IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 2, pp. 677–713. DOI:10.46586/tches.v2024.i2.677-713

UpWB: An Uncoupled Architecture Design for White-box Cryptography Using Vectorized Montgomery Multiplication

Xiangren Chen[1], Bohan Yang[1], Jianfeng Zhu[1], Jun Liu[3], Shuying Yin[1], Guang Yang[1], Min Zhu[2], Shaojun Wei[1] and Leibo Liu[1,*]

[1]Beijing National Research Center for Information Science and Technology (BNRist), School of Integrated Circuits, Tsinghua University, Beijing, China.
chen-xr23@mails.tsinghua.edu.cn;{bohanyang,jfzhu,yinshuying,linquanyehe,wsj,liulb}@tsinghua.edu.cn
[2]Wuxi Micro Innovation Integrated Circuit Design Co., Ltd., Wuxi, China. zhumin@mucse.com
[3]Shaanxi Normal University, Xi'an, China. jliu6@snnu.edu.cn
[*] Corresponding Author.

Abstract. White-box cryptography (WBC) seeks to protect secret keys even if the attacker has full control over the execution environment. One of the techniques to hide the key is space hardness approach, which conceals the key into a large lookup table generated from a reliable small block cipher. Despite its provable security, space-hard WBC also suffers from heavy performance overhead when executed on general purpose hardware platform, hundreds of magnitude slower than conventional block ciphers. Specifically, recent studies adopt nested substitution permutation network (NSPN) to construct dedicated white-box block cipher [BIT16], whose performance is limited by a massive number of rounds, nested loop dependency and high-dimension dynamic maximal distance separable (MDS) matrices.

To address these limitations, we put forward UpWB, an **u**ncou**p**led and efficient accelerator for NSPN-structure WBC. We propose holistic optimization techniques across timing schedule, algorithms and operators. For the high-level timing schedule, we propose a fine-grained task partition (FTP) mechanism to decouple the parameter-oriented nested loop with different trip counts. The FTP mechanism narrows down the idle time for synchronization and avoids the extra usage of FIFO, which efficiently increases the computation throughput. For the optimization of arithmetic operators, we devise a flexible and vectorized modular multiplier (VMM) based on the complexity-reduced Montgomery algorithm, which can process multi-precision variable data, multi-size matrix-vector multiplication and different irreducible polynomials. Then, a configurable matrix-vector multiplication (MVM) architecture with diagonal-major dataflow is presented to handle the dynamic MDS matrix. The multi-scale (Inv)Mixcolumns are also unified in a compact manner by intensively sharing the common sub-operations and customizing the constant multiplier.

To verify the proposed methodology, we showcase the unified design implementation for three recent families of WBCs, including SPNbox-8/16/24/32, Yoroi-16/32 and WARX-16. Evaluated on FPGA platform, UpWB outperforms the optimized software counterpart (executed on 3.2 GHz Intel CPU with AES-NI and AVX2 instructions) by 7× to 30× in terms of computation throughput. Synthesized under TSMC 28nm technology, 36× to 164× improvement of computation throughput is achieved when UpWB operates at the maximum frequency of 1.3 GHz and consumes a modest area 0.14 mm^2. Besides, the proposed VMM also offers about 30% improvement of area efficiency without pulling flexibility down when compared to state-of-the-art work.

Keywords: white-box cryptography · space hardness · domain-specific accelerator

Received: 2023-10-15 Accepted: 2023-12-15 Published: 2024-03-12

1 Introduction

White-box cryptography originates from works [CEJvO02b][CEJvO02a] by Chow, Eisen, Johson and Oorschot, which are categorized as the CEJO scheme. Through combining look-up table (LUT) with encoding techniques, CEJO scheme aims to protect the standard block cipher (i.e., Advanced Encryption Standard) from key extraction attack even under white-box attack model. This strong security model assumes that the hostile user has full access and control to the environment, such as manipulating memory addresses, content, execution flow and so forth. In contrast, the conventional black-box security model assumes that only the external execution environment of cryptographic algorithm, such as the chosen plaintext and ciphertext pair, is observed by the attacker. Additionally, the well-known side-channel attack that examines execution time, power consumption, etc. leads to the gray-box model of relatively middle-level hypothetical strength. As of classical application, devices on the server side are usually executed under black-box model, while the client side commonly corresponds to gray-box or white-box model. Recently, WBC is in fast-rising demand among many security-critical scenarios, such as digital right management (DRM), mobile payment, banking and memory-leakage resilient software and so on [BIT16] [DLPR13][BABM20].

1.1 Space-hard White-box Block Ciphers

Incompressibility. One research line of WBC follows the CEJO framework to provide white-box implementation for the standard block cipher, such as the white-box version of Advanced Encryption Standard (AES) [CEJvO02a] and lightweight block cipher [CG22]. Nevertheless, almost all of them are still penetrated by key extraction attack up to now. The other research line devises new dedicated white-box block ciphers (DWBCs) to achieve provable security while imposes large computational overheads, which is the focus of this paper. As mentioned in [DLPR13] [BABM20], the basic aim of white-box programs includes resisting key extraction attack and leakage of message (one-wayness). Additionally, the code-lifting attack is also an important security issue under the white-box setting. In the syntax of code-lifting attack, an attacker could directly copy the code program and run it on chosen devices without the need to recover the value of secret key. At this point, the whole extracted program is equivalent to a big key. Incompressibility, as a security property to mitigate the code-lifting attack, is proposed in [DLPR13], which is popularly adopted in modern dedicated white-box block ciphers. The purpose of incompressibility is to increase the difficulty of extracting and transferring code programs by converting the programs into a significantly larger but functionally equivalent version. Furthermore, the program only keep functional in their complete form, i.e., the malfunction will occur when fragments of it are moved. In this way, if the adversary obtains part of the program, he should not be able to recover the value of secret key or should not be able to derive the compressed version of equivalent functionality and use it to decrypt arbitrary ciphertexts. For brevity, this work would like to refer to literatures [BAB+19] [DLPR13] for formal definition of incompressibility.

When designing the dedicated white-box block cipher, a typical method to achieve incompressibility is to derive a large and incompressible program from a small-sized symmetric encryption scheme by implementing the key-related encryption functions with look-up tables. In this case, security against key extraction attack in the white-box setting is reduced to the well studied problem of key recovery for block ciphers in the standard black-box setting. Usually, the table-based incompressible version is referred to as the *white-box implementation* (formalized as big key mode in [BBL23]) while the original small-sized encryption scheme is referred to as the *black-box implementation* [BIT16] (also formalized as small key mode in [BBL23]).

Space-hardness. Following the concept of incompressibility, [BI15] also proposes a

novel notation named as (M,Z)-space hardness, which quantifies the security against code lifting by the amount of code that needs to be extracted from the implementation by a white-box attacker to maintain its functionality.

Theorem 1 ((M, Z)-space hardness [BI15])**.** *The implementation of a block cipher is (M,Z)-space hard if it is infeasible to encrypt (decrypt) any randomly drawn plaintext (ciphertext) with probability of more than* 2^Z *given any code (table) of size less than M.*

It is noted in [BI15] that weak white-box security can be seen as a special case of (M,Z)-space hardness and corresponds to (M, 0)-space hardness. [FKKM16] also proposes the notation of weak incompressibility and makes it as essentially a formalization of space hardness. The first space-hard block cipher named as SPACE [BI15] is designed with different code sizes, which is applicable to a wide range of environments and use cases. Without the necessity of the external code, it is of higher portability than the CEJO scheme as well. However, the security-prioritized design strategy also leads to heavy performance penalties and restricts its wide application. Since the birth of space-hard cipher, algorithm designers have been conducting studies to improve its performance. One of the prevailing techniques is to utilize nested SPN structure, where the S-box is developed by another small-scale SPN block cipher. [BIT16] presents the first NSPN based white-box block cipher named as SPNbox, which utilizes AES variants to build the internal S-box and applies MDS matrix to develop the linear layer. More recently, [KI21] puts forward an updatable white-box scheme (Yoroi) based on the partial MDS matrix, which could refresh the LUT without extra re-encryption. WARX [LRH+22] employs addition/rotation/XOR (ARX) primitive and random MDS matrix to improve the overall performance. However, contemporary DWBCs still suffer from limited performance due to the enormous round count, high dimension MDS matrix and nested loop structure (detailed in section 2).

1.2 The Story: Accelerating Black-box Implementation of WBC

Motivations. In general, DWBCs are designed with different security parameters and table sizes to meet a variety of use cases. The capacity of incompressible LUT under white-box implementation ranges from several KiloBytes to GigaBytes, allowing us to seek balance between security strength and resource constraint. Actually, for WBC with medium-sized parameter (e.g. SPNbox-16), the speed (measured by average cycle per bye) of black-box implementation is about three times slower than that of the white-box version when executed on software platform. The main reason for this performance gap is that the black-box implementation incurs nested loop dependency, which is hard to be parallelized under software platform and prone to deteriorate the performance. Modern CPU could resort to specific instruction (e.g. AES NI) or vector instruction (i.e. AVX) to achieve parallelism in the data level, but the nested loop dependency hinders the further parallelization in the loop level. Our objective is to show that the performance of space-hard cipher under black-box implementation can be significantly promoted by utilizing well-designed hardware accelerator. UpWB could be integrated into the cloud server and it would capture better area and energy efficiency than software-only platform. The recent progress on WBC has been nothing short of spectacular, which makes us optimistic that future advancements will bring the power of WBC to many more applications.

WBC is originally invented for *software protection* when we are short of the hardware support. Hence, questions may arise that there is no need to apply WBC if we already possess the secure hardware resource. The feasibility and profit to adopt hardware acceleration are summarized as following from three-fold perspectives.

(1) We would like to emphasize that this work aims to accelerate the server-side WBC *under black-box implementation.* The aforementioned question only considers the white-box implementation by default but fails to hold water for black-box implementation. The execution environment of secure hardware is commonly assumed to be non-white-box

[SMG16a], which is consistent with the trusted environment of server-side and exactly explains why we only accelerate the WBC under black-box setting. The client-side WBC is still implemented with software program under white-box setting as usual.

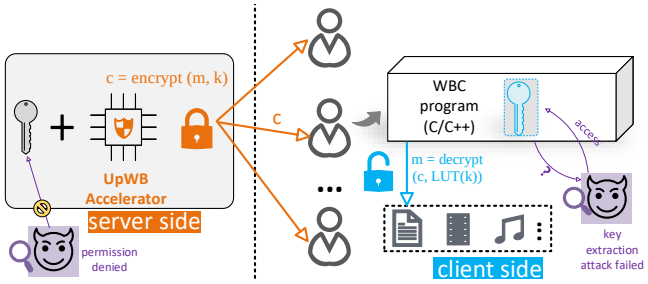


Figure 1: The application of WBC in cloud-based content distribution. **Hardware WBC** with small key is deployed under black-box setting (only the input and output can be observed). **Software WBC** with big key is deployed under white-box setting (untrusted open environment).

(2) Both white-box and black-box implementations are usually deployed in pair to fulfill the application of DWBC. Compared with the memory-hard white-box implementations of WBC, the black-box version features the advantages of facilitating fast key switching and consuming much less storage. Since the server side is assumed to already know the secret keys, the server-side WBC could be implemented in form of *black-box implementation*. Space-hard block ciphers actually resemble the asymmetrically hard functions in terms of memory hardness as noted in [BP17], which create two classes of users: one party (like the client side) has to compute the *white-box implementation* with increased memory hardness, while the other party (like the server side) who knows the secret can evaluate the *black-box implementation* with original memory efficiency. However, as mentioned before, the performance of black-box implementation on general-purpose processor is still problematic, which is hard to meet the needs of massive client terminals and the highly concurrent applications on the server side. Adopting hardware acceleration for the *black-box implementation* naturally boosts the throughput of WBC and matches the black-box security model of the server side, thus compensating the inferiority of space-hard ciphers to performance.

For example, Figure 1 depicts the application of WBC in cloud-based content distribution [BIT16]. Cloud server has to encrypt contents in the *black-box setting* (thus retain conventional memory efficiency) and distribute them to user devices. *Since running multiple white-box implementations for all users would require a prohibitive amount of memory, it is unreasonable to replace the server-side black-box implementation with memory-bound white-box version.* The UpWB accelerator is leveraged to improve the computation throughput for the server side. User devices decrypt the contents in the white-box setting using the incompressible software program, without needing to rely on hardware. Additionally, although most private and remote servers could be protected much better than edge devices, it may still have additional threats, such as cache-timing attack [GSM15]. This attack utilizes the architectural features of software processor, such as data dependency and cache memory access time, to extract the secret key, which has recently received wide attentions. By implementing the black-box implementation with a specific hardware accelerator, this issue could be addressed at the cipher-implementation level, in analogy with the profit of AES-NI instruction. Based on the above analysis, we explicitly summarize the difference between this work and the conventional WBC applied in DRM, as shown in Table 1.

(3) The server side under black-box implementation always deals with a large number of user keys simultaneously and demands a relatively higher throughput. In particular, devices

on the server-side inevitably encounter cryptographic algorithms with diverse security parameters [LWD+18]. It is profitable to devise a configurable hardware architecture supporting different algorithms and parameters for the server side. Although a few works have achieved performance improvement for WBC on the general-purpose processing platform [RFS+19][TGS+22], the final computation throughput is still limited because CPUs cannot directly explore the parallelism of nested operations well enough. Therefore, a domain-specific hardware accelerator is more promising to achieve better performance and energy efficiency. Hardware acceleration of basic primitives tends to exert down-stream effects. We hope that this acceleration can also motivate more applications of WBC for developing potentially fancy cryptographic infrastructure [DLS+22].

Table 1: The comparison of deployments for WBC in DRM.

Features	Conventional WBC [BIT16]		**This Work (UpWB)**	
	server side	client side	server side	client side
platform	software	software	**hardware**	software
security model	black-box	white-box	black-box	white-box
main attacks	KEA CTA, etc.	KEA CLA, etc.	**KEA** ~~CTA~~, etc.	KEA CLA, etc.
protection mechanism	access control etc.	incompressibility etc.	access control + hardware, etc.	incompressibility etc.
speed/energy efficiency	—	—	↑	—
concurrent application	—	—	✓	—
fast key-switching	—	✗	✓	✗
storage overhead	low (KBs)	high (MBs)	low (KBs)	high (MBs)

NOTES: KEA - key extraction attack. CTA - cache-timing attack [GSM15]. CLA - code-lifting attack. This table also reflects the difference between black-box and white-box implementation.

Discussions on the security threats and applications. The recent hybrid code-lifting attack [TI22] supposes that there are white-box and black-box attackers working together to achieve program recovery. The black-box attacker receives the leakage generated by the white-box attacker and uses it to perform cryptanalysis under the black-box setting, which determines whether the attack succeeds or not. In other words, the leakage from the collaborative white-box attacker enhances the ability of black-box attacker, which is similar to the attack model of strong incompressibility in terms of motivation. However, SPNbox and Yoroi themselves are designed under the assumption that the property of space-hardness is achieved in the presence of only a white-box attacker, i.e., without such collaboration of the hybrid-code lifting. In the context of hybrid code-lifting attack model, the 128-bit security strength of SPNbox against key-recovery attacks in the black-box model is compromised, but not meaning that SPNbox is completely broken. It is also remarked in [TI22] that increasing the number of rounds in SPNbox by several rounds (e.g., 12-round SPNbox-16) produces a good candidate to resist such hybrid code-lifting attack, if the original security strength is still demanded. As for the Yoroi scheme, its canonical representation and partial MDS matrix are vulnerable to the hybrid code-lifting attack, which cannot be circumvented by just increasing the number of round. Thus, Yoroi should be deployed under the very limited use case which supposes the black-box attacker cannot obtain the leakage, as suggested by [TI22]. Based on the above considerations, the proposed UpWB hardware accelerator supports configurable parameters, which thereby is able to efficiently update the algorithmic parameters of SPNbox, Yoroi and WARX, including the number of rounds and random MDS matrix elements. [MN11] also makes an intuitive remark that the dynamic generation of MDS matrix causes the prediction of statistical properties and relationships between plain and cipher texts more difficult, which helps to avert the linear and differential cryptanalysis.

To mitigate code-lifting attacks is particularly central to the application of white-box

cryptography besides the security against key extraction, as explicated in [BABM20]. The concept of incompressibility requires the cryptographic programs to be implemented with a very large program, which seems to be unsuitable for resource-constrained mobile devices and internet of things. As a consequence, large-sized space-hard ciphers are of main interests to the typical applications like DRM, but seem not to be the appropriate solution to mobile payment applications, which favor more lightweight deployments and demand other security properties including confidentiality, integrity and so on [BABM20]. Therefore, in addition to the property of incompressibility, other notable techniques like application binding, hardware binding [BBF+20], traceability [DLPR13] and so forth, should also be carefully considered for a broader applicability.

