A Pipelined AES and SM4 Hardware Implementation for Multi-tasking Virtualized Environments

Yukang Xie, Hang Tu, Qin Liu(✉), and Changrong Chen

Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China
{bathtub,tuhang,qinliu,chenchangrong}@whu.edu.cn

Abstract. Virtualization techniques are becoming increasingly prevalent and are driving trends in hardware development to offer parallelization support for multi-tasking. Existing works on hardware designs of the Advanced Encryption Standard (AES) and SM4 encryption algorithms have primarily focused on optimizing metrics such as throughput and area, but have not fully addressed the demands in virtualized environments. In this article, we propose innovative optimization schemes that partition the resources in AES and SM4 cipher modules into smaller, independent units that can execute tasks from different guests in parallel. Such designs can improve hardware utilization efficiency and enhance the user experience in virtualized environments. Our FPGA-validated designs achieve comparable circuit performance in terms of throughput/area efficiency to existing work. Experiments show that in virtualized environments lacking block-wise parallelism (e.g., cipher block chaining (CBC) mode), our approach reduces context switches over 50% and decreases average task pending time around 75% with similar hardware needs.

Keywords: Hardware acceleration · Parallelism · Virtualization · AES · SM4

1 Introduction

Modern computer architecture practices commonly employ dedicated hardware to offload compute-intensive tasks, such as graphics processing and tensor processing, from CPUs [8,17]. Cryptographic operations lend themselves well to hardware implementation due to their routine patterns of execution, along with their demands for considerable resources. To address various use cases ranging from server-level acceleration to co-processing on embedded devices, industry professionals and researchers have made significant efforts to optimize the hardware implementation of widely-used encryption algorithms. These optimizations have targeted several metrics, including throughput, circuit area, and countermeasures against side-channel attacks. The rise of virtualization techniques

Z. Tari et al. (Eds.): ICA3PP 2023, LNCS 14488, pp. 275–291, 2024.
https://doi.org/10.1007/978-981-97-0801-7_16

[1,14], in which multiple tasks from different guests are executed on a shared physical infrastructure, has introduced new optimization requirements for hardware designers. In this paper, we propose an innovative optimization scheme for implementing two widely-used symmetric-key encryption algorithms, Advanced Encryption Standard (AES) and SM4, to enhance the designs' suitability for virtualized environments.

Our motivation stems from an observation of existing hardware implementations of AES and SM4, particularly those with pipelined structures. While these implementations leverage additional register resources to maximize circuit frequency and achieve higher throughput, this can result in resource wastage when the pipeline is not fully utilized during operation. Such issue becomes especially pronounced when the guest employs a block operation mode that necessitates sequential processing of data blocks, meaning that the hardware can only handle one data block at a time throughout the entire process. In our designs, resources within an AES or SM4 cipher module can be subdivided into smaller units based on the number of pipeline stages. From the perspective of guests, each resource unit operates independently and can be combined with a specific user key, allowing the module to execute tasks from different guests in parallel. As a result, hardware utilization rate is enhanced in these serial-styled block operation modes. Moreover, since there are more independent resource units available, hypervisors or operating systems can switch contexts less frequently between different guest tasks in the cipher hardware, thereby improving overall system performance.

In this paper, we adopt this optimization strategy for both AES and SM4 implementations, developing four-staged pipeline structures capable of processing tasks from four different guests in parallel. Our designs are open-sourced and have been validated and evaluated on a Zynq UltraScale+ and a Kintex-7 FPGA device. Synthesis and implementation results demonstrate that our designs achieve comparable results in circuit efficiency relative to existing studies. We also notice that previous studies have devoted less attention to the practical methods for incorporating cryptographic hardware into larger systems. To fill this gap, we have developed a prototype cryptosystem based on our cryptographic modules and discussed its key design principles. Experiments conducted on the prototype system reveal that, in virtualized environments, our designs reduce context switches by over 50% and decrease average task pending time by approximately 75% in operation modes lacking block-wise parallelism. Our designs achieve these improvements while maintaining similar hardware consumption compared to classic architectures.

