodwill.pdf, 2011.

[27] L. Zhang, A. A. Ding, F. Durvaux, F. Standaert, and Y. Fei, "Towards sound and optimal leakage detection procedure," *IACR Cryptol. ePrint Arch.*, p. 287, 2017. [Online]. Available: http://eprint.iacr.org/2017/287