State-of-the-art. To our best knowledge, hardware designs for DWBCs have not been reported by now. Although some works focus on the performance improvement of WBCs under central processing unit (CPU) execution [RFS+19][TGS+22], the final throughput is still limited and only the white-box implementation is evaluated. Some works [SMG16b] [Sas18] implement the relatively lightweight white-box AES/Noekeon version on FPGA rather than the type of DWBCs discussed in this paper. In general, the nested loop dependency and computation-intensive modular operations are unfriendly to CPUs, which commonly have a low degree of parallelism. Graphic processing units (GPUs), on the other side, have rich computation resources but consumes much more energy and money cost. Nevertheless, hardware design has good scalability and could leverage the specialized data-flow information to achieve better computation efficiency. As a special focus, various algorithms for modular multiplication over GF(2^m) have been intensively studied. However, an efficient and vectorized MM algorithm supporting random MDS matrix is still in lack, which will be further discussed in section 2 and 5.

Contribution. In this work, we focus on the hardware acceleration for multiple NSPN-based block ciphers under black-box setting. Our main contributions are illustrated as following:

- We analyze several NSPN-based WBCs to identify the underlying operations and point out their distinction from conventional block ciphers. We manifest that the nested loop is prone to deteriorate the performance (Section 2).

- For the first time, we develop UpWB, an uncoupled and efficient hardware architecture for the server-side WBC under black-box setting, which can accelerate three series of DWBCs including SPNbox-8/16/24/32, Yoroi-16/32 and WARX-16. We introduce a fine-grained task partition mechanism to decouple the nested loop dependency, which can serve as a generic method and efficiently promote the computation throughput per area (Section 3).

- We propose several optimized hardware modules to achieve decent area-time efficiency, which encompasses 1) an adaptive and redundancy-free VMM to process multi-precision data, multi-size vector length and different irreducible polynomials; The refined VMM achieves almost the lowest area complexity among state-of-the-art works without pulling performance and flexibility down. 2) a configurable MVM to deal with multi-size MDS matrices; The proposed MVM architecture adopts diagonal-major data-flow to circumvent the vast connection cost and reduce memory footprint. 3) unified multi-scale (Inv)MixColumns by intensively sharing sub-operations; A design generator is developed to customize the constant modular multiplier for minimal circuit depth and 32% average savings on area cost (Section 4).

- We evaluate UpWB based on FPGA implementation and ASIC synthesis. The FPGA implementation of UpWB outperforms the optimized software counterpart by 7× to 30× in terms of computation throughput. The synthesis result of TSMC 28nm technology shows that 36× to 164× speedup is achieved when UpWB works under

the peak frequency 1.3 GHz (Section 5). Last but not least, our implementation codes are available on https://github.com/xiang-rc/UpWB_ref.

2 Background

In this section, we firstly introduce the relevant algorithmic background along with the analysis of main operations. Then, we identify the design challenges that are specific for a unified and efficient architecture supporting NSPN-based WBCs.

2.1 Notations

Let $\mathrm{GF}(2^m)$ denote the binary extension Galois field, whose element is in form of bit strings taken from the set $\{0,1\}^m$. Define the quotient ring of polynomials as $\mathrm{GF}(2)[x]/f(x)$, where $f(x)$ is a suitable irreducible polynomial and $f(x) \in \mathrm{GF}(2)[x]$. A polynomial $\boldsymbol{a} = a(x) = a_0 + a_1 x + ... + a_{m-1}x^{m-1} \in \mathrm{GF}(2)[x]$ can be denoted as $(a_0, a_1, ..., a_{m-2}, a_{m-1})$. Let A_i represent the i-th w-bit chunk of $\boldsymbol{a}$ for $0 \leq i < \lceil \frac{m}{w} \rceil = W$. Let $A_i[j]$ denote the j-th bit of A_i, where $j = 0, 1, ..., w-1$. Constants over $\mathrm{GF}(2^m)$ are represented in Hexadecimal form. The addition over $\mathrm{GF}(2^m)$ is the same as XOR operation $\oplus$.

2.2 Algorithmic Overview and Operations

The nested SPN-based DWBC takes in n-bit data block and k-bit secret key to perform encryption and decryption. The nested structure is embodied by the n_s-bit substitution box, which is a small-sized SPN-type cipher as well. For WBC-8/16/32 ($n_s = 8/16/32$), n and k are both determined as 128 bits. For WBC-24 ($n_s = 24$), n and k are both scaled to 120 bits. From the implementation point of perspective, NSPN-based WBCs share common data-flow structure. Readers could refer to the original paper for more specifications.

Table 2: The summarized parameter sets for DWBCs.

Scheme	State size	R_i	R_o	Type	Involutory	Finite field	Elements of the first row
S-8	16×8	64	10	H	✓	$\mathrm{GF}(2^8)$/ 0x11b	SM_8=[8, 16, 8a, 1, 70, 8d, 24, 76, a8, 91, ad, 48, 5, b5, af, f8]
S-16	8×16	32	10	H	✓	$\mathrm{GF}(2^{16})$/ 0x1002b	SM_{16}=[1, 3, 4, 5, 6, 8, b, 7]
S-24	5×24	20	10	C	×	$\mathrm{GF}(2^{24})$/ 0x100001b	SM_{24}=[1, 2, 5, 3, 4] InvSM_{24}=[bb3217, 6f80ec, 37285e, 33c9b4, d05310]
S-32	4×32	16	10	H	✓	$\mathrm{GF}(2^{32})$/ 0x10000008c	SM_{32}=[1, 2, 4, 6]
Y-16	8×16	32	8	H	×	$\mathrm{GF}(2^4)$/ 0x10011	YM_8=[5, 4, a, 6, 2, d, 8, 3] InvYM_8=[7, 3, e, b, 8, 1, 6, b]
Y-32	4×32	16	16	C	×	$\mathrm{GF}(2^4)$/ 0x10011	YM_4=[2, 3, 1, 1] InvYM_4=[e, b, d, 9]
W-16	8×16	24	7	D	—	$\mathrm{GF}(2^{16})$/ Config.	WM_{16}=[$a_0, a_1, ..., a_7$] InvWM_{16}=[$a'_0, a'_1, ..., a'_7$]

* NOTES: Involutory denotes Involuntary property, which means the inverse version of matrix is identical to itself. H: Hadamard. C: Circulant. D: Dynamic.

State. The state in SPNbox, Yoroi and WARX is organized in form of row vector containing $t = n/n_s$ elements: $X = (X_0, X_1, ..., X_{t-1})$. Here each n_s-bit element can be further divided into $l = n_s/8$ bytes: $X_i = (X_{i,0}, ..., X_{i,l-1})$.

Data structure: To make full use of the inherent data-level parallelism of block cipher, the round-based implementation strategy is adopted in our architecture [UHM+20], which means $n = t \times n_s$-bit data block is processed per cycle.

Key Schedule. The round keys are generated by a key derivation function (KDF), where a k-bit master key is expanded to $(R_i + 1)$ round keys, namely $(k_0, k_1, ..., k_{R_i}) = KDF(k, n_s \cdot (R_i + 1))$. Here R_i denotes the number of rounds for internal SPN cipher. KDF is specified as the hash function SHAKE-128 [Dwo15]. Table 2 summarizes the parameter set for related NSPN block ciphers.

Round Function. Since the round function for encryption is exactly opposite to that of decryption, we just introduce the former one here. As Figure 2 depicts, the overall structure of round function consists of three components, namely non-linear substitution box γ, linear layer θ and affine layer σ. The final iterative result is obtained by applying R_o rounds of round functions to the state: $X^{R_o} = (\bigcirc_{r=1}^{R_o}(\sigma^r \circ \theta \circ \gamma))(X^0)$ (for SPNbox/WARX). In this paper, the θ and σ layer are fused as the linear transformation (LT) loop, while γ as the non-linear transformation (NLT) loop from a computation point of view. (De-)multiplexers are mainly required to support different execution orders and data transmission. **(1) Non-linear Layer.** The γ layer maps $t \times n_s$-bit input to the $t \times n_s$-bit output using the substitution box: $(X_0, X_1, ..., X_{t-1}) \mapsto (\gamma_0(X_0), \gamma_1(X_1), ..., \gamma_{t-1}(X_{t-1}))$. As Figure 2 shows, the instantiations of γ layer for SPNbox/Yoroi and WARX are different from each other. **(2) Linear Layer.** The linear diffusion layer is specified as the multiplication between MDS matrix and state vector: $(X_0, X_1, ..., X_{t-1}) \mapsto (X_0, X_1, ..., X_{t-1}) \times$ MDS matrix. Table 2 summarizes the parameter sets for MDS matrices and their inverse versions, which contain different sizes, finite fields and types. As a special case, WARX adopts random matrices. **(3) Affine Layer.** For SPNbox and WARX, the affine layer σ^r is unified by adding round-dependent constants to the state: $(X_0, X_1, ..., X_{t-1}) \mapsto (X_0 \oplus C_0^r, X_1 \oplus C_1^r, ..., X_{t-1} \oplus C_{t-1}^r)$, where $C_i^r = (r-1) \cdot t + i + 1$ for $0 \le i \le t-1$. For Yoroi, the affine layer is defined as: $(X_0, X_1, ..., X_{t-1}) \mapsto (X_0 \oplus C^r, X_1 \oplus C^r, ..., X_{t-1} \oplus C^r)$, where $C^r = r + 1$. Obviously, vectorized modular multiplication (MM) and addition (MA) over GF(2^m) are required to support the iterative vector operation within the linear and affine layer. The right bottom of Figure 2 further presents the hierarchical operations for NSPN-structure DWBCs.

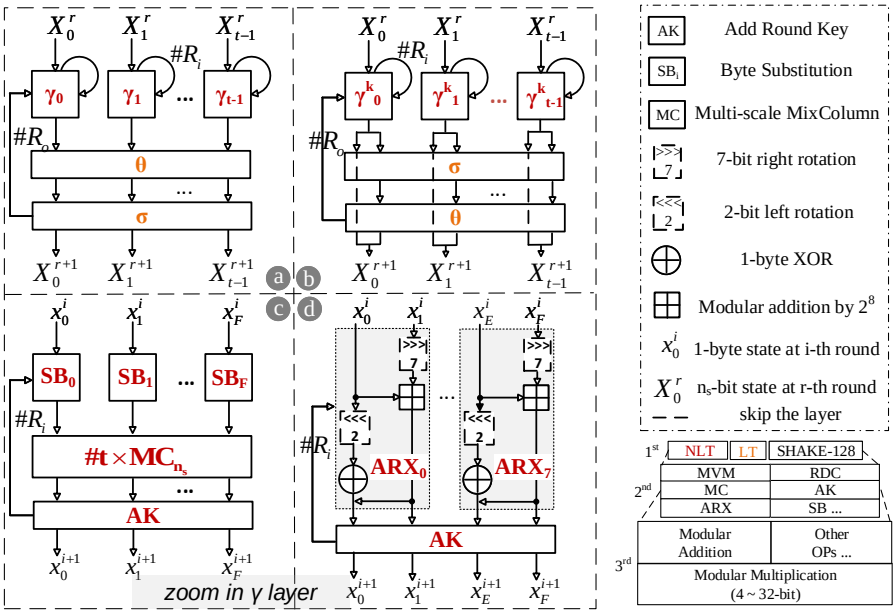


Figure 2: (a) Round function of SPNbox/WARX. (b) Round function of Yoroi. (c) The γ layer of SPNbox/Yoroi. (d) The γ layer of WARX.

2.3 Design Challenges

1) Nested loop dependency with large trip count. Figure 3 presents the difference of data dependency graph (DFG) between the conventional block cipher and NSPN-based WBCs. The conventional block cipher (e.g., AES) consists of the iteration of a round function, yielding a single loop. To improve the throughput by approximately x times, the effective method is to duplicate the round function by x times and then insert pipeline registers among them. Nevertheless, three data loops will be formed in the NSPN-based WBCs. First, the NLT loop needs to iterate M rounds of internal SPN function (γ layer). Then, the high-dimension MDS matrix-vector multiplication is usually decomposed into iterative scalar-vector operations [SKOP15], which leads to the second LT loop. Finally, the round count of external SPN structure is determined as R, which results in the third loop nested with the former two sibling loops. *Obviously, the inner-and-inter loop dependency renders the traditional strategy impractical for the NSPN-based WBCs.* Specifically, the inter-loop dependency leads to pipeline stall and incurs extra usage of FIFO. Since the trip count is hundreds of magnitude and varies with security parameters, fully unrolling the nested data loop not only entails tremendous resource overhead but also results in low resource utilization. Prior works aim to improve the mapping efficiency of nested loop on coarse-grained reconfigurable architectures (CGRA), but most of them are applicable for inner loops with small trip counts and limited by the fixed size of PE array [YLLW16][LLM+21] [KTM+18] [LYLW16]. Thus, we further leverage the specific information of WBC to attain case-oriented design strategies and gained performance.

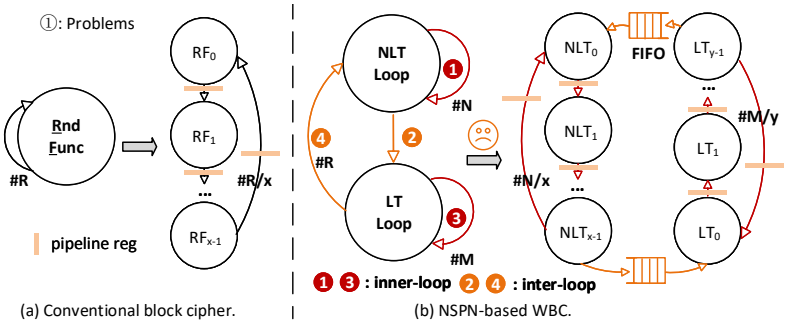


Figure 3: The comparison of DFG between conventional block cipher and WBC.

2) High-dimension and dynamic MDS matrices. As a kind of cryptographic primitive, the linear mapping commonly adopts MDS matrices to achieve diffusion in block cipher. However, there are four different points existing between conventional block ciphers and NSPN-based WBCs. (1) The dimension of MDS matrix applied in WBC is always larger than that of conventional block cipher, which tends to deteriorate the execution performance. For instance, SPNbox-8 applies 16×16-sized MDS matrix, whose dimension is four times as large as AES. (2) The sizes of related MDS matrices are diverse, scaling with different security levels. For example, SPNbox-family block ciphers contain four different sizes. (3) The elements of MDS matrix within WARX are dynamically generated rather than fixed constants. (4) Different types of MDS matrices (e.g., circulant or hadamard) are involved as Table 2 shows. A plethora of works are carried out to either search lightweight MDS matrices [AF14, SKOP15, LS16, LSL+19, VKS22] or optimize implementation of the prescribed matrix for low latency or low area footprint [LWF+22, XZL+20, VKS22]. This optimization can be transformed as the shortest linear program problem, which is deemed as NP-hard [BMP08]. *Nevertheless, a configurable and efficient hardware design for dynamic MDS matrices with different sizes is still left fewly explored by now.*

The value of implementing a configurable MVM architecture supporting random MDS matrix has been demonstrated in prior works. Besides the reduction of time for re-

fabrication, devices on the server side are always required to handle several responses from different client sides, which inevitably encounter cryptographic algorithms with diverse security parameters [LWD+18]. From a security perspective, some communication protocols even adopt more security-conservative parameters [WSH+10], which are likely to be periodically updated. Additionally, benefiting from the usage of random elements, dynamic matrix makes the statistic property of plaintext-ciphertext pairs hard to be predicted, which averts the linear and differential analysis to some degree. By randomly and uniformly distributing the hamming weight of elements, it tends to be more immerse towards side channel attacks, especially to the power related attacks [MN11].

3) Bit-precision reconfigurability without redundancy. As illustrated in section 2.2, DWBCs involve different bit-widths ($4 \sim 32$-bit), parameter sets and execution order, which easily result in complex routings and imbalanced workloads. As identified in [CLF+17] [CLK+16], it is challenging to devise a redundancy-free modular multiplier to support variable bit-width GF operations under different irreducible polynomials. Due to the polynomial modulo operation, a smaller GF bit-width multiplication cannot directly use a larger GF bit-width data-path by simply setting the most significant bits to zeros. In this work, besides the flexibility of precision and $f(x)$, one extra dimension of flexibility, namely the number of modular multiplier, is further required to handle multi-size MDS matrices. For example, to process SM_8 and SM_{16} in a round-based manner, we need #16×8-bit and #8×16-bit modular multipliers, respectively. *In a nutshell, a flexible vectorized modular multiplier has to be designed to support multi-precision data, multi-size matrices and different irreducible polynomials.*

3 Architectural Design Methodology

In this section, we firstly introduce the FTP mechanism, based on which the performance is profiled. Afterwards, we present the overall architecture of UpWB and describe the task distribution for each component.