The remainder of this paper is organized as follows: Sect. 2 discusses the background of both AES and SM4 algorithms, as well as related work in their hardware implementations. Section 3 and 4 presents the details of the proposed designs, while Sect. 5 includes evaluation and experimental data. Section 6 provides a discussion on our designs, and the final section concludes the paper.

2 Background

This section presents a concise overview of the AES and SM4 algorithms, as well as a survey of prior research on their hardware implementations.

2.1 Overview of AES and SM4

AES is a widely adopted symmetric-key encryption algorithm that operates on blocks of 128 bits [6]. As shown in Fig. 1, this algorithm relies on iterative rounds of transformations-including substitution, permutation, and mixing operations-to encrypt and decrypt input data. The number of transformation rounds varies between 10, 12, or 14, depending on the key length, which is either 128, 192, or 256 bits. In each round, the intermediate state is XORed with a 128-bit round key, which is generated by the AES key schedule algorithm using the main cipher key. The encryption and decryption processes of the AES algorithm are inverses of each other, and their datapaths can be partially combined in specific implementations.

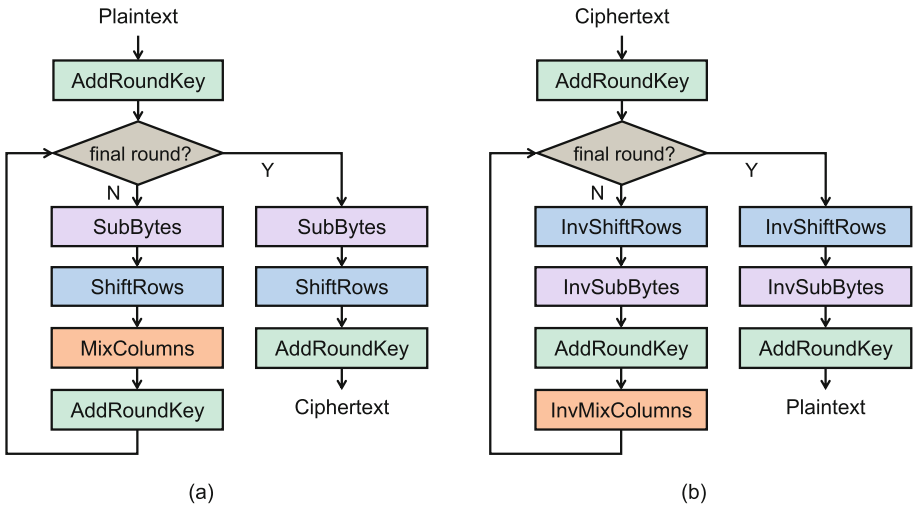

Fig. 1. AES (a) encryption and (b) decryption process.

SM4, released by the Office of State Commercial Cryptography Administrator of China, is a standard algorithm in ISO/IEC 18033-3:2010/Amd 1:2021 [11]. It operates on 128-bit blocks and uses a fixed key length of 128 bits. Like AES, this algorithm comprises an encryption/decryption process and a key scheduling process. Figure 2 illustrates that SM4 employs 32 rounds of permutations, with each round making use of a 32-bit round key. The round permutation in SM4 is less complex than the round transformation in AES, and only updates a 32-bit

word in each iteration. The datapaths for encryption and decryption in the SM4 algorithm are identical, with the only difference being the order in which the round keys are used.

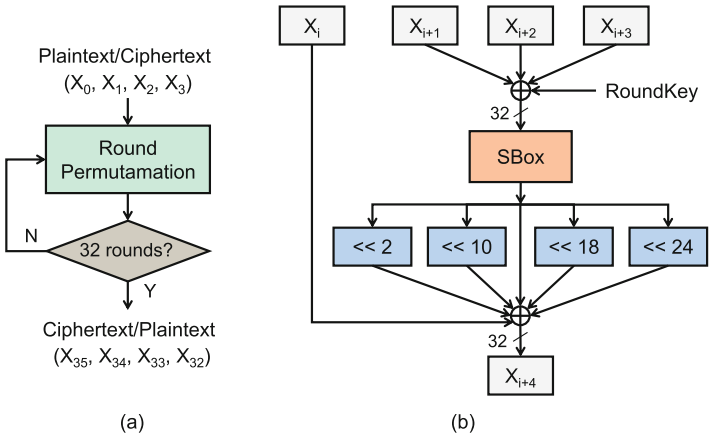

Fig. 2. (a) Overall process and (b) round permutation of SM4.