3.1 Proposed FTP Mechanism

Exemplary FTP mechanism. Figure 4 (a) shows that the nested loop dependency results in low degree of parallelism if both NLT and LT are processed serially. A straightforward overlapped processing strategy is presented in Figure 4 (b). However, the gap of loop count between NLT and LT leads to idle time for synchronization, which lowers the resource utilization. Being aware that the loop count of NLT is assumed to be two times as large as that of LT, we could overlap the execution time of two NLT modules with that of one LT module to eliminate idle time. In other words, we partition NLT function into two sub-tasks to decouple the nested loop dependency, which avoids the usage of FIFO for synchronization [WNCY16] and efficiently increases the overall computation throughput. In this way, the operation throughput (TP) is raised by approximately $3\times$ but only incurs $1.5\times$ resource overhead (RO), which achieves better area-time efficiency than other methods like fully unrolling or direct duplication. The FTP mechanism is inspired by [WNCY16], which implements in-memory AES and faces similar data dependency.

Performance model. Based on the FTP mechanism, we attempt to decompose the N-round NLT and M-round LT into about $x\times$ (N/x)-round NLT_i and $y\times$ (M/y)-round LT_j, respectively. The uncoupled data-flow graph is shown in the right bottom of Figure 4. In terms of timing schedule, we try to overlap the data processing to hide much more latency. In the ideal case, all of the function modules are utilized to alternatively process

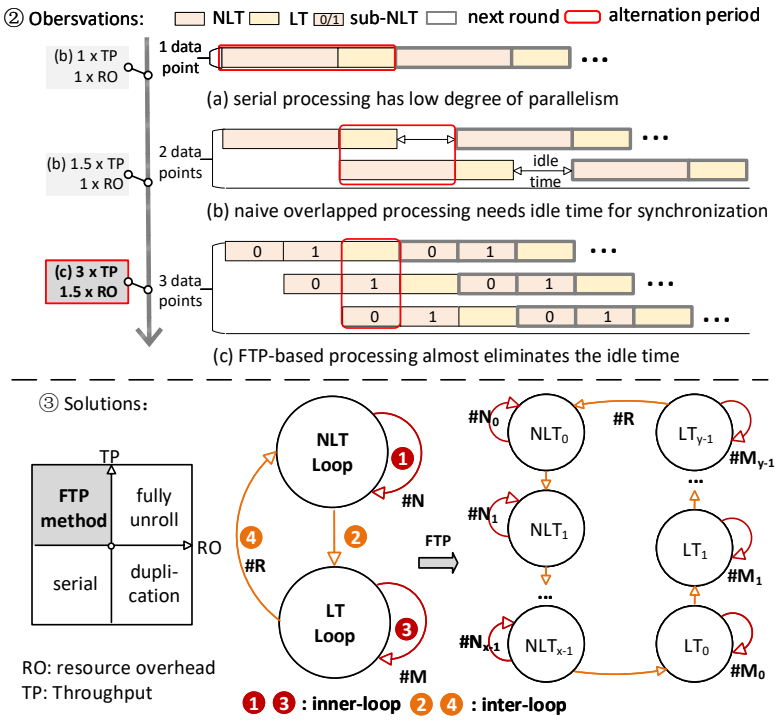


Figure 4: The exemplary FTP-based processing strategy.

$x + y$ data points. Then, the cycle count for black-box encryption is calculated as below:

$$CC = \overbrace{(max\{N_i, M_j\} + 1) \cdot (x + y) \cdot R_o}^{\text{periodical alternation time}} + \underbrace{\sum_{i=0}^{x-1}(N_i + 1) + \sum_{j=0}^{y-2}(M_j + 1) + \delta}_{\text{warm-up time}} \tag{1}$$

Here, $\delta = (max\{N_0, N_1\} - N_1) + ... + (max\{N_0, N_1, ..., N_{x-1}\} - N_{x-1}) + ... + (max\{N_0, N_1, ..., N_{x-1}, M_0, M_1, ..., M_{y-2}\} - M_{y-2})$, indicating the timing penalty for waiting the function module with the largest sub-round count. The largest sub-round count, namely $\max\{M_i, N_j\}$ for $0 \leq i \leq x-1$, $0 \leq j \leq y-1$, determines the period of alternation. The warm-up time represents the cycle count required to fill all of function modules with data points. Consequently, the computation throughput (TP) is formulated as below:

$$\textbf{minimize } TP = \frac{DW \cdot (x + y) \cdot f_{max}}{CC}, \textbf{ subject to } \sum_{i=0}^{x-1} N_i = N, \sum_{j=0}^{y-1} M_j = M.$$

DW denotes the bit-width of data and f_{max} denotes the peak frequency. Considering that $\max\{M_i, N_j\} \geq \frac{M+N}{x+y}$ (pigeonhole principle), the equality holds if and only if $M_0 = M_1 = \ldots = M_{x-1} = N_0 = N_1 = \ldots = N_{y-1} = \frac{M+N}{x+y}$. By plugging the equality condition into the above equation, the computation throughput will further satisfy the condition:

$$TP \leq \frac{DW \cdot (x + y) \cdot f_{max}}{(M + N + x + y) \cdot R_o + N + M + x + y - M_{y-1}}$$

Based on the above analysis, a useful design principle is to narrow down the gap between M_i and N_j, which reduces the amount of idle time for synchronizing the function modules

with different amortized round counts. Being aware of the round parameters of all involved block ciphers, we determine $x = 4$ and $y = 2$ to develop the NLT and LT cluster.

As a typical example, Figure 5 presents the timing diagram of overlapped processing for SPNbox-16. The round count for LT_j module is about twice as much as the NLT_i module. Thus, we align the operation time of two NLT modules with one LT module. Then, 4 data points will be alternatively processed and the TP is calculated as: $\frac{4 \cdot DW \cdot f_{max}}{688}$.

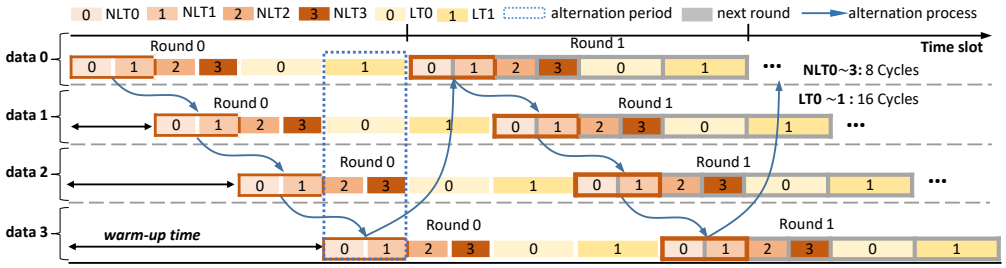


Figure 5: An example of FTP mechanism for SPNbox-16 black-box encryption.

3.2 Overall Hardware Architecture

Figure 6 presents the overall architecture for UpWB, which mainly consists of a top control module, KDF module, NLT and LT cluster. Ideally, UpWB will handle 6 data channels synchronized by 6 sequencers. To execute different types of algorithms, data blocks and configuration context are sent to the buffer at the initial phase, including the algorithm type, enc/decryption mode and parameter set. The KDF module is instantiated to the SHAKE-128 function, which affords to generates round keys for NLT cluster at the second phase. In our Keccak design, five permutation steps are conducted upon the 1600-bit state cube sequentially per cycle. At the third phase, the NLT cluster carries out the internal SPN function in a round-based manner, processing n-bit data block per cycle. The LT cluster conducts the MVM of different sizes. As the core arithmetic unit, an adaptive VMM is devised to handle random matrices based on a folded architecture.

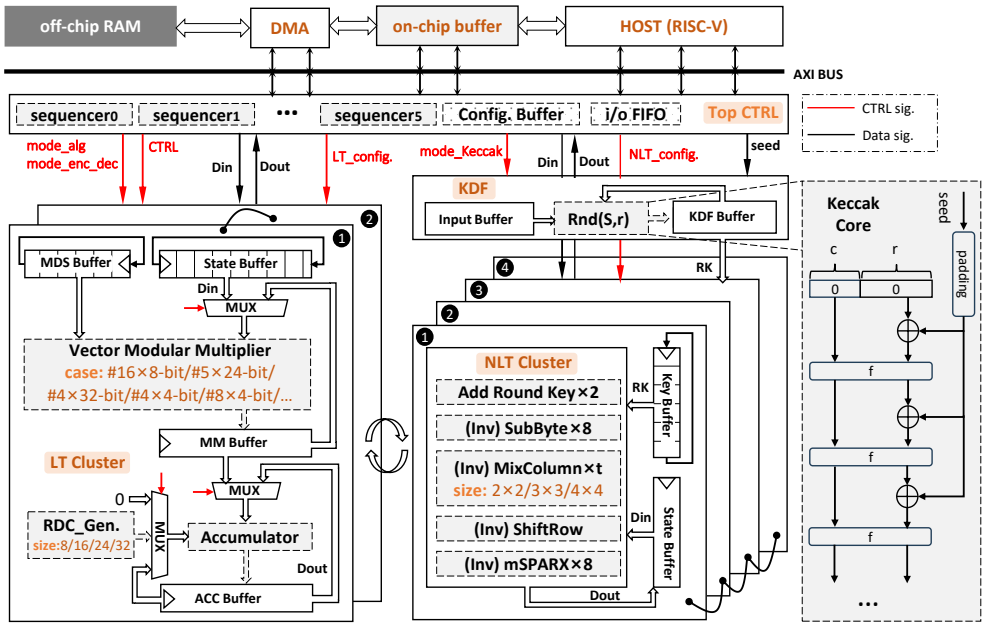


Figure 6: The proposed hardware architecture for nested SPN-type white-box block cipher.

4 Algorithm-hardware Co-design for Operators

4.1 Adaptive VMM

To obtain the precision-reconfigurability without redundancy, the new VMM is devised to meet two requirements: (1) The VMM should be able to process multi-precision variable elements under different irreducible polynomials without under-utilization of hardware resource. (2) A single modular multiplier with large q bit width can be exactly decomposed (vectorized) into n modular multipliers with small $\frac{q}{n}$ bit width.

Prior work on single MM. A scalable radix-2 Montgomery multiplication algorithm is proposed in [GTSK02] [KAJ96][JH07], which decomposes operand $a(x)$ into several chunks, namely $a(x) = \boldsymbol{a} = \sum_{i=0}^{W-1} A_i(x) \cdot x^{i \cdot w}$ where $A_i(x) = \sum_{j=0}^{w-1} a_{i \cdot w+j} \cdot x^j$, $W = \lceil \frac{m+1}{w} \rceil$. $b(x)$ is scanned one bit at a time while $a(x)$ is scanned serially per chunk. [RM13] modifies this algorithm by scanning multiple bits of $b(x)$ per cycle as shown in algorithm 1. The iterative operation within the third j-loop is divided into two types of computation blocks. The *parity* signal and C_0 are generated in block α. Based on *parity* signal, block β is responsible to compute the most significant bit (MSB) of C_0 and the remaining C_j for $1 \leq j \leq W-1$.

Algorithm 1 Scalable radix-2 Montgomery multiplication in [RM13]

Input: Let $a(x) = \boldsymbol{a} = (A_0, A_1, ..., A_{W-1})$ and $b(x) = \boldsymbol{b} = (B_0, B_1, ..., B_{W-1})$ be two polynomials over GF(2^m). Here $W = \lceil \frac{m+1}{w} \rceil$ and w is the bit-length of chunk. Similarly, $f(x) = \boldsymbol{f} = (F_0, F_1, ..., F_{W-1})$ is the irreducible polynomial.
Output: $c(x) = \boldsymbol{c} = (C_0, C_1, ..., C_{W-1}) = \boldsymbol{a} \cdot \boldsymbol{b} \cdot \boldsymbol{r^{-1}} \bmod \boldsymbol{f}$. Here $f(x) = x^m + r(x)$ and $r(x)^{-1} \cdot r(x) \bmod f(x) = 1$.

1: $\mathbf{c} \leftarrow (0, 0, ..., 0)$
2: **for** $i = 0$ to $W-1$ **do** ▷ `Travel each chunk index of` $b(x)$.
3: **for** $k = 0$ to $w-1$ **do** ▷ `Travel each bit within` $b(x)$ `chunk.`
4: `/* The computation task of block` α`.*/`
5: $C_0 = C_0 \oplus (B_i[k] \cdot A_0)$
6: $parity = C_0[0]$
7: $C_0 = C_0 \oplus (parity \cdot F_0)$
8: **for** $j = 1$ to $W-1$ **do** ▷ `Travel each chunk index of` $a(x)$ `and` $f(x)$.
9: `/* The computation task of block` β`.*/`
10: $C_j = C_j \oplus (B_i[k] \cdot A_j)$
11: $C_j = C_j \oplus (parity \cdot F_j)$
12: $C_{j-1} = \{C_j[0], C_{j-1}[w-1:1]\}$
13: **end for**
14: $C_{W-1} = \{0, C_{W-1}[w-1:1]\}$
15: **end for**
16: **end for**
17: **return** $\boldsymbol{c}$

Insight on the data-flow of systolic array. Figure 7 depicts the data-flow of systolic array proposed by [RM13]. The systolic architecture suffers from large pipeline latency in that block $\alpha_{i+1,j}$ ought to be conducted before block $\beta_{i,j+1}$. Taking $m = 5$, $w = 1$ as an example, Figure 7 presents the timing diagram for one-dimension array with #PE = 3, which takes 16 cycles to obtain the final result. However, the fact that *parity* signal of the first row can be broadcasted parallelly (1 cycle) rather than serially ($\approx m$ cycles) to the rest rows is not explored in prior work. In the following, we show that the cycle count can be considerably reduced due to this interesting observation.

Proposed VMM Design. Three optimization tricks in terms of area cost and latency are proposed to generate the adaptive radix-2 VMM shown in algorithm 2. ① Eliminating the redundant operations. Note that the MSB of $f(x)$ is fixed as 1. Indeed, it is sufficient to decompose the operand into $W = \lceil \frac{m}{w} \rceil$ chunks rather than $W = \lceil \frac{m+1}{w} \rceil$. *As a result,*

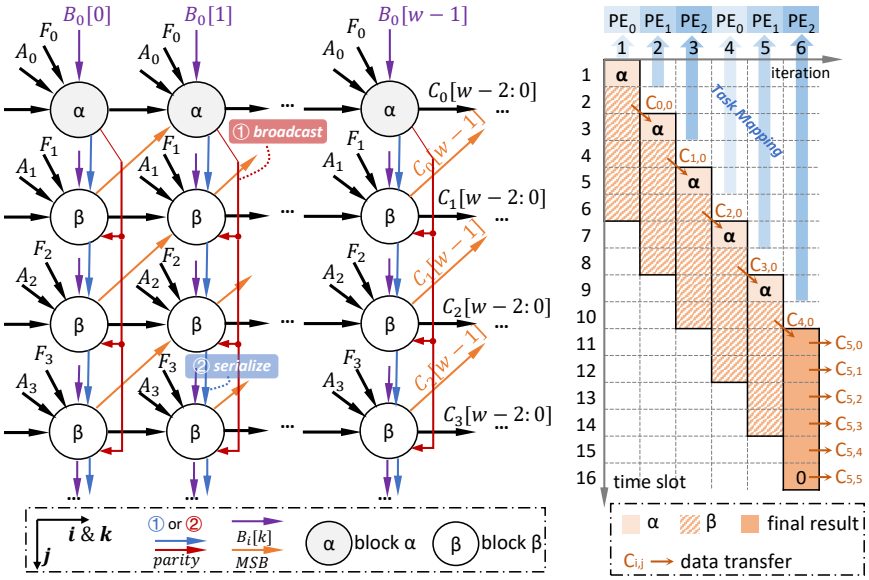
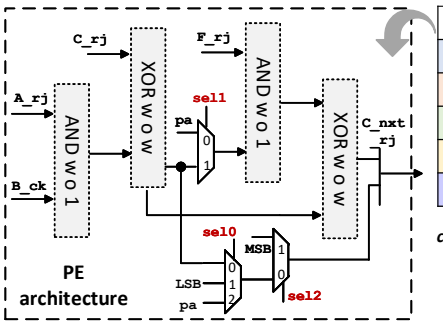


Figure 7: Analysis of data flow and timing diagram for systolic array.

the last iteration on index j can be eliminated when m is divisible by w. We make a proof as following. Let $W = \lceil \frac{m}{w} \rceil$ and $C_{i,j}$ denote the j-th chunk of C in the i-th iteration. For $j = W$, by taking $A_W = 0$ and $F_W = 1$, the last iteration is shown as below:

$$C_{i,W} = C_{i-1,W} \oplus B_i[k] \cdot A_W = C_{i-1,W} \oplus B_i[k] \cdot 0 = 0$$
$$C_{i,W} = C_{i,W} \oplus parity \cdot F_W = 0 \oplus parity \cdot 1 = parity$$
$$C_{i,W-1} = \{C_{i,W}[0], C_{i,W-1}[w-1:1]\} = \{parity, C_{i,W-1}[w-1:1]\}$$

Note that the MSB of C_{W-1} is *parity*, which can already be obtained in the $(j = 0)$-th iteration (Q.E.D.). The elimination of the last iteration will reduce the area cost by m PEs, which is further quantified by practical implementation in section 5.1. ② Unrolling the j-loop and folding the i&k-loop. Note that modular addition over GF(2^m) can naturally avoid the carry chain. Thus, we would like to remark that the pipeline registers used in [RM13] are actually redundant, i.e., have no impact on the critical path. The data-flow in the j-loop direction could be unrolled and processed parallelly. In contrast, the data-flow in the i/k-loop direction should be processed serially. Thus, we replace the systolic array with loop-based folded structure to avoid the unnecessary pipeline registers and reduce approximately $m\times$ cycles. ③ Vectorizing the modular multiplier to achieve adaptivity.



Functions	sel0_rj	sel1_rj	sel2_rj
Block A	2'b00	1'b1	1'b0
Block B	2'b01	1'b1	1'b0
Block C	2'b10	1'b0	1'b0
Block D	2'b01	1'b0	1'b0
Block E	2'b00	1'b1	1'b1

critical path = 2 ANDs +2 XORs + 1 MUX

`-rj:` the signal at the j-th row.
`-ck:` the signal at the k-th column.
`pa:` parity
`C_nxt_rj:` chunk of output, feeding to next PE at the j-th row.

Figure 8: Proposed configurable PE design.

Prior arts [GTSK02, RM13, SC06] solely focus on the optimization of a single modular

multiplication for ECC based cryptography, where the value of m is typically up to hundreds of bits. However, the matrix-vector multiplication in the linear mapping involves vector (multiple) modular multiplications. We point out that the almost redundancy-free vectorization can be achieved by configuring the source of the MSB for each output of PE. As shown in algorithm 2, the iterative computation consists of four types of blocks by now, which is further extended to support the configuration for the number of modular multipliers. Besides, bm signal is introduced to change the bit width of block, so that MM with data width less than w-bit can also be conducted.

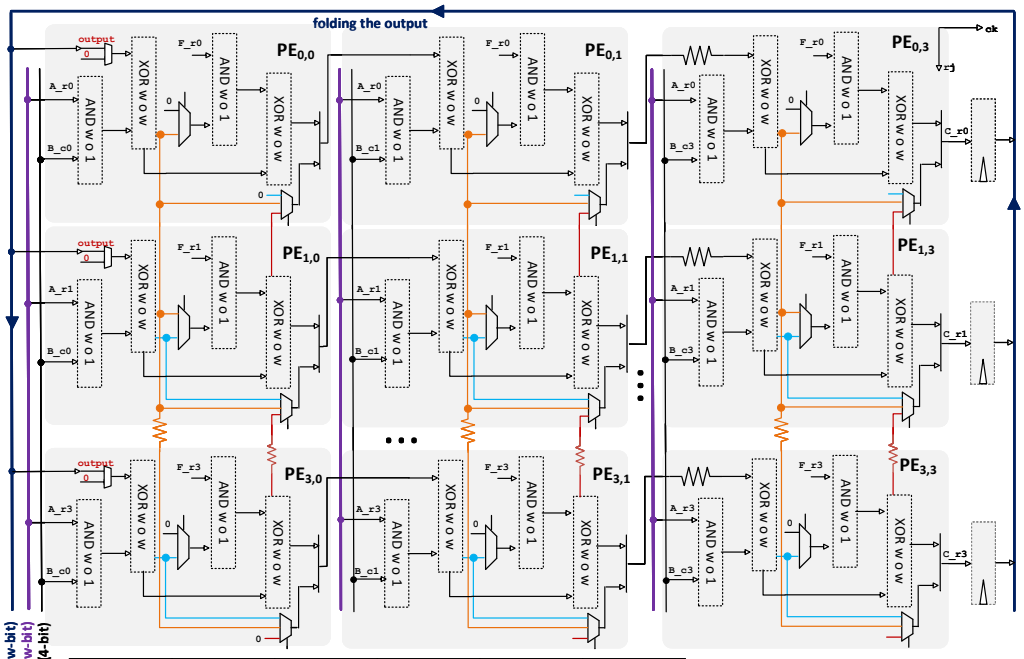
Algorithm 2 Proposed vectorized Montgomery multiplication

Input: Let $a(x) = \boldsymbol{a} = (A_0, A_1, ..., A_{W-1})$ and $b(x) = \boldsymbol{b} = (B_0, B_1, ..., B_{W-1})$ be two polynomials over GF(2^m). Here $W = \lceil \frac{m}{w} \rceil$ and w is the bit-length of chunk. Differently, $f(x) = \boldsymbol{f} = (F_0, F_1, ..., F_{W-1})$ is the irreducible polynomial, whose most significant bit is fixed to 1 and hence neglected in the vector. Finally, bm is the bit-width mode of PE cell and $bm \leq w$.

Output: $c(x) = \boldsymbol{c} = (C_0, C_1, ..., C_{W-1}) = \boldsymbol{a} \cdot \boldsymbol{b} \cdot \boldsymbol{r^{-1}} \bmod \boldsymbol{f}$. Here $f(x) = x^m + r(x)$ and $r(x)^{-1} \cdot r(x) \bmod f(x) = 1$.

$\mathbf{c} \leftarrow (0, 0, ..., 0)$
for $i = 0$ to $W - 1$ **do** ▷ `Travel each chunk index of` $b(x)$.
  for $k = 0$ to $w - 1$ **do** ▷ `Travel each bit within` $b(x)$ `chunk.`
    `/* The computation task of block A and B. */`
    $C_0 = C_0 \oplus (B_i[k] \cdot A_0)$
    $parity = C_0[0]$
    $C_0 = C_0 \oplus (parity \cdot F_0)$
    `/* The computation task of block A. */`
    if $W \leq 1$ **then**
      $C_0 = \{(w - bm)'b0, parity, C_0[bm - 1 : 1]\}$
    else
      for $j = 1$ to $W - 1$ **do** ▷ `Travel each chunk index of` $a(x)$ `and` $f(x)$.
        `/* The computation task of block D and C. */`
        $C_j = C_j \oplus (B_i[k] \cdot A_j)$
        $C_j = C_j \oplus (parity \cdot F_j)$
        $C_{j-1} = \{C_j[0], C_{j-1}[w - 1 : 1]\}$
        `/* The computation task of block C. */`
        if $j = W - 1$ **then**
          $C_{j-1} = \{(w - bm)'b0, parity, C_{j-1}[bm - 1 : 1]\}$
        end if
      end for
    end if
  end for
end for
return $\boldsymbol{c}$

Exemplary VMM with redundancy-free configuration. Figure 8 presents the hardware circuit of processing element (PE), which can be configured as five types of blocks by asserting control signals `sel0` $\sim$ `sel2`. Figure 9 presents an instantiation of VMM along with on-the-fly configuration cases. For example, by setting {`sel0_rj,sel1_rj,sel2_rj`} = $\{2'b00, 1'b1, 1'b0\}$, PEs within each row are all configured as block A. Thus, #4 w-bit MMs are developed. If we set `sel2_rj` $= 1'b1$, the width of modular multiplier can be further halved, whereby #4 $(w/2)$-bit MMs are formed. The third case is configured as #2 $2 \cdot w$-bit MMs. Block B generates the *parity* signal and propagates it to block C at the same column. The MSB of output in block B comes from the LSB of output in block C, while the MSB of output in block C is set as the *parity* signal. It is worth mentioning that the presented VMM consumes constant connection cost (fan-in/outs of Multiplexer) even if the configuration cases are increased.



Config. Cases	#4 w-bit MMs	#4 (w/2)-bit MMs	#2 2w-bit MMs	#1 3w-bit MMs	#1 4w-bit MMs
row0	A	E	B	B	B
row1	A	E	C	D	D
row2	A	E	B	C	D
row3	A	E	C	X	C

Figure 9: An example of proposed VMM over GF(2^m).

4.2 Configurable MVM

Determining the data-flow structure. Figure 10 ❶ depicts the mapping strategy of row-major MVM, which multiplies each row of matrix with the column vector like the school-book order. A circuit structure for implementing row-major MVM is also depicted in the right part, which assumes that the critical path consists of an adder and multiplier (MAC). As a drawback, this circuit structure suffers from extra clock cycles for pipeline filling and emptying, which requires $2 \cdot n$ clock cycles to finish the entire MVM. Figure 10 ❷ presents the mapping strategy of column-major MVM, which multiplies an element of vector with a column of matrix, and then accumulates the vector product. Each element of vector is broadcasted to all channels through the fully-connected crossbar for performing parallel MAC operation. Benefiting from the data-level parallelism, it performs n MACs per cycle and only occupies n clock cycles to complete MVM. Although it matches well with the round-based implementation, the fully-connected communication entails cumbersome MUXs, which becomes even more prominent as the number of channel increases. Figure 10 ❸ describes the mapping strategy of diagonal-major MVM, which multiplies the rotated state vector with each diagonal of matrix and then accumulates the vector product in sequence. The hardware circuit of diagonal-major MVM consumes n clock cycles to perform MVM, while the crossbar is replaced by the lightweight shift register. Considering the comprehensive metrics of area, flexibility and performance, we employ the diagonal-major strategy to circumvent the ponderous fan-out of broadcast and achieve data-level parallelism in the meantime.

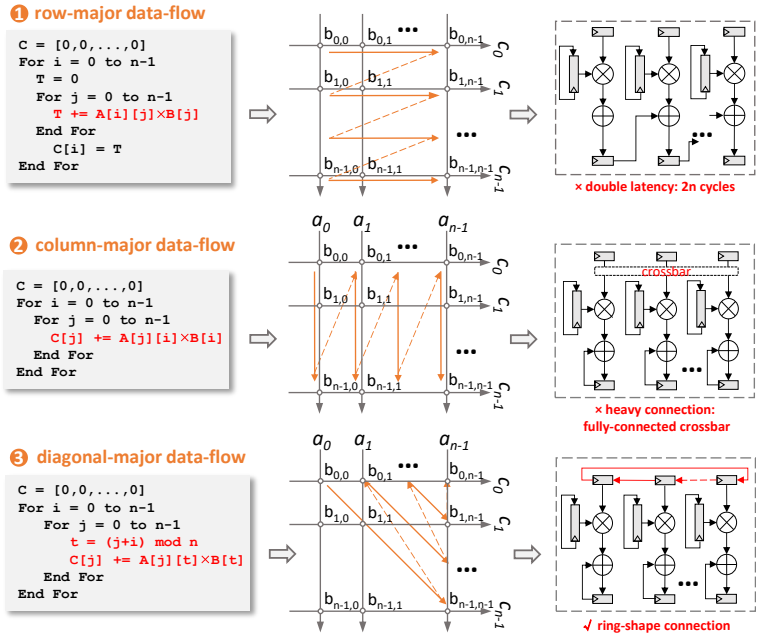


Figure 10: Comparison of data-flow structure for MVM.

Determining the size of PE array. As mentioned in section 4.1, the critical path of VMM is composed of m PEs, which could be further folded to x PEs. Although the folding mechanism can reduce the area cost and improve the frequency, it also brings the price of much more consumption of clock cycles. Thus, the overall latency is hard to be

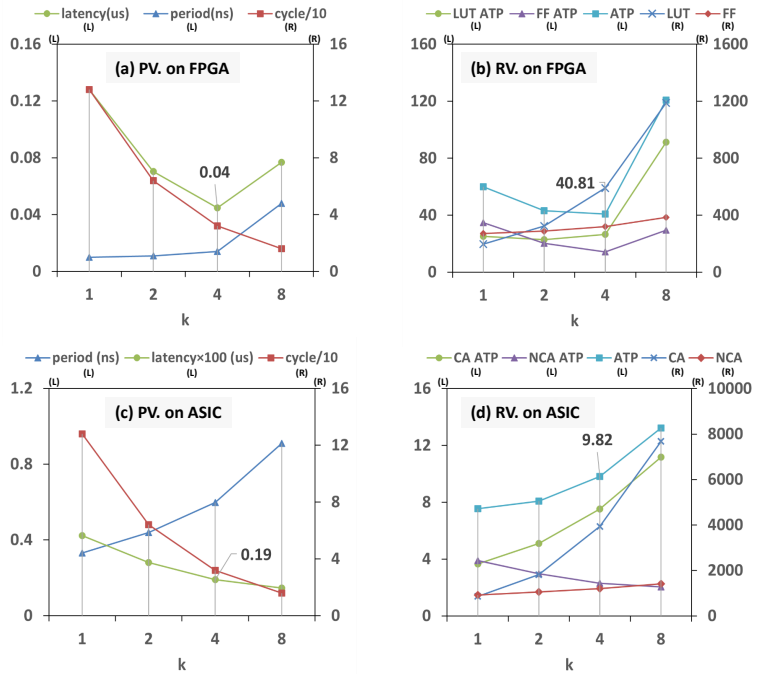
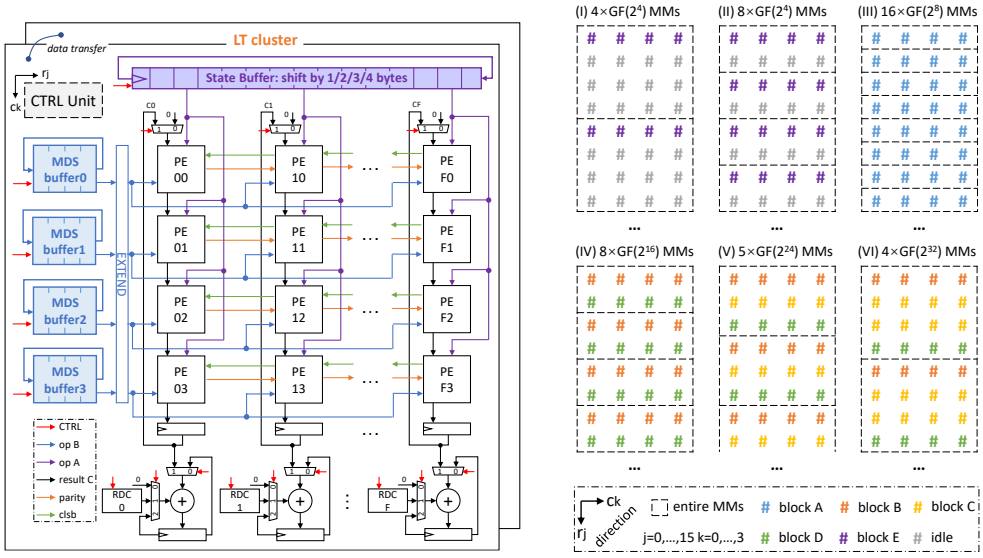


Figure 11: The relationship between k and area efficiency. CA: combinational logic area. NCA: non-combinational logic area. ATP: area time product.

theoretically quantified, which motivates us to make practical experiments. Figure 11 (a)

presents the performance variation (PV.) on FPGA when increasing the number of PE (k) in the ck direction. The right coordinate depicts the cycle counts. Figure 11 (b) depicts the resource variation (RV.) with the increment of k. The right coordinate indicates the value of LUT and Flip-flop (FF). The left coordinate reflects the variation of ATP with k. It is observed that the best area-time efficiency on FPGA is obtained by setting $k = 4$. Figure 11 (c) (d) also depict how the overall latency and ATP vary with the folding degree under ASIC evaluation.

Architecture of LT cluster. As shown in Figure 12 (a), operand A is the rotated state vector coming from the shift buffer while operand B is 64-bit element of MDS matrix fetched from 4 MDS buffers. As the core computation module of entire MVM, the PE array can be configured as different number of modular multipliers with variable data width. Figure 12 (b) shows six configuration cases for PE array. For example, the PE array is configured as 4 MMs over GF(2^4) in case I, which affords to perform 4×4 MVM for Yoroi-32. Additionally, the diagonal-wise memory layout of Hadmard MDS matrix can be reduced by 75% through eliminating the repetitive storage.



(a) Proposed MDS matrix vector multiplier. (b) Configuration cases for PE array.

Figure 12: The overall hardware architecture for LT cluster.

4.3 United (Inv)MixColumns

For SPNbox and Yoroi, multi-scale (inv)MixColumns are applied to construct AES variants with different sizes. For the first time, we present a unified design of multi-scale (inv)MixColumns through sharing sub-expressions and customizing the constant modular multipliers (CMMs).