To process input data of arbitrary size, block cipher algorithms like AES and SM4 must be employed in specific modes of operation. Various modes of operation have been developed, including Electronic Codebook (ECB), Cipher Block Chaining (CBC), Output Feedback (OFB), and Counter (CTR). From an efficiency standpoint, current mainstream operation modes can be categorized into two types: block-wise parallel support mode, exemplified by Galois/Counter Mode (GCM), and modes that do not support block-wise parallelism, as represented by Counter with CBC-MAC (CCM). In operation modes that support block-wise parallelism, there are no dependencies between data blocks, allowing them to be processed in parallel. In contrast, operation modes that lack block-wise parallelism require data blocks to be processed serially as each block's processing result depends on the previous one. Since both types of operation modes are widely adopted, hardware designers must consider their characteristics and the impact they bring.

2.2 Related Works and Discussion

Optimized hardware implementations for both AES and SM4 have been extensively studied. Various designs have been proposed in prior works, targeting different optimization metrics based on application requirements. Numerous research efforts have investigated the trade-off between circuit area and hardware throughput. For example, the authors in [4] presented an ultra-compact design for AES on FPGA, utilizing only 184 logic slices and no Block RAM

(BRAM). Another work in [20] introduced a low-area SM4 design costing only 164 slices on a Virtex-6 FPGA device.

Conversely, to achieve higher throughput, unrolled and pipelined designs have been proposed, such as those in [3,12,13,19]. Although these designs can achieve throughputs of several tens of gigabits per second, they also consume considerable resources. Moreover, the theoretical throughput of these unrolled and pipelined designs can only be achieved when operating in a mode that supports block-wise parallelism, such as ECB and GCM. This limitation is discussed in [21], which abandons pipelined design and presents a round-based architecture with a highly compressed datapath structure.

However, our analysis indicates that hardware resource wastage in pipelined structures is not necessarily inevitable in operation modes that do not support block-wise parallelism, such as CBC and CCM. In these block-chaining operation modes, idle resources in the pipeline can still serve tasks from other guests, which is particularly beneficial for scenarios involving virtualization. Building upon this observation, we propose rolled, four-staged pipeline structures for both AES and SM4 algorithms in this paper. This design supports task-wise parallelism for up to four guests, effectively raising the hardware utilization rate and addressing the limitations of existing approaches.

3 Proposed AES and SM4 Architecture

In this section, we introduce the core of our work, including the implementation of the four-staged encryption and decryption modules for both AES and SM4 algorithms.

Our design is implemented in Chisel [2], a modern hardware description language with rich parameterization and modularity features [18]. The architectures of both the AES and SM4 modules adopt a round-based, four-staged pipeline structure. The key idea for enabling guest-level parallelism is to maintain a set of control information for each running task in the pipeline. This allows the circuit to index the corresponding round key when conducting key-related operations (e.g., AddRoundKey in AES). With this measure, each pipeline stage can be regarded as an independent resource from the user's perspective.

We made several crucial design decisions in our implementation:

(a) Although using lookup tables to implement the S-box is common in high-throughput designs, pipelined combinational logic circuits usually show better throughput/area efficiency. We employed the S-box presented by Maximov and Ekdahl in [15] because it is a state-of-the-art design that can be evenly partitioned without introducing excessively large registers.
(b) We chose a round-based architecture because it is less area-consuming and can adapt to more usage scenarios. When higher throughput is required, users can instantiate multiple modules to achieve that. We have determined the number of pipeline stages to be four, based on the objective of balancing register usage and maximizing circuit frequency.

(c) Since the key scheduling module is used much less frequently than the encryption/decryption modules in most cases, we decided to separate their implementation. The prototype cryptosystem proposed in Sect. 4 combines a key scheduling module with multiple encryption/decryption modules.

3.1 AES Encryption and Decryption Architecture

In our envisioned usage scenarios, multiple cryptographic modules will be instantiated to handle the workload from a large number of guests. Consequently, we focus on simplifying the functionality of each individual module, opting not to merge the encryption and decryption modules. Users can configure and generate the necessary number of encryption and decryption modules in cryptosystems based on their application needs, leveraging the flexibility provided by Chisel.