Multi-scale MixColumns. Figure 13 presents the implementation scheme for multi-scale MixColumns (MC_{16}, MC_{24}, MC_{32}). Inspired by [MLH+20], the core idea is to share as much logic resource as possible to eliminate the redundant operations. Note that MC_{16} and MC_{24} are actually 2×2 and 3×3-sized sub-matrices of MC_{32}. As a result, the first insight is to reuse partial products/sums of MC_{16} to compute MC_{32}, which can be trivially achieved due to gcm(2,4) = 4. The second insight is to reuse partial products/sums of MC_{16} and MC_{32} to compute MC_{24}. Since the size of MC_{24} is prime with those of MC_{16} and MC_{32} (gcd (2,3,4) = 1), the calculation of MC_{24} is relatively non-trivial. However, by dividing index i into even and odd cases, MC_{24} can still be obtained just by the re-usage

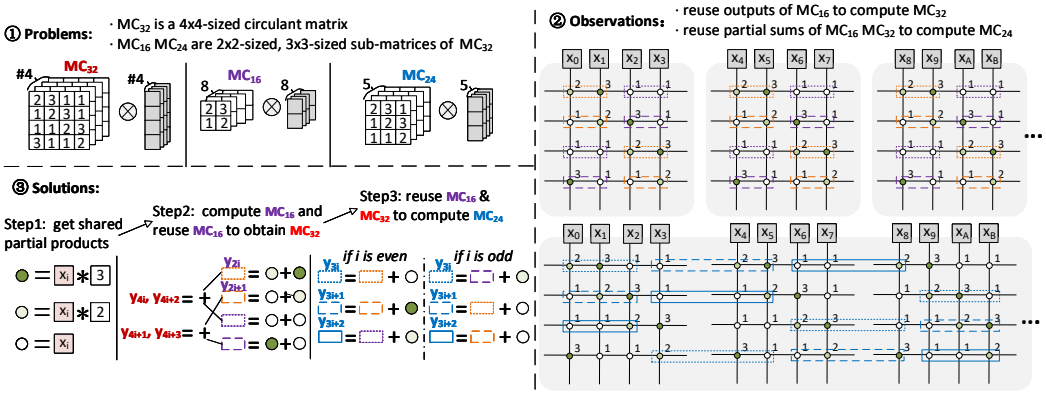


Figure 13: The implementation scheme for multi-scale MixColumns.

of immediate results (shown in step3). The final solution is summarized at the bottom of Figure 13. Ultimately, 49 (60.5%) MMs and 31 (33%) modular adders (MAs) are saved when compared to the direct method (DM) as shown in Figure 15. The implementation details are also shown in appendix A.

Multi-scale InvMixColumns. Figure 14 presents the implementation scheme for multi-scale InvMixColumns (InvMC_{16}, InvMC_{24}, InvMC_{32}). The inverse matrices feature disjoint matrix elements with large hamming weight, which are not as regular as the forward ones. Here, we adopt multiplicative decomposition for InvMC_{32} so that MC_{32} is reused. For InvMC_{16} and InvMC_{24}, the additive matrix decomposition technique is proposed to allow much more sub-expression sharing. Note that the adder performing $68 \cdot x_{2i} + 68 \cdot x_{2i+1}$ can be reused to compute $68 \cdot x_{3k} + 68 \cdot x_{3k+1}$ when $2i = 3k$ for $0 \leq i \leq 7$, $0 \leq k \leq 4$, that is to say $i = k = 0$ or $i = 3, k = 2$. The final solution is summarized at the right part of Figure 14. As shown in Figure 15, we need 69 MAs and 81 MMs to unify the multi-scale InvMCs. Thus, 52% MAs and 13.8% MMs are saved when compared to DM. The concrete implementation details are also shown in appendix B.

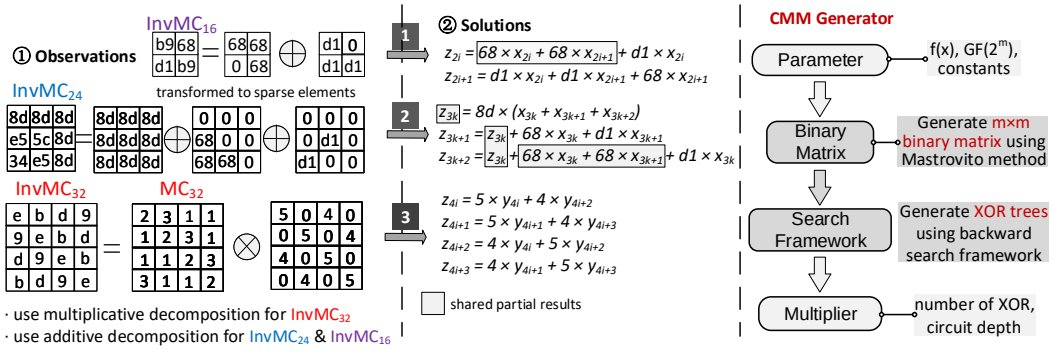


Figure 14: The implementation scheme for multi-scale InvMixColumns.

Design generator for CMMs. The rightmost part of Figure 14 presents the framework of design generator for CMM. Since the irreducible polynomial within (inv)MCs is fixed as $x^8 + x^4 + x^3 + x + 1$, the modular multiplications by constants with large hamming weight are customized as Mastrovito's multiplier [Mas88] (detailed in appendix C). At the first step, we generate the binary matrix based on input parameter $f(x)$, $GF(2^m)$ and constant. Then, we leverage a backward search framework [LWF+22] to implement the binary matrix vector multiplication, which aims to consume as less XORs as possible while

guaranteeing a minimum circuit depth. We make CMMs about 68 and $d1$ as case study. As shown in Figure 15, when computing CMMs about 68 and $d1$ using DM, 26 and 29 XORs are needed. Based on the backward search framework, the area cost can be reduced to 18 XORs and 19 XORs with minimum circuit depth 3 XORs.

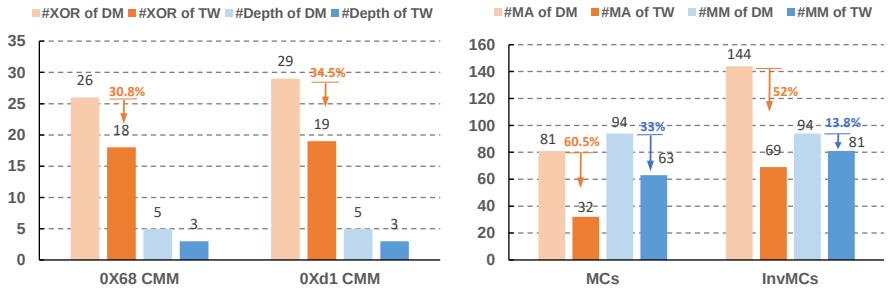


Figure 15: Evaluations on united (Inv)MCs and proposed CMMs.

5 Implementation Results and Comparisons

The coprocessor is designed and simulated with Verilog HDL, whose functional correctness is further verified with Python model. We obtain the implementation results based on TSMC 28nm ASIC synthesis and Xilinx Zynq UltraScale+ FPGA (xczu7ev-ffvc1156-3-e), respectively. We use Vivado 2020.2 to synthesize and implement the FPGA design. The synthesis result is also reported with Design Compiler P-2019.03. Since evaluations on the FTP mechanism and multi-scale (Inv)MCs modules are already presented in previous sections, we mainly make both theoretical and experimental performance evaluation on VMM, MVM and overall coprocessor in this section.

5.1 Alternative Approaches and Comparisons about VMM

Table 3 lists proposed alternative approaches for VMM, based on which we make comparison about area and time complexity. We assume that one m-bit operand is entirely processed and the other operand is scanned x-bit per cycle. For radix-2 LSB-first and MSB-first algorithms, the area cost and cycle count approximately amount to the proposed VMM. However, one important point to consider is that both LSB-first and MSB-first algorithms tend to consume proportionally increased fan-ins of MUXs ($\mathcal{O}(n)$) to support configuration, because the MSB of output inevitably comes from n non-adjacent PEs (detailed in appendix F). This crucial drawback causes the critical path and area to become larger when increasing configuration cases. The proposed VMM scheme, by contrast, consumes constant fan-outs of MUXs ($\mathcal{O}(1)$) which is independent of the configuration case. The cycle count of radix-2^w ($x = 2^w$) Montgomery algorithm [MÖPV04] is on par with the proposed scheme. Nevertheless, there are two drawbacks to *radix-x* Mont. algorithm. First, the area complexity is considerably higher than that of our scheme. Second, extra precomputation is needed. Similar shortcomings also exist in the *radix-x* LSB-First and MSB-First algorithms [KWP06]. Compared with the state-of-the-art radix-2 Mont. [RM13], our scheme reduces the area cost by $2 \cdot w \cdot x \cdot (A + X) + 2 \cdot x \cdot$ MUX(2:1) $+ w \cdot$ FFs while still maintaining the same critical path and even less clock cycles. Figure 16 further presents the comparison of practical implementation between this work (TW) and prior work (PW) [RM13] under the same parameter. It is observed that TW offers 1.2 (1.4) $\times$ area-efficiency measured by ATP under FPGA (ASIC) evaluation.

Table 3: The theoretical comparison of different algorithms about MMs.

Schemes	Cycles	Critical path	Flex. (n./f./w.)	Pre.	Conn.
radix-2 LSB-First	$\lceil m/x \rceil$	$(x+1)\cdot(A+X)+$ $x\cdot MUX(n:1)$	N/Y/Y	N	$\mathcal{O}(n)$
	$AC=(2x+1)\cdot m\cdot(A+X)+x\cdot W\cdot MUX(n:1)+m\cdot FF$				
radix-x LSB-First	$\lceil m/x \rceil$	$A+(2\cdot W+\lceil\log_2 x\rceil)\cdot X$ $+MUX(wn:w)$	N/Y/Y	Y	$\mathcal{O}(n)$
	$AC=2\cdot m\cdot w\cdot(A+X)-m\cdot X+W\cdot MUX(wn:w)+m\cdot FF$				
radix-2 MSB-First	$\lceil m/x \rceil$	$(2\cdot x-1)\cdot(A+X)-X+$ $(x-1)\cdot MUX(n:1)$	N/Y/Y	N	$\mathcal{O}(n)$
	$AC=m\cdot(2\cdot x-3)\cdot(A+X)+(x-2)\cdot W\cdot MUX(n:1)+m\cdot FF$				
radix-x MSB-First	$\lceil m/x \rceil$	$2\cdot A+(2\cdot W+2\cdot\lceil\log_2 x\rceil)\cdot X$ $+MUX(wn:w)$	N/Y/Y	Y	$\mathcal{O}(n)$
	$AC=2\cdot m\cdot w\cdot(A+X)-m\cdot X+W\cdot MUX(wn:w)+m\cdot FF$				
radix-x Mont.	$\lceil m/x \rceil$	$3\cdot(A+\lceil\log_2 x\rceil\cdot X)$ $+(W+4)\cdot X+MUX(wn:w)$	N/Y/Y	Y	$\mathcal{O}(1)$
	$AC=3\cdot w\cdot m\cdot(A+X)+2\cdot W\cdot(X+MUX(2w:w))$				
radix-2 Mont.[RM13]	$\lceil \frac{m+1}{x} \rceil$	$2\cdot x\cdot X+(x+1)\cdot A$ $+x\cdot MUX(2:1)$	N/Y/Y	N	$\mathcal{O}(1)$
	$AC=2\cdot(m+w)\cdot x\cdot(A+X)+(2\cdot W+2)\cdot x\cdot MUX(2:1)+(m+w)\cdot FF$				
Mastrovito's Mult.	1	$A+(\log_2 m)X$	N/N/N	Y	$\mathcal{O}(n)$
	$AC=m^2\cdot A+m\cdot(m-1)\cdot X$				
Karatsuba [PCHS20]	N/A	N/A	N/Y/N	Y	N/A
Poly. RNS [SSS12]	N/A	N/A	N/Y/N	Y	N/A
This Work	$\lceil m/x \rceil$	$2\cdot x\cdot X+(x+1)\cdot A+$ $x\cdot MUX(2:1)$	Y/Y/Y	N	$\mathcal{O}(1)$
	$AC=2\cdot x\cdot m\cdot(A+X)+2\cdot x\cdot W\cdot MUX(2:1)+m\cdot FF$				

* NOTES: AC: Area cost. Flex.: flexibility. Pre: precomputation. Conn.: connection cost. A: 2-input AND. X: 2-input XOR. MUX(m:n): m-to-n multiplexer. N: No. Y: Yes. $n./f./w.$: flexibility in terms of number of multiplier / irreducible polynomials / data width.

The aforementioned algorithms support MM under arbitrary irreducible polynomials. However, Mastrovito's multiplication cannot support the on-the-fly configuration. Some works leverage Karatsuba [PCHS20] or residue polynomial system [SSS12] to implement a single MM over large-sized GF(2^m). Obviously, their coarse-grained computation patterns lead to limited flexibility in terms of bit-width precision. On the contrary, the proposed VMM makes full advantage of the hybrid-grained radix-2 Mont. to achieve multi-dimension flexibility. Other works aim to further prune the circuit structure by using a special class of irreducible polynomials [FD05], like trinomials [BSF15], pentanomials [Ima18, XHM13] and all-one polynomials [SK99], but most of them cannot cater to the adaptivity.

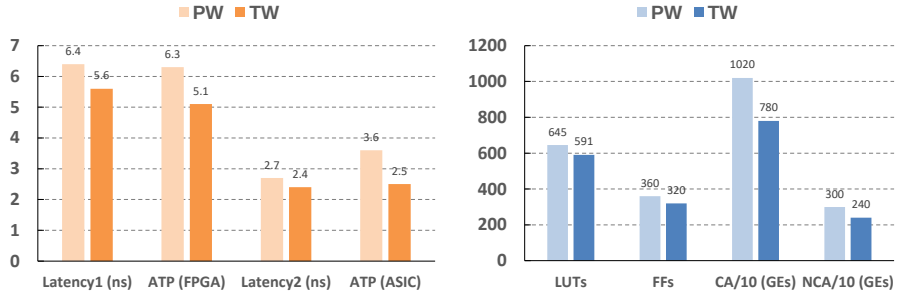


Figure 16: The practical comparison between TW and SOTA work (PW) [RM13].

5.2 Evaluations on MVM

Comparison with standalone designs. We implement the first row of MDS matrices following the method of [SKOP15]. In total, 9 individual implementations consume 2621 XORs. Additionally, [SKOP15][BKL16] ignore the extra overhead of MUXs incurred by reusing the multipliers of the first row. Indeed, MUXs cause large area overhead for connections, especially for high-dimension matrix. Moreover, this method [SKOP15] requires pre-computation and cannot support dynamic configuration for MDS matrices of different sizes. Fortunately, our configurable method approximately consumes 1024 XORs, 1024 ANDs (AND:XOR=1:2.25 in 28nm process) and enjoys a sound balance between flexibility and area efficiency.

Sensitivity study on random MDS matrices. To quantify the area variation for individually implementing random MDS matrices generated by different scalars ($1 \sim 20$), Figure 17 showcases $\mathsf{SM}_8 \sim \mathsf{InvSM}_{24}$ following typical method of [SKOP15]. As the comparison object, the red broken line denotes the area cost (≈ 2048 XORs) of the configurable MVM proposed in this work (TW). We make two important observations as following. First, multiplying each MDS matrix with different scalars yields varying area consumption, which may be even lower than that of the original one. Second, the area accumulation of $2 \sim 9$ generated matrices already overtakes that of TW, which reveals the profit of our configurable design.

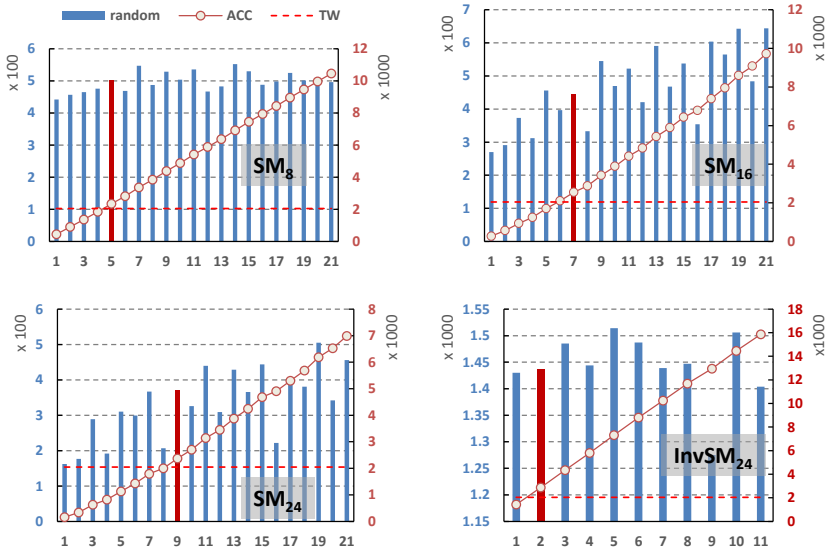


Figure 17: The area distribution and accumulation (ACC) for random MDS matrices generated by different scalars.

5.3 Evaluations on UpWB

Timing Evaluation. The optimized software counterparts are made as the baseline for comparison of computation throughput. The highest frequency of UpWB is up to 1.3 GHz under 28nm synthesis and 240 MHz under FPGA implementation with performance-optimized strategy, respectively. The software implementations of DWBCs are written in C code and run on a laptop computer equipped with 3.2 GHz Intel i7 CPU. A high-performance C++ library Givaro is applied to implement the MM over GF(2^m). Both AES-NI and AVX2 instruction sets are utilized to speed up the AES variants. The software performance is measured by taking the average over 100000 repetitions, each

time encrypting a random message of 2048 bytes. Table 4 offers the evaluation of TP for both software and hardware implementation. SW-TP is calculated as: (F × 8) / CPB (bps) (F denotes the peak frequency, CPB denotes cycle per byte). HW-TP1/2 is calculated following the equation 1. As can be seen, SPNbox-32 possesses the largest TP for black-box en/decryption. Since small-scaled MDS matrix is adopted by the Yoroi-family block ciphers, the obtained TP turns out to be the second largest. As shown in Figure 18, synthesized under TSMC 28nm technology, the TP of UpWB offers approximately 36× speed-up versus 164× software implementations. The FPGA design achieves 7× to 30× speed-up over software counterparts as well.