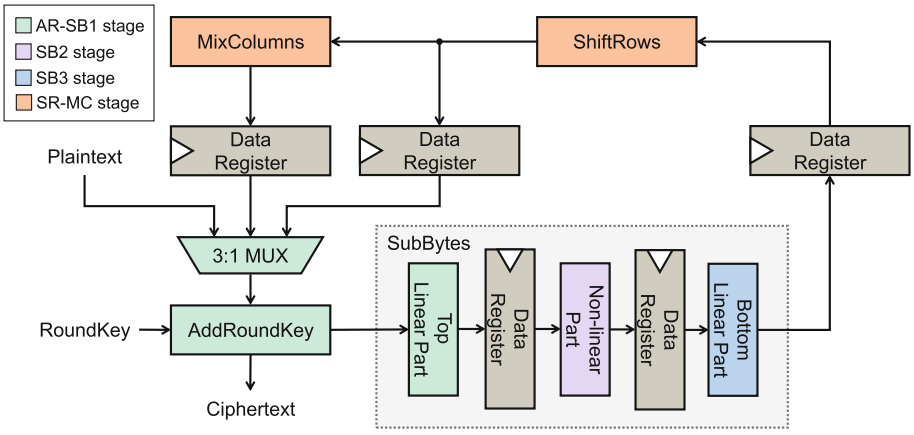

Fig. 3. Proposed AES encryption architecture.

The proposed four-staged architectures for AES encryption and decryption are illustrated in Fig. 3 and Fig. 4. For both processes, two pipeline registers are inserted before and after the non-linear transformation part of the S-box to reduce the critical path in the SubBytes/InvSubBytes module. Additionally, two other pipeline registers are inserted before and after the MixColumns/InvMixColumns module, separating its datapath from the linear transformation part of the S-box.

Here we use the AES encryption architecture as an example to explain the involved pipeline stages. The input data is first processed in the `AR-SB1` stage, which contains the AddRoundKey module and the top linear part of the S-box. The second and third stages, `SB2` and `SB3`, contain the non-linear part and bottom linear part of the S-box, respectively. The last stage of the round transformation is `SR-MC`, which includes the ShiftRows and MixColumns modules. There is an extra register bypassing the MixColumns module since the final round of

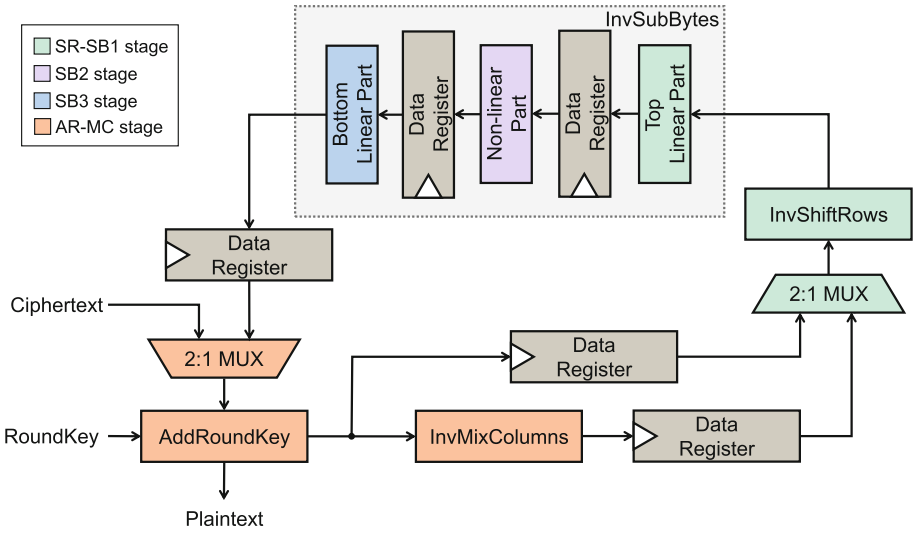

Fig. 4. Proposed AES decryption architecture.

AES encryption does not contain it. Figure 5 depicts a timing diagram for AES-128 encryption, illustrating how distinct input blocks are processed through the pipeline. Note that a round transformation of AES in our design takes four cycles, therefore an AES-128 encryption operation requires 40 cycles. Given that the four-staged pipeline can process four tasks in parallel, the proposed architecture can accomplish four AES-128 encryptions in 43 cycles.

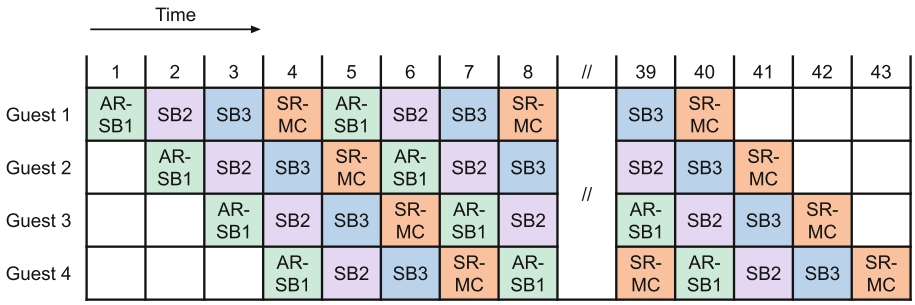

Fig. 5. Example pipeline timing diagram for AES-128 encryption.

In our design, the data register incorporated in the pipeline contains not only a 128-bit AES state but also a set of control information. Each control information set is associated with a specific task, indicating the status of the AES state for that task. The subfields of the control information include:

- `taskID`: a 2-bit specifier for the task.
- `isIdle`: a 1-bit flag indicating whether the task is valid.
- `keyLength`: a 2-bit field indicating whether the task is running in AES-128, AES-192, or AES-256.
- `rounds`: a 4-bit field indicating the number of rounds for the task.

The only key-related operation in AES is AddRoundKey. This module reads the corresponding round key from an external source based on the control information of its input task. Unlike unrolled architectures, our design natively supports all three AES variants of key length. From the user's perspective, each instance of our design includes four physical resources that can independently perform AES tasks, offering good flexibility in virtualized environments.

3.2 SM4 Encryption/Decryption Architecture

Figure 6 displays the proposed SM4 encryption/decryption architecture. The SM4 algorithm employs an unbalanced Feistel network structure, updating a 32-bit word in each round. SM4 encryption and decryption share the same datapath and only differ in the order of applying round keys. Similar to the AES architectures, two pipeline registers are allocated in the S-box, before and after the non-linear transformation part. Two additional two pipeline registers are placed before and after the linear transformation logic outside the S-box. Furthermore, the unchanged part in each round is preserved by shift registers.

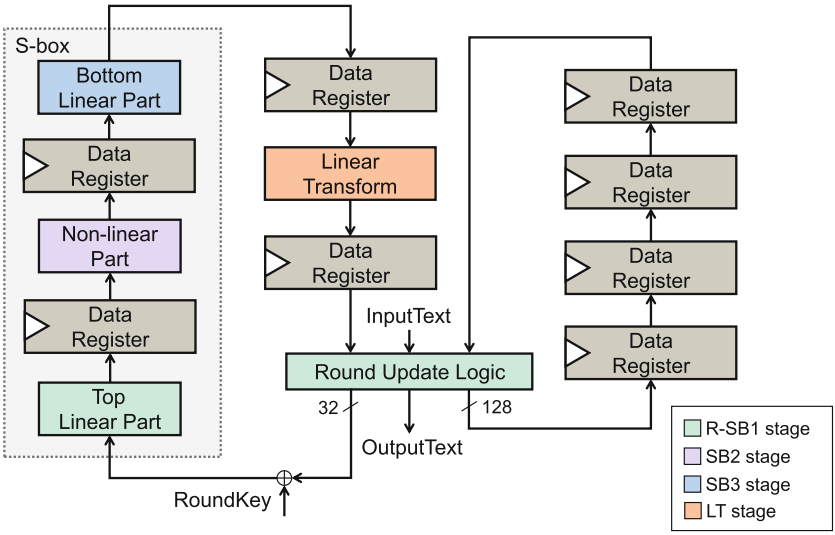

Fig. 6. Proposed SM4 encryption/decryption architecture.