Table 4: Performance evaluation of UpWB on different platforms.

Schemes	DPs	Cycles	Latency (ns)	HW-TP1 (Mbps)	HW-TP2 (Mbps)	SW-CPB	SW-TP (Mbps)
S-8(Enc.)	6	1040	800.8	960.0	177.2	2102	12.2
S-8(Dec.)	6	1100	847	907.6	167.6	2105	12.2
S-16(Enc.)	4	688	529.76	967.4	178.6	2012	12.7
S-16(Dec.)	4	731	562.87	910.5	168.1	2010	12.7
S-24(Enc.)	4	255	196.35	2610.2	481.9	1609	15.9
S-24(Dec.)	4	688	529.76	967.4	178.6	2024	12.6
S-32(Enc.)	4	212	163.24	3139.6	579.6	1032	24.8
S-32(Dec.)	4	215	165.55	3095.8	571.5	1125	22.8
Y-16(Enc.)	5	402	309.54	2069.7	382.1	1532	16.7
Y-16(Dec.)	5	410	315.7	2029.3	374.6	1526	16.8
Y-32(Enc.)	5	426	328.02	1953.1	360.6	1021	25.1
Y-32(Dec.)	5	450	346.5	1848.9	341.3	1023	25.0
W-16(Enc.)	4	488	375.76	1363.9	251.8	729	35.1
W-16(Dec.)	4	527	405.79	1263.0	233.2	725	35.3
Avg. Case	5	545	420	1720	318	1470	20

* DPs denote the number of data points processed in every batch. HW-TP1, HW-TP2 and SW-TP denote the TP evaluated under ASIC synthesis, FPGA implementation and software platform.

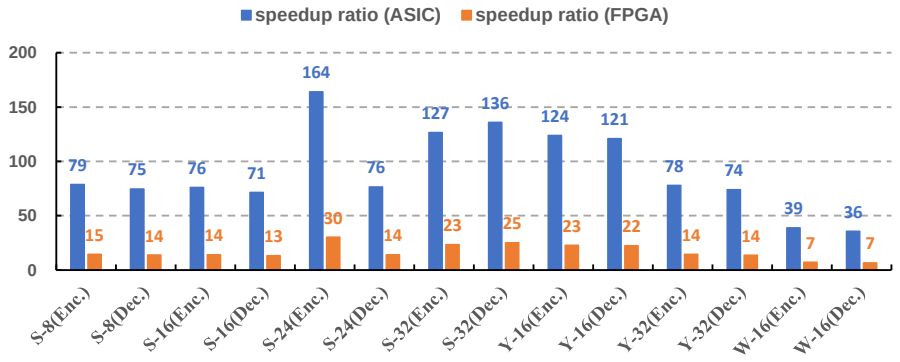


Figure 18: The speed-up ratio of ASIC (FPGA) versus software.

Area cost. Table 5 provides the detailed area breakdown for UpWB. As can be seen, each LT kernel only consumes 6975.4 μm^2, which could serve as a building block to be further integrated in many other processors. Thanks to the compact implementation, (inv)MixColumns only consume 895 LUTs (8.6 KGEs) and occupy about 24.8% (29.1%) within each NLT module. Each combined (Inv)S-box is also optimized with the state-of-the-art tower field technique to pursue high area efficiency [ME19].

Comparison with advanced works. As mentioned before, works on the hardware

Table 5: Area breakdown for UpWB under FPGA and ASIC evaluation.

Modules	FPGA eval.				ASIC eval.		
	LUTs	**Per.(%)**	**FFs**	**Per.(%)**	**Area(μm^2)**	**KGEs**	**Per.(%)**
SHAKE-128	8754	28.5	2537	52.7	31206.2	61.9	22.4
NLT_group	17832	58.1	1152	23.9	84594.7	167.8	60.7
⌊NLT0	3606	20.2	0	0	14954.2	29.7	17.7
⌊ARK	256	7.1	0	0	386	0.8	2.6
⌊(inv)MC	895	24.8	0	0	4359.4	8.6	29.1
⌊(inv)S-box	1280	35.5	0	0	7192	14.3	48.0
⌊(inv)ARX	232	6.4	0	0	532.2	1.1	3.6
⌊Other Parts	943	26.2	0	0	2484.6	4.9	16.6
⌊NLT1	4475	25.1	0	0	26416.8	52.4	31.2
⌊NLT2	3617	20.3	0	0	15068.9	29.9	17.8
⌊NLT3	4194	23.5	0	0	18112.2	35.9	21.4
⌊Other Parts	1940	10.9	1152	100.0	10042.6	19.9	11.9
LT_group	3387	11.0	992	20.6	13950.9	27.7	10.0
⌊PE array×2	1210	35.7	256	25.8	6067.1	12.0	43.5
⌊State Buffer×2	514	15.2	256	25.8	1744.3	3.5	12.5
⌊RDC×2	174	5.1	0	0.0	361.3	0.7	2.6
⌊Other Parts	1489	44.0	480	48.4	5778.2	11.5	41.4
Top Ctrl.	668	2.2	98	2.0	4988.6	9.9	3.6
Other Parts	30	0.1	36	0.7	4728.3	9.4	3.4
Total Area	30671	100.0	4815	100.0	139468.7	276.7	100.0
Avg. CE.	8.9K (13.1K)*bps/(LUTs+FFs)				6.2K (8.0K)*bps/GEs		

* The average throughput per area without consi dering KDF module (SHAKE-128).

architecture for WBC have yet been released. To demonstrate where the efficiency and flexibility of UpWB locate among similar works, we make comparison with advanced works targeting conventional block ciphers (BCs), which mainly involve congeneric operations over binary Galois filed. The key implementation results of both dedicated and configurable hardware designs are listed in Table 6 after approximate process normalization. The round count of WBCs is naturally larger than that of the conventional BCs. To make reasonable comparisons, we also normalize the computation efficiency by multiplying it with the average round count (denoted as Norm. CE.). [UHM+20] presents a decent work for dedicated hardware design of AES with high throughput efficiency (measured by throughput per gate). UpWB has 2.2× lower Norm. CE. than that of [UHM+20] but supports much more heavy algorithms. Compared with work [WSH+10] about AES with configurable parameters, UpWB achieves 2.1× improvement of CE. and supports more algorithms. The possible reason is that [WSH+10] utilizes heavy look-up tables to achieve configuration for modular multiplication with different precisions and irreducible polynomials, which introduces redundant resource utilization. As for the designs pursuing low-power metric, [CLF+17][DLDN20] propose configurable processors for conventional BCs and asymmetric cryptography over binary Galois field, which are supplied with sound programmability but have about 2.3× lower CE. than this work. Finally, we highlight that the proposed MVM architecture can bring three dimensions of flexibility including size, $f(x)$ and bit-width, which are not completely obtained in other works.

Discussions. Prior works about GF(2^m) arithmetic mainly target at the conventional block ciphers, error correcting codes, elliptic curve cryptography (ECC) and recent post-quantum cryptography. [IG16] [PLM13] present single MM hardware designs for ECC, which achieve high throughput but have limited flexibility. [SC14] proposes high-

Table 6: Comparison of implementation efficiency between UpWB and related works.

Work	TC'20 [UHM+20]	TVLSI '10[WSH+10]	ISCA '17[CLF+17]	TVLSI '20[DLDN20]	**This Work**
Algorithm	AES	config. AES	block coding +AES+ECC	config. BCs	7 WBCs+AES +SHAKE-128
Process	NanGate 45nm	250nm	TSMC 28nm	55nm	TSMC 28nm
Freq.(Hz)	694M	66M	100M	110M	1.3G
Avg. Latency	15.84 ns	N/A	N/A	N/A	84 ns
Avg. Round	10	10	10	N/A	286
size/f(x)/width	N/N/N	N/Y/Y	N/Y/Y	N/Y/Y	**Y/Y/Y**
Area	16.4KGEs	200.5KGEs	11.7KGEs	1250KGEs/ $12.25mm^2$	276.7KGEs/ $0.14mm^2$
Power	511uW @100MHz	N/A	431uW @100MHz	34.7mW @116MHz	20.8mW @100MHz
Avg. TP (bps)	8G	844.9M	12.2M	265M	1.7G
Avg. CE. (bps/GEs)	512K	16.8K*	102K	212K	8K
Norm. CE. (bps×r/GEs)	5.1M	168K*	1M	N/A	2.3M

* Considering the process variation.

performance processor for conventional BCs, which obtains configurability mainly based on LUT. [HFW11] addresses AES and ECC on the same kernel by sharing control units and memory, but two individual data-paths are still needed. In terms of MVM, [MKAF11] implements the dynamic MixColumn by using dual-port BRAM on FPGA, which is limited to fixed size and $f(x)$. [WSH+10][LWD+18] propose highly flexible and efficient processors capable of processing MixColumns with diverse $f(x)$, but the optimization for large-scale MDS matrix is out of scope. Owing to the adaptivie method, the proposed VMM can further support broad schemes based on GF(2^m), like ECC with even larger bit-width ($\geq$ 100bit) operations. Additionally, if we only focus on multiple $f(x)$, the data-path of VMM can be further pruned by exploiting the sparsity of $f(x)$.

6 Conclusion

The enormous performance overhead is one of the obstacles for the wide application of WBC. This work takes effort to develop the first hardware accelerator for a series of NSPN-based WBCs under black-box setting. To improve the resource utilization, we propose an efficient FTP mechanism to considerably decouple the nested data dependency and hide much more latency. By adopting algorithm-hardware co-design, we achieve decent area-time efficiency for several kernels, including an adaptive VMM, a configurable MVM and a compact (Inv)MCs. The computation throughput of UpWB outperforms the optimized software counterparts to a large degree under both ASIC synthesis and FPGA platform. This work mainly serves as an academic project whose principal focus is to demonstrate the feasibility and profit to adopt hardware acceleration for the black-box implementation of DWBC. Future works could extend the UpWB to accelerate other feasible non-NSPN white-box block ciphers like Feistel-structure-based WhiteBlock[FKKM16] and FPL [KLLM20], since some prior works present the practicability to devise the domain-specific accelerator supporting different structures of traditional block ciphers.

Acknowledgement

This work is supported in part by the National Key R&D Program of China (Grant No. 2023YFB4403500), and in part by the National Natural Science Foundation of China (Grant No. 62274102, No. 62104129), and in part by the National Key R&D Program

of China (Grant No.2021YFB2701201), and in part by the National Natural Science Foundation of China (No. 62302285).

References

[AF14] Daniel Augot and Matthieu Finiasz. Direct construction of recursive MDS diffusion layers using shortened BCH codes. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2014.

[BAB+19] Estuardo Alpirez Bock, Alessandro Amadori, Joppe W. Bos, Chris Brzuska, and Wil Michiels. Doubly half-injective prgs for incompressible white-box cryptography. In Mitsuru Matsui, editor, *Topics in Cryptology - CT-RSA 2019 - The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4-8, 2019, Proceedings*, volume 11405 of *Lecture Notes in Computer Science*, pages 189–209. Springer, 2019.

[BABM20] Estuardo Alpirez Bock, Alessandro Amadori, Chris Brzuska, and Wil Michiels. On the security goals of white-box cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):327–357, 2020.

[BBF+20] Estuardo Alpirez Bock, Chris Brzuska, Marc Fischlin, Christian Janson, and Wil Michiels. Security reductions for white-box key-storage in mobile payments. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 221–252. Springer, 2020.

[BBL23] Estuardo Alpirez Bock, Chris Brzuska, and Russell W. F. Lai. On provable white-box security in the strong incompressibility model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(4):167–187, 2023.

[BI15] Andrey Bogdanov and Takanori Isobe. White-box cryptography revisited: Space-hard ciphers. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1058–1069. ACM, 2015.

[BIT16] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. Towards practical whitebox cryptography: Optimizing efficiency and space hardness. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 126–158, 2016.

[BKL16] Christof Beierle, Thorsten Kranz, and Gregor Leander. Lightweight multiplication in gf(2^n) with applications to MDS matrices. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 625–653. Springer, 2016.

[BMP08] Joan Boyar, Philip Matthews, and René Peralta. On the shortest linear straight-line program for computing linear forms. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, volume 5162 of *Lecture Notes in Computer Science*, pages 168–179. Springer, 2008.

[BP09] Joan Boyar and Rene Peralta. New logic minimization techniques with applications to cryptology. Cryptology ePrint Archive, Paper 2009/191, 2009. https://eprint.iacr.org/2009/191.

[BP17] Alex Biryukov and Léo Perrin. Symmetrically and asymmetrically hard cryptography. In Tsuyoshi Takagi and Thomas Peyrin, editors, *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 III*, volume 10626 of *Lecture Notes in Computer Science*, pages 417–445. Springer, 2017.

[BSF15] Siavash Bayat-Sarmadi and Mohammad Farmani. High-throughput low-complexity systolic montgomery multiplication over gf(2^m) based on trinomials. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(4):377–381, 2015.

[CEJvO02a] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer, 2002.

[CEJvO02b] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In Joan Feigenbaum, editor, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

[CG03] Pawel Chodowiec and Kris Gaj. Very compact FPGA implementation of the AES algorithm. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 319–333. Springer, 2003.

[CG22] Alex Charlès and Chloé Gravouil. Review of the white-box encodability of nist lightweight finalists. Cryptology ePrint Archive, Paper 2022/804, 2022. https://eprint.iacr.org/2022/804.

[CLF+17] Yajing Chen, Shengshuo Lu, Cheng Fu, David T. Blaauw, Ronald Dreslinski Jr., Trevor N. Mudge, and Hun-Seok Kim. A programmable galois field processor for the internet of things. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*, pages 55–68. ACM, 2017.

[CLK+16] Yajing Chen, Shengshuo Lu, Hun-Seok Kim, David Blaauw, Ronald G. Dreslinski, and Trevor Mudge. A low power software-defined-radio baseband

processor for the internet of things. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 40–51, 2016.

[DLDN20] Yiran Du, Wei Li, Zibin Dai, and Longmei Nan. Pvharray: An energy-efficient reconfigurable cryptographic logic array with intelligent mapping. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(5):1302–1315, 2020.

[DLPR13] Cécile Delerablée, Tancrède Lepoint, Pascal Paillier, and Matthieu Rivain. White-box security notions for symmetric encryption schemes. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2013.

[DLS+22] Srinivas Devadas, Simon Langowski, Nikola Samardzic, Sacha Servan-Schreiber, and Daniel Sánchez. Designing hardware for cryptography and cryptography for hardware. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1–4. ACM, 2022.

[Dwo15] Morris Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions, 2015-08-04 2015.

[FD05] Haining Fan and Yiqi Dai. Fast bit-parallel gf(2^n) multiplier for all trinomials. *IEEE Transactions on Computers*, 54(4):485–490, 2005.

[FKKM16] Pierre-Alain Fouque, Pierre Karpman, Paul Kirchner, and Brice Minaud. Efficient and provable white-box primitives. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 159–188, 2016.

[GSM15] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. Cache template attacks: Automating attacks on inclusive last-level caches. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pages 897–912. USENIX Association, 2015.

[GTSK02] Adnan Abdul-Aziz Gutub, Alexandre F. Tenca, Erkay Savas, and Çetin Kaya Koç. Scalable and unified hardware to compute montgomery inverse in gf(p) and GF(2). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 484–499. Springer, 2002.

[HFW11] Michael Hutter, Martin Feldhofer, and Johannes Wolkerstorfer. A cryptographic processor for low-resource devices: Canning ECDSA and AES like sardines. In Claudio A. Ardagna and Jianying Zhou, editors, *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication - 5th IFIP WG 11.2 International Workshop, WISTP 2011, Heraklion, Crete, Greece, June 1-3, 2011. Proceedings*, volume 6633 of *Lecture Notes in Computer Science*, pages 144–159. Springer, 2011.

[IG16] Atef Ibrahim and Fayez Gebali. Low power semi-systolic architectures for polynomial-basis multiplication over gf(2^m) using progressive multiplier reduction. *J. Signal Process. Syst.*, 82(3):331–343, 2016.

[Ima18] José L. Imaña. Reconfigurable implementation of gf(2^m) bit-parallel multipliers. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 893–896, 2018.

[JH07] Nan Jiang and David Harris. Parallelized radix-2 scalable montgomery multiplier. In *2007 IFIP International Conference on Very Large Scale Integration*, pages 146–150, 2007.

[KAJ96] Çetin Kaya Koç, Tolga Acar, and Burton S. Kaliski Jr. Analyzing and comparing montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996.

[KHZ17] Martin Kumm, Martin Hardieck, and Peter Zipf. Optimization of constant matrix multiplication with low power and high throughput. *IEEE Transactions on Computers*, 66(12):2072–2080, 2017.