Our proposed architecture for SM4 utilizes the same control information set as that used by the AES architecture, except for the absence of the `keyLength`

subfield since SM4 only supports a fixed key length of 128 bits. Although it is possible to add an `isEnc` subfield to the control information set to enable both encryption and decryption tasks to run within the same module, we chose not to implement this feature to avoid increasing the complexity of the external controller.

4 A Prototype Cryptosystem for Virtualization

As a reference for applying our design in virtualized environments, we further construct a prototype cryptosystem comprising four instances of the cryptographic modules discussed in Sect. 3. In this section we present the design of our prototype system, as well as its programming sequence for users.

4.1 Prototype Cryptosystem Architecture

Figure 7 depicts the structure of the prototype cryptosystem. As each cryptographic module is capable of processing four guest tasks in parallel, this prototype system can support a maximum of sixteen guest tasks concurrently. Each guest task can operates either in ECB mode or CBC mode. The prototype system employs the AXI interface for easy integration into FPGA designs.

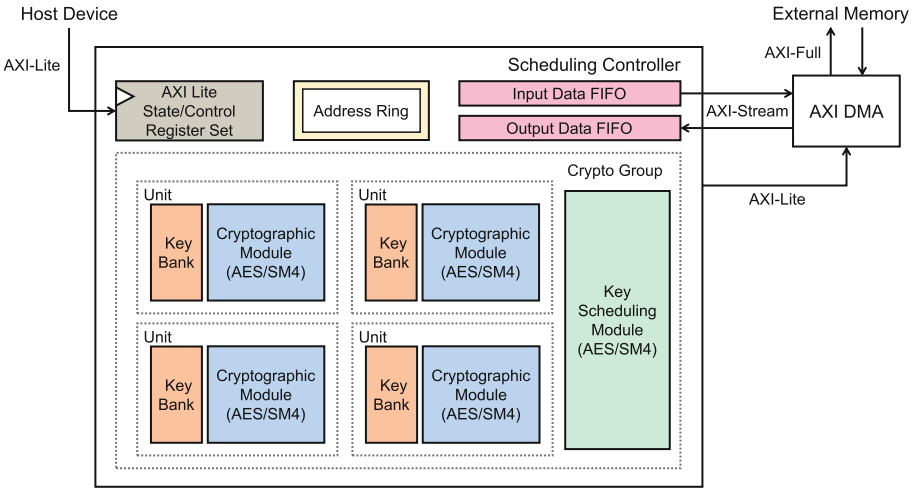

Fig. 7. The multi-instance prototype cryptosystem.

In our design, each cryptographic module is combined with a key bank to form a "unit". The input and output dataflows of a single cryptographic module are managed by the logic within the unit. Our implementation adopts the idea of another Chisel design proposed in [9], enabling users to easily generate the

desired combination of AES/SM4 encryption/decryption units through simple configuration. Since the key scheduling module is less utilized than the cryptographic module in typical cases, we have a single key scheduling module serving four units. The round keys generated by this scheduling process are stored in key banks for future use. This combination is referred to as a "crypto group" in our prototype. The control logic within the crypto group manages the plaintext, ciphertext, and key dataflows. The crypto group is then wrapped into a scheduling controller, which receives task configuration sent by a host device through the AXI-Lite bus.

Maximizing the utilization of cryptographic acceleration hardware is contingent on ensuring efficient data transfer between the accelerator and external devices. Therefore, we employed a DMA IP in our prototype system to move plaintext/ciphertext between external memory and our design. To ensure even parallelization of tasks from different guests within the module, the scheduling controller must issue DMA commands from various tasks alternately. This is achieved through an address ring structure within the scheduling controller.

4.2 Programming Sequence of the Prototype Cryptosystem

As previously mentioned, from the user's perspective, our prototype system offers sixteen independent cryptographic resources capable of performing encryption or decryption operations. Typically, the hypervisor or OS allocates a single cryptographic resource to each guest. Figure 8 illustrates the programming sequence for utilizing cryptographic resources, which consists of the following steps:

- `keySet`: The hypervisor/OS sets a guest's main key, initiating the key scheduling process.
- `taskAssign`: An encryption or decryption task from the guest begins. The task configuration specifies the source and target addresses of input and output data, the data length, and the mode of operation.
- `keyDestroy`: The round keys in the key storage bank are destroyed, freeing up the cryptographic resource for future use.

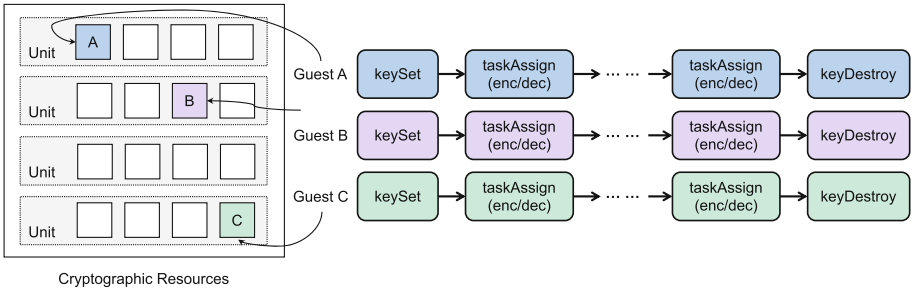

Fig. 8. Programming sequence of the prototype cryptosystem.

As the manager of hardware resources, the hypervisor or OS is responsible for maintaining the mapping between guests and cryptographic resources. Once a cryptographic resource is assigned to a guest, it cannot serve other guests until the hypervisor or OS sends a `keyDestroy` signal to release the resource. In situations where all cryptographic resources are occupied by various guests, and new guest requests arrive, context switches are triggered. The hypervisor or OS must free up some cryptographic resources and reallocate them to the incoming guests. The overhead associated with context switches arises from both software and hardware aspects. On the software side, the hypervisor or OS must update the mapping between guests and resources. Meanwhile, on the hardware side, before conducting encryption or decryption operations, the module must perform the key scheduling process again.

5 Experimental Results

This section presents a comparison of circuit performance across proposed designs and existing works, along with experiments evaluating the efficiency gains in multi-guest virtualized environments afforded by our designs.

5.1 Circuit Performance Comparison

The designs proposed in Sect. 3 are thoroughly verified on a Zynq UltraScale+ (xczu7eg-ffvc1156) and a Kintex-7 (xc7k325t-ffg900) FPGA device using Xilinx Vivado 2022.1 for synthesis and implementation.

Table 1. Comparison of AES Implementation

Designs	Platforms	Slices	Max. freq. (MHz)	Throughput (Gbps)	Efficiency (Mbps/slice)
[4]	XC3S50-5	184	45.64	0.037	0.2
[13]	XC7VX690T	3436	516.8	66.1	19.2
[16]	XC6VLX240T	4830	617.63	79	16.36
[19]	XC5VLX	5974	622.4	79.7	13.3
[10]	XC7V585TFF	1355+80BRAMs	374	47.8	-
This work	XC7K325T	371	414.3	5.3	14.29
	XCZU7EG	430	535.3	6.85	15.93

The synthesis results of both the AES and SM4 encryption/decryption architectures are compared with previous works in Tables 1 and 2. These tables report hardware utilization, maximum frequency, throughput, and efficiency. Since our round-based designs strike a balance between throughput and hardware utilization, we have selected works focusing on throughput maximization and area minimization for comparison. The proposed AES implementation achieves a maximum frequency of 414.3 MHz and a throughput of 5.3 Gbps, occupying 371 slices. The corresponding figures for the proposed SM4 implementation are 415.3 MHz,

Table 2. Comparison of SM4 Implementation

Designs	Platforms	LUTs	FFs	Slices	Max. freq. (MHz)	Throughput (Gbps)	Efficiency (Mbps/slice)	Efficiency (Mbps/LUT+FF)
[20]	XC6VLX240T	-	-	164	253	0.25	1.54	-
[7]	EP4SE230F29	687	448	-	210.26	0.82	-	0.72
		7667	5438	-	212.13	27.1	-	2.07
[3]	XCZU7EV	8655	10071	-	923.36	118.19	-	6.31
This work	XC7K325T	575	816	236	415.3	1.66	7.03	1.19
	XCZU7EG	574	816	266	547.3	2.19	8.23	1.58

1.66 Gbps, and 236 slices, respectively. The results show that our designs surpass existing compact designs of AES and SM4 in both throughput and efficiency and achieve fair results in efficiency compared to existing high-speed designs. On the one hand, our designs are lightweight enough to be deployed on resource-constrained platforms where virtualization techniques are applied. On the other hand, due to the good throughput/area efficiency of our design, users can always choose to instantiate multiple modules to achieve higher throughput, which is essentially equivalent to the unrolled structure.