[KI21] Yuji Koike and Takanori Isobe. Yoroi: Updatable whitebox cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):587–617, 2021.

[KLLM20] Jihoon Kwon, ByeongHak Lee, Jooyoung Lee, and Dukjae Moon. FPL: white-box secure block cipher using parallel table look-ups. In Stanislaw Jarecki, editor, *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings*, volume 12006 of *Lecture Notes in Computer Science*, pages 106–128. Springer, 2020.

[KTM+18] Manupa Karunaratne, Cheng Tan, Aditi Kulkarni Mohite, Tulika Mitra, and Li-Shiuan Peh. Dnestmap: mapping deeply-nested loops on ultra-low power cgras. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, pages 129:1–129:6. ACM, 2018.

[KWP06] Sandeep S. Kumar, Thomas J. Wollinger, and Christof Paar. Optimum digit serial gf(2^m) multipliers for curve-based cryptography. *IEEE Trans. Computers*, 55(10):1306–1311, 2006.

[LLM+21] Dajiang Liu, Ting Liu, Xingyu Mo, Jiaxing Shang, and Shouyi Yin. Polyhedral-based pipelining of imperfectly-nested loop for cgras. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021, Munich, Germany, November 1-4, 2021*, pages 1–9. IEEE, 2021.

[LRH+22] Jun Liu, Vincent Rijmen, Yupu Hu, Jie Chen, and Baocang Wang. WARX: efficient white-box block cipher based on ARX primitives and random MDS matrix. *Sci. China Inf. Sci.*, 65(3), 2022.

[LS16] Meicheng Liu and Siang Meng Sim. Lightweight MDS generalized circulant matrices. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 101–120. Springer, 2016.

[LSL+19] Shun Li, Siwei Sun, Chaoyun Li, Zihao Wei, and Lei Hu. Constructing low-latency involutory MDS matrices with lightweight circuits. *IACR Trans. Symmetric Cryptol.*, 2019(1):84–117, 2019.

[LWD+18] Leibo Liu, Bo Wang, Chenchen Deng, Min Zhu, Shouyi Yin, and Shaojun Wei. Anole: A highly efficient dynamically reconfigurable crypto-processor for symmetric-key algorithms. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 37(12):3081–3094, 2018.

[LWF+22] Qun Liu, Weijia Wang, Yanhong Fan, Lixuan Wu, Ling Sun, and Meiqin Wang. Towards low-latency implementation of linear layers. *IACR Trans. Symmetric Cryptol.*, 2022(1):158–182, 2022.

[LYLW16] Xinhan Lin, Shouyi Yin, Leibo Liu, and Shaojun Wei. Exploiting parallelism of imperfect nested loops with sibling inner loops on coarse-grained reconfigurable architectures. In *21st Asia and South Pacific Design Automation Conference, ASP-DAC 2016, Macao, Macao, January 25-28, 2016*, pages 456–461. IEEE, 2016.

[Mas88] Edoardo D. Mastrovito. VLSI designs for multiplication over finite fields $GF\ (2^{\mathrm{m}})$. In Teo Mora, editor, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 6th International Conference, AAECC-6, Rome, Italy, July 4-8, 1988, Proceedings*, volume 357 of *Lecture Notes in Computer Science*, pages 297–309. Springer, 1988.

[ME19] Alexander Maximov and Patrik Ekdahl. New circuit minimization techniques for smaller and faster AES sboxes. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(4):91–125, 2019.

[MKAF11] Ghulam Murtaza, Azhar Ali Khan, Syed Wasi Alam, and Aqeel Farooqi. Fortification of AES with dynamic mix-column transformation. *IACR Cryptol. ePrint Arch.*, page 184, 2011.

[MLH+20] Huiyu Mo, Leibo Liu, Wenjing Hu, Wenping Zhu, Qiang Li, Ang Li, Shouyi Yin, Jian Chen, Xiaowei Jiang, and Shaojun Wei. Tfe: Energy-efficient transferred filter-based engine to compress and accelerate convolutional neural networks. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 751–765, 2020.

[MN11] Muhammad Yasir Malik and Jong-Seon No. Dynamic MDS matrices for substantial cryptographic strength. *CoRR*, abs/1108.6302, 2011.

[MÖPV04] Nele Mentens, Siddika Berna Örs, Bart Preneel, and Joos Vandewalle. An FPGA implementation of a montgomery multiplier over gf(2^m). *Comput. Artif. Intell.*, 23(5):487–499, 2004.

[PCHS20] Sun-Mi Park, Ku-Young Chang, Dowon Hong, and Changho Seo. Space efficient gf(2^m) multiplier for special pentanomials based on n -term karatsuba algorithm. *IEEE Access*, 8:27342–27360, 2020.

[PLM13] Jeng-Shyang Pan, Chiou-Yng Lee, and Pramod Kumar Meher. Low-latency digit-serial and digit-parallel systolic multipliers for large binary extension fields. *IEEE Trans. Circuits Syst. I Regul. Pap.*, 60-I(12):3195–3204, 2013.

[RFS+19] Félix Carvalho Rodrigues, Hayato Fujii, Ana Clara Zoppi Serpa, Giuliano Sider, Ricardo Dahab, and Julio César López-Hernández. Fast white-box implementations of dedicated ciphers on the armv8 architecture. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, volume 11774 of *Lecture Notes in Computer Science*, pages 341–363. Springer, 2019.

[RM13] Guillaume Reymond and Victor Murillo. A hardware pipelined architecture of a scalable montgomery modular multiplier over gf(2^{m}). In *ReConFig 2013, Cancun, Mexico, December 9-11, 2013*, pages 1–6. IEEE, 2013.

[Sas18] Pascal Sasdrich. *Cryptographic hardware agility for physical protection.* PhD thesis, Ruhr University Bochum, Germany, 2018.

[SC06] Ravi Kumar Satzoda and Chip-Hong Chang. A fast kernel for unifying gf(p) and gf(2^{m}) montgomery multiplications in a scalable pipelined architecture. In *International Symposium on Circuits and Systems (ISCAS 2006), 21-24 May 2006, Island of Kos, Greece*. IEEE, 2006.

[SC14] Gokhan Sayilar and Derek Chiou. Cryptoraptor: high throughput reconfigurable cryptographic processor. In Yao-Wen Chang, editor, *The IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2014, San Jose, CA, USA, November 3-6, 2014*, pages 154–161. IEEE, 2014.

[SK99] B. Sunar and C.K. Koc. Mastrovito multiplier for all trinomials. *IEEE Transactions on Computers*, 48(5):522–527, 1999.

[SKOP15] Siang Meng Sim, Khoongming Khoo, Frédérique E. Oggier, and Thomas Peyrin. Lightweight MDS involution matrices. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 471–493. Springer, 2015.

[SMG16a] Pascal Sasdrich, Amir Moradi, and Tim Güneysu. White-box cryptography in the gray box - - A hardware implementation and its side channels -. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 185–203. Springer, 2016.

[SMG16b] Pascal Sasdrich, Amir Moradi, and Tim Güneysu. White-box cryptography in the gray box - - A hardware implementation and its side channels -. pages 185–203, 2016.

[SSS12] Dimitrios Schinianakis, Alexander Skavantzos, and Thanos Stouraitis. Gf(2n) montgomery multiplication using polynomial residue arithmetic. In *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 3033–3036, 2012.

[TGS+22] Yufeng Tang, Zheng Gong, Tao Sun, Jinhai Chen, and Zhe Liu. Wbmatrix: An optimized matrix library for white-box block cipher implementations. *IEEE Transactions on Computers*, 71(12):3375–3388, 2022.

[TI22] Yosuke Todo and Takanori Isobe. Hybrid code lifting on space-hard block ciphers application to yoroi and spnbox. *IACR Trans. Symmetric Cryptol.*, 2022(3):368–402, 2022.

[UHM+20] Rei Ueno, Naofumi Homma, Sumio Morioka, Noriyuki Miura, Kohei Matsuda, Makoto Nagata, Shivam Bhasin, Yves Mathieu, Tarik Graba, and Jean-Luc Danger. High throughput/gate AES hardware architectures based on datapath compression. *IEEE Trans. Computers*, 69(4):534–548, 2020.

[VKS22] Ayineedi Venkateswarlu, Abhishek Kesarwani, and Sumanta Sarkar. On the lower bound of cost of MDS matrices. *IACR Trans. Symmetric Cryptol.*, 2022(4):266–290, 2022.

[WNCY16] Yuhao Wang, Leibin Ni, Chip-Hong Chang, and Hao Yu. Dw-aes: A domain-wall nanowire-based aes for high throughput and energy-efficient data encryption in non-volatile memory. *IEEE Transactions on Information Forensics and Security*, 11(11):2426–2440, 2016.

[WSH+10] Mao-Yin Wang, Chih-Pin Su, Chia-Lung Horng, Cheng-Wen Wu, and Chih-Tsun Huang. Single- and multi-core configurable AES architectures for flexible security. *IEEE Trans. Very Large Scale Integr. Syst.*, 18(4):541–552, 2010.

[XHM13] Jiafeng Xie, Jian jun He, and Pramod Kumar Meher. Low latency systolic montgomery multiplier for finite field $gf(2^m)$ based on pentanomials. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(2):385–389, 2013.

[XZL+20] Zejun Xiang, Xiangyong Zeng, Da Lin, Zhenzhen Bao, and Shasha Zhang. Optimizing implementations of linear layers. *IACR Trans. Symmetric Cryptol.*, 2020(2):120–145, 2020.

[YLLW16] Shouyi Yin, Xinhan Lin, Leibo Liu, and Shaojun Wei. Exploiting parallelism of imperfect nested loops on coarse-grained reconfigurable architectures. *IEEE Transactions on Parallel and Distributed Systems*, 27(11):3199–3213, 2016.

[ZP01] Tong Zhang and Keshab K. Parhi. Systematic design of original and modified mastrovito multipliers for general irreducible polynomials. *IEEE Trans. Computers*, 50(7):734–749, 2001.

Appendix

A Multi-scale MixColumns

Step 1: Observing that the elements in MixColumns are all constants of 3,2,1, we firstly multiply each byte of the state x_i ($0 \leq i \leq 15$) by 3,2 respectively as input for subsequent additions. As a result, a total of 32 constant modular multipliers are generated as $m_{i,0} = 2 \cdot x_i$, $m_{i,1} = 3 \cdot x_i$, for $0 \leq i \leq 15$.

Step 2: Based on step 1, the MixColumn about $\mathsf{MC_{16}}$ is calculated as:

$$\begin{bmatrix} y_{2i}^{(16)} \\ y_{2i+1}^{(16)} \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} x_{2i} \\ x_{2i+1} \end{bmatrix} = \begin{bmatrix} 2 \cdot x_{2i} + 3 \cdot x_{2i+1} \\ x_{2i} + 2 \cdot x_{2i+1} \end{bmatrix} = \begin{bmatrix} m_{2i,0} + m_{2i+1,1} \\ x_{2i} + m_{2i+1,0} \end{bmatrix}$$

Thus, extra 16 modular adders (= 8-bit XORs) will be consumed to generate $y_{2i}^{(16)}$, $y_{2i+1}^{(16)}$ for $0 \leq i \leq 7$ at this step.

Step 3: Note that the addition nodes generated in step 2 can be totally reused to construct the MixColumn about $\mathsf{MC_{32}}$ as below:

$$\begin{bmatrix} y_{4i}^{(32)} \\ y_{4i+1}^{(32)} \\ y_{4i+2}^{(32)} \\ y_{4i+3}^{(32)} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} x_{4i} \\ x_{4i+1} \\ x_{4i+2} \\ x_{4i+3} \end{bmatrix} = \begin{bmatrix} 2 \cdot x_{4i} + 3 \cdot x_{4i+1} + x_{4i+2} + x_{4i+3} \\ x_{4i} + 2 \cdot x_{4i+1} + 3 \cdot x_{4i+2} + x_{4i+3} \\ x_{4i} + x_{4i+1} + 2 \cdot x_{4i+2} + 3 \cdot x_{4i+3} \\ 3 \cdot x_{4i} + x_{4i+1} + x_{4i+2} + 2 \cdot x_{4i+3} \end{bmatrix}$$

$$= \begin{bmatrix} y_{4i}^{(16)} + x_{4i+2} + x_{4i+3} \\ y_{4i+1}^{(16)} + 3 \cdot x_{4i+2} + x_{4i+3} \\ x_{4i} + x_{4i+1} + y_{4i+2}^{(16)} \\ 3 \cdot x_{4i} + x_{4i+1} + y_{4i+3}^{(16)} \end{bmatrix} = \begin{bmatrix} y_{4i}^{(16)} + x_{4i+2} + x_{4i+3} \\ y_{4i+1}^{(16)} + m_{4i+2,1} + x_{4i+3} \\ x_{4i} + x_{4i+1} + y_{4i+2}^{(16)} \\ m_{4i,1} + x_{4i+1} + y_{4i+3}^{(16)} \end{bmatrix}, \ 0 \le i \le 3$$

Then, we accumulate elements of each row in a tree shape to pursue low circuit depth. To do this, extra 16 modular adders (= 8-bit XORs) at the second layer are used to generate a_{4i}, a_{4i+1}, a_{4i+2} and a_{4i+3} as: $a_{4i} = x_{4i+2} + x_{4i+3}$, $a_{4i+1} = m_{4i+2,1} + x_{4i+3}$, $a_{4i+2} = x_{4i} + x_{4i+1}$ and $a_{4i+3} = m_{4i,1} + x_{4i+1}$. At last, the final results of MixColumn about $\mathsf{MC_{32}}$ are calculated as: $y_{4i}^{(32)} = y_{4i}^{(16)} + a_{4i}$, $y_{4i+1}^{(32)} = y_{4i+1}^{(16)} + a_{4i+1}$, $y_{4i+2}^{(32)} = y_{4i+2}^{(16)} + a_{4i+2}$ and $y_{4i+3}^{(32)} = y_{4i+3}^{(16)} + a_{4i+3}$. To sum up, a total of extra $32 \times 8 = 256$ XOR gates are required to generate $y_{4i}^{(32)}$, $y_{4i+1}^{(32)}$, $y_{4i+2}^{(32)}$ and $y_{4i+3}^{(32)}$ at this step.

Step 4: Since the size of $\mathsf{MC_{24}}$ is prime with those of $\mathsf{MC_{16}}$ and $\mathsf{MC_{32}}$ (gcd (2,3,4) = 1), the calculation of MixColumn about $\mathsf{MC_{24}}$ is non-trivial compared with the above two Mixcolumns. However, we will show that the addition nodes generated in step 2 and 3 are still enough to construct MixColumn about $\mathsf{MC_{24}}$ without the need to yield new addition nodes at the second layer. First, the calculation of MixColumn about $\mathsf{MC_{24}}$ is directly written as below:

$$\begin{bmatrix} y_{3i}^{(24)} \\ y_{3i+1}^{(24)} \\ y_{3i+2}^{(24)} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} x_{3i} \\ x_{3i+1} \\ x_{3i+2} \end{bmatrix} = \begin{bmatrix} m_{3i,0} + m_{3i+1,1} + x_{3i+2} \\ x_{3i} + m_{3i+1,0} + m_{3i+2,1} \\ x_{3i} + x_{3i+1} + m_{3i+2,0} \end{bmatrix}$$

Then, when i is even, the first two operands of each row are added together, after which the outcome is added with the last operand. The opposite is true when i is odd. Thus, the final results can be computed as below:

$$y_{3i}^{(24)} = \begin{cases} y_{2i}^{(16)} + x_{3i+2}, & i = 2 \cdot k \\ m_{3i,0} + a_{4i+1}, & i = 2 \cdot k + 1 \end{cases}, \quad y_{3i+1}^{(24)} = \begin{cases} y_{2i+1}^{(16)} + m_{3i+2,1}, & i = 2 \cdot k \\ x_{3i} + y_{2i}^{(16)}, & i = 2 \cdot k + 1 \end{cases}$$

$$y_{3i+2}^{(24)} = \begin{cases} a_{4i+2} + m_{3i+2,0}, & i = 2 \cdot k \\ x_{3i} + y_{2i+1}^{(16)}, & i = 2 \cdot k + 1 \end{cases}, \quad 0 \le i \le 4, \ 0 \le k \le 1.$$

In this way, only 15 modular adders of third layer are further needed to obtain the final results $y_{3i}^{(24)}$, $y_{3i+1}^{(24)}$ and $y_{3i+2}^{(24)}$.