It is important to note that, despite the lack of full authority in the comparative data due to differences in FPGA platforms and the randomness of synthesis, placing, and routing optimization algorithms in Vivado, our design successfully maintains competitive hardware performance while introducing support for parallelism of multiple guest tasks. This is reasonable because our designs introduce relatively little additional control information.

5.2 Multi-guest Efficiency Improvement

To validate the performance improvement of the proposed design in multi-tasking virtualization scenarios, we conducted experiments based on the prototype system described in Sect. 4, integrated with the Zynq UltraScale+ MPSoC. The results demonstrate that our design is well-suited for virtualized environments.

Two critical performance metrics in virtualized environments are the average pending time for each task and context switching rate. We compared these two metrics between the classic design and the proposed architecture in the experiment. In the classic design, guest-level parallelism is not exploited, and each guest utilizes the full resources of a cryptographic hardware instance. However, in our design, each guest only occupies a single pipeline stage of the instance and is processed in parallel. To avoid potential unfairness introduced by differences in hardware utilization and given that most existing works are not open-sourced, we decided to simulate the classic architecture using our own design with slight modifications. This is possible because the working method of the classic architecture can be considered a subset of that of our proposed design. There are four AES encryption modules initialized in the prototype system.

In the application for the experiment, a randomly generated task sequence, containing tasks belonging to random guests, was read in by the scheduling function, which allocates cryptographic resources to guest tasks and records their pending time. A context switch occurs when idle resources exist but no

resources have been occupied by the incoming guest. We fix the total length of tasks in the task sequence to 4 GB, and the length of each task is randomly determined between 4 MB and 64 MB. The number of guests is set to 8, 16, 32, and 64. Additionally, two operation modes, CBC and ECB, are tested in the experiment, representing block-wise parallelism unsupported and block-wise parallelism supported modes, respectively. We conducted twenty runs for each setup, and each task sequence is reused for setups with the same guest number configuration.

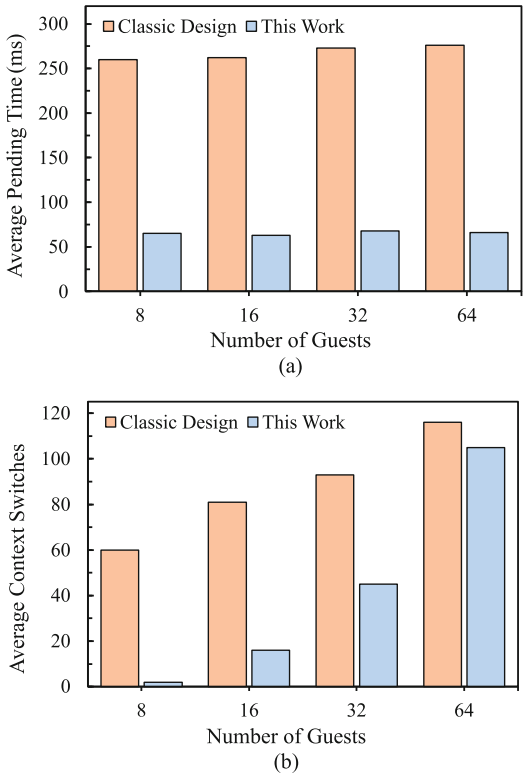

Fig. 9. Multi-guest performance evaluation in CBC mode in terms of (a) task average pending time and (b) average context switches.

Figure 9 and 10 summarize the average task pending time and the count of context switches for both the classic design and our architecture. The experimental results show that the proposed design can significantly reduce context switches when the guest number is within a certain range. When the number of guests is 32 or less, our architecture can reduce context switches by over 50%. This is because our design contains sixteen independent resources, which is four times that of the classic one. However, when the number of guests greatly exceeds