B Multi-scale Inverse MixColumns

For inverse MixColumn about $\mathsf{InvMC_{32}}$, we adopt the technique of multiplicative decomposition to reuse computation resource of forward MixColumn. To be specific, the $\mathsf{InvMC_{32}}$

matrix is decomposed as below [CG03]:

$$\begin{bmatrix} e_x & b_x & d_x & 9_x \\ 9_x & e_x & b_x & d_x \\ d_x & 9_x & e_x & b_x \\ b_x & d_x & 9_x & e_x \end{bmatrix} = \begin{bmatrix} 2_x & 3_x & 1_x & 1_x \\ 1_x & 2_x & 3_x & 1_x \\ 1_x & 1_x & 2_x & 3_x \\ 3_x & 1_x & 1_x & 2_x \end{bmatrix} \times \begin{bmatrix} 5_x & 0_x & 4_x & 0_x \\ 0_x & 5_x & 0_x & 4_x \\ 4_x & 0_x & 5_x & 0_x \\ 0_x & 4_x & 0_x & 5_x \end{bmatrix}$$

Therefore, the inverse MixColumn about $\mathsf{InvMC_{32}}$ is calculated as: $z_{4i}^{(32)} = 5 \cdot y_{4i}^{(32)} + 4 \cdot y_{4i+2}$, $z_{4i+1}^{(32)} = 5 \cdot y_{4i+1}^{(32)} + 4 \cdot y_{4i+3}$, $z_{4i+2}^{(32)} = 4 \cdot y_{4i}^{(32)} + 5 \cdot y_{4i+2}$ and $z_{4i+3}^{(32)} = 4 \cdot y_{4i+1}^{(32)} + 5 \cdot y_{4i+3}$. For matrices $\mathsf{InvMC_{16}}$ and $\mathsf{InvMC_{24}}$, a kind of additive matrix decomposition technique is proposed to allow much more sub-expression sharing as below:

$$\begin{bmatrix} b9_x & 68_x \\ d1_x & b9_x \end{bmatrix} = \begin{bmatrix} 68_x & 68_x \\ 0 & 68_x \end{bmatrix} + \begin{bmatrix} d1_x & 0_x \\ d1_x & d1_x \end{bmatrix}$$

$$\begin{bmatrix} 8d_x & 8d_x & 8d_x \\ e5_x & 5c_x & 8d_x \\ 34_x & e5_x & 8d_x \end{bmatrix} = \begin{bmatrix} 8d_x & 8d_x & 8d_x \\ 8d_x & 8d_x & 8d_x \\ 8d_x & 8d_x & 8d_x \end{bmatrix} + \begin{bmatrix} 0_x & 0_x & 0_x \\ 68_x & d1_x & 0_x \\ 68_x + d1_x & 68_x & 0_x \end{bmatrix}$$

Thus, the inverse MixColumn about $\mathsf{InvMC_{16}}$ for $0 \le i \le 7$ can be calculated as:

$$z_{2i}^{(16)} = 68_x \cdot x_{2i} + 68_x \cdot x_{2i+1} + d1_x \cdot x_{2i} \quad z_{2i+1}^{(16)} = d1_x \cdot x_{2i} + d1_x \cdot x_{2i+1} + 68_x \cdot x_{2i+1}$$

The inverse MixColumn about $\mathsf{InvMC_{24}}$ is calculated as:

$$z_{3i}^{(24)} = 8d_x \cdot (x_{3i} + x_{3i+1} + x_{3i+2}) \quad z_{3i+1}^{(24)} = 8d_x \cdot (x_{3i} + x_{3i+1} + x_{3i+2}) + 68_x \cdot x_{3i} + d1_x \cdot x_{3i+1}$$
$$z_{3i+2}^{(24)} = 8d_x \cdot (x_{3i} + x_{3i+1} + x_{3i+2}) + 68_x \cdot x_{3i} + 68_x \cdot x_{3i+1} + d1_x \cdot x_{3i},\ 0 \le i \le 4$$

Note that the adder performing $68_x \cdot x_{2i} + 68_x \cdot x_{2i+1}$ can be reused to compute $68 \cdot x_{3k} + 68_x \cdot x_{3k+1}$ when $2i = 3k$ for $0 \le i \le 7$, $0 \le k \le 4$, that is to say $i = k = 0$ or $i = 3, k = 2$.

C Mastrovito's Multiplier

The straightforward method to implement modular reduction over GF(2^m) is xtime function, which multiplies $a(x)$ by x based on a left shift and a subsequent conditional bit-wise XOR with the irreducible polynomial:

$$\begin{aligned} c(x) = a(x) \cdot x \bmod f(x) &= (0, a_0, a_1, \ldots, a_{n-2}) + (a_{n-1} \cdot f_0, a_{n-1} \cdot f_1, \ldots, a_{n-1} \cdot f_{n-1}) \\ &= (a_{n-1} \cdot f_0, a_0 + a_{n-1} \cdot f_1, \ldots, a_{n-2} + a_{n-1} \cdot f_{n-1}) \end{aligned} \tag{2}$$

The Mastrovito's multiplier transforms the polynomial-basis based modular multiplication to the binary matrix-vector multiplication, which actually combines the least-significant-bit (LSB) first method with xtime function [ZP01]. First, the modular multiplication is written in an LSB-first way as below:

$$\begin{aligned} c(x) &= a(x) \cdot b(x) \bmod f(x) \\ &= (b_0 + b_1 \cdot x + ... + b_{n-1} \cdot x^{n-1}) \cdot a(x) \bmod f(x) \\ &= b_0 \cdot a(x) + b_1 \cdot (a(x) \cdot x \bmod f(x)) + ... + b_{n-1} \cdot (a(x) \cdot x^{n-1} \bmod f(x)) \\ &= [a(x) \bmod f(x), a(x) \cdot x \bmod f(x), ..., a(x) \cdot x^{n-1} \bmod f(x)] \times [b_0, b_1, ..., b_{n-1}]^T \\ &= [A^{(0)}, A^{(1)}, \ldots, A^{(n-1)}] \times [b_0, b_1, ..., b_{n-1}]^T \end{aligned} \tag{3}$$

Then, based on xtime, $A^{(k)}$ can be computed iteratively as:

$$\begin{aligned} A^{(k)} &= A^{(k-1)} \cdot x = a_{n-1}^{(k-1)} \cdot x^n + a_{n-2}^{(k-1)} \cdot x^{n-1} + \ldots + a_0^{(k-1)} \cdot x \\ &= (a_{n-1}^{(k-1)}, a_0^{(k-1)} + a_{n-1}^{(k-1)} \cdot f_1, \ldots, a_{n-2}^{(k-1)} + a_{n-1}^{(k-1)} \cdot f_{n-1}) \quad k = 1, 2, ..., n-1 \end{aligned} \tag{4}$$

Finally, substituting equation 4 into 3, we obtain the bit-parallel multiplication:

$$\begin{bmatrix} c_{n-1} \\ c_{n-2} \\ \cdots \\ c_0 \end{bmatrix} = \begin{bmatrix} a_{n-1}^{(0)} & a_{n-1}^{(1)} & \cdots & a_{n-1}^{(n-1)} \\ a_{n-2}^{(0)} & a_{n-2}^{(1)} & \cdots & a_{n-2}^{(n-1)} \\ \cdots & \cdots & \cdots & \cdots \\ a_0^{(0)} & a_0^{(1)} & \cdots & a_0^{(n-1)} \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ \cdots \\ b_{n-1} \end{bmatrix} \tag{5}$$

For a certain fixed irreducible polynomial $f(x)$ and operand $a(x)$, the binary matrix A can be predetermined so that only XORs are required to implement the entire modular multiplication. It is reported in [SKOP15] that the choices of both $f(x)$ and constant operand exert an influence on the number of XOR gates. Additionally, the number of XOR gates and circuit depth for implementing matrix-vector multiplication can be further minimized through sub-expression sharing. Therefore, Mastrovito's multiplier is usually used to customize the constant multiplier with regard to a fixed irreducible polynomial at the price of losing flexibility and scalability.

Table 7: The detailed implementation of constant modular multipliers.

constant modular multiplier about 68_x			
$y_7^{(1)} = y_1^{(1)} \oplus x_1$	$y_3^{(1)} = t_0 \oplus y_0^{(1)}$	$y_6^{(1)} = t_1 \oplus y_2^{(1)}$	$y_4^{(1)} = t_2 \oplus x_1$
$y_1^{(1)} = t_3 \oplus t_2$	$y_2^{(1)} = t_4 \oplus x_3$	$y_0^{(1)} = t_5 \oplus t_4$	$y_5^{(1)} = t_5 \oplus t_6$
$t_1 = t_7 \oplus t_6$	$t_0 = t_8 \oplus t_9$	$t_8 = x_0 \oplus x_4$	$t_6 = x_0 \oplus x_5$
$t_7 = x_1 \oplus x_4$	$t_5 = x_2 \oplus x_3$	$t_2 = x_2 \oplus x_4$	$t_4 = x_5 \oplus x_6$
$t_3 = x_5 \oplus x_7$	$t_9 = x_6 \oplus x_7$		
constant modular multiplier about $d1_x$			
$y_3^{(2)} = y_1^{(2)} \oplus t_0$	$y_7^{(2)} = t_1 \oplus y_1^{(2)}$	$y_0^{(2)} = t_2 \oplus y_2^{(2)}$	$y_6^{(2)} = t_3 \oplus t_0$
$t_2 = t_1 \oplus x_2$	$y_4^{(2)} = t_4 \oplus t_1$	$t_3 = t_5 \oplus t_1$	$y_1^{(2)} = t_4 \oplus t_6$
$t_0 = t_7 \oplus x_1$	$y_2^{(2)} = t_8 \oplus x_4$	$y_5^{(2)} = t_9 \oplus t_{10}$	$t_1 = x_0 \oplus x_1$
$t_9 = x_1 \oplus x_2$	$t_7 = x_2 \oplus x_3$	$t_4 = x_3 \oplus x_4$	$t_{10} = x_4 \oplus x_5$
$t_5 = x_5 \oplus x_6$	$t_8 = x_5 \oplus x_7$	$t_6 = x_6 \oplus x_7$	

D Constant Modular Multiplier

For the modular multiplication by constants $8d_x$, 68_x and $d1_x$, we firstly generate three 8×8 binary matrices by following the equation 4 and 5 respectively.

M_0, HW=11

$$\begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&0&0 \\ 1&0&0&0&1&0&0&0 \\ 1&0&0&1&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0 \end{bmatrix}$$

M_1, HW=34

$$\begin{bmatrix} 0&1&1&0&1&1&0&1 \\ 1&1&0&1&1&0&1&0 \\ 1&0&1&1&0&1&0&0 \\ 0&1&1&0&1&0&0&0 \\ 1&0&1&1&1&1&0&1 \\ 0&0&0&1&0&1&1&0 \\ 0&0&1&0&1&1&0&1 \\ 0&0&1&1&0&1&1&0 \end{bmatrix}$$

M_2, HW=37

$$\begin{bmatrix} 1&1&0&1&1&0&1&1 \\ 1&0&1&1&0&1&1&0 \\ 0&1&1&0&1&1&0&0 \\ 1&1&0&1&1&0&0&0 \\ 0&1&1&0&1&0&1&1 \\ 0&0&0&0&1&1&0&1 \\ 0&0&0&1&1&0&1&1 \\ 1&1&1&0&1&1&0&1 \end{bmatrix}$$

It is observed that matrix M_0 with respect to $8d_x$ is very sparse, so that only 3 XORs are required to implement the constant modular multiplier, namely $y_0^{(0)} = x_0$, $y_1^{(0)} = x_7$, $y_2^{(0)} = x_6$, $y_3^{(0)} = x_5$, $y_4^{(0)} = x_0 \oplus x_4$, $y_5^{(0)} = x_0 \oplus x_3$, $y_6^{(0)} = x_2$, $y_7^{(0)} = x_0 \oplus x_1$. However,

when directly computing M_1 and M_2 based matrix vector multiplication, $34-8=26$ XORs and $37-8=29$ XORs are needed, respectively. Based on the backward search framework, the area cost of constant modular multiplications about 68_x and $d1_x$ can be reduced to 18 XORs with minimum circuit depth 3 XORs and 19 XORs with minimum circuit depth 3 XORs, respectively. The concrete implementation is shown in Table 7.

E Backward Search Framework

To customize the constant modular multiplier, we adopt the backward search framework proposed in [LWF+22]. This heuristic search algorithm is inspired by the optimization method for constant matrix multiplication proposed in [KHZ17]. Compared with the forward search framework [BP09], the backward search framework guarantees the circuit depth to be minimum while still reducing as much area cost as possible. The overall execution steps of backward search framework are described as below:

Step 1: Transform the constant C_x into binary matrix $A_{m\times m}$ based on Mastrovito's multiplication under a certain irreducible polynomial $f(x)$. Then, the goal is turned to calculate the binary matrix vector multiplication: $y_{m\times 1}=A_{m\times m}\cdot x_{m\times 1}$.

Step 2: Calculate the Hamming weight HW_i and logarithmic depth $S=\log_2 HW_i$ for each row of the matrix, where the maximum depth is determined as S_{max} for $0\le i\le m-1$. Then, node is defined as the bit vector of each row.

Step 3: Initialize the X set as the collection of input unit nodes x_i with depth equal to 1, the W set as the collection of nodes w_i with depth equal to S_{max}, and the P set as the collection of nodes p_i with depth less than S_{max}, where $0\le i\le m-1$. Sort the nodes p_i in descending order according to depth.

Step 4: Split the node w_i into two nodes until set W is empty, which is based on one of the following strategies:

1) Construct node w_i with one node x_j and the other node p_k.
2) Construct node w_i with one node p_j and another generated new node g_k. The depth of g_k should be less than S_{max}. Append the new node to set P.
3) Construct node w_i with two generated new nodes g_k and g_n. The depth of g_k and g_n should be less than S_{max} but at least one of them is equal to $S_{max}-1$. Append nodes g_k and g_n to set P.

Note that each splitting operation generates a directed sub-graph with two edges and three nodes.

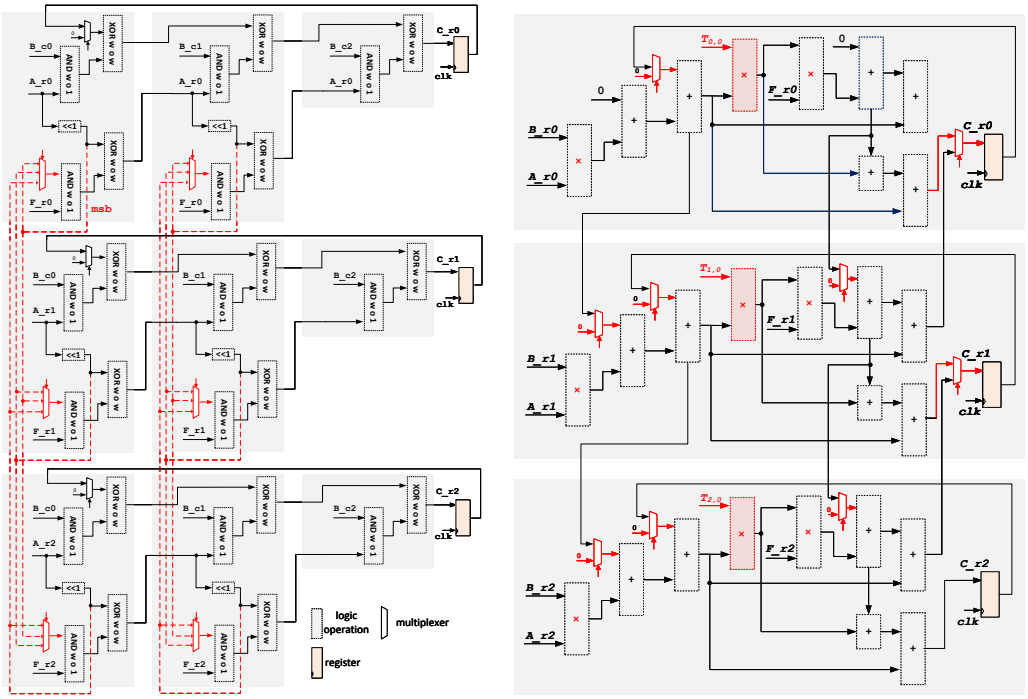
Step 5: Update the maximum depth as $S_{max}=S_{max}-1$. Update the sets W and P based on new S_{max}. If $W\cup X=X$, then return the current directed graph. Otherwise, we go to Step 4 for next iteration.

We add an extra sorting operation for set P at each iteration. The descending order guarantees that the node with maximum depth among p_i is preferentially picked up, which aims to speed up the convergence for small-dimension matrix. More details about the backward search framework could refer to the original paper [LWF+22].

F Alternative Methods to VMM

As comparison objects to the proposed radix-2 Montgomery based VMM, Figure 19 presents two examples of alternative methods to VMM. As shown in Figure 19(a), radix-2 LSB-first

based VMM consumes proportionally increased fan-ins (3×1-bit inputs marked by the red broken line) of MUXs to support configuration, because the MSB of output comes from 3 PEs. Figure 19(b) reveals that high-radix Montgomery based VMM needs extra pre-computation for $T_{0,0}$,$T_{1,0}$,$T_{2,0}$ and involves multiple w-bit multipliers, which results in high area overhead.



(a) Radix-2 LSB-first VMM (b) High-radix Montgomery VMM

Figure 19: Two typical examples of alternative methods to VMM (setting parameter as 3w-bit A and 3-bit B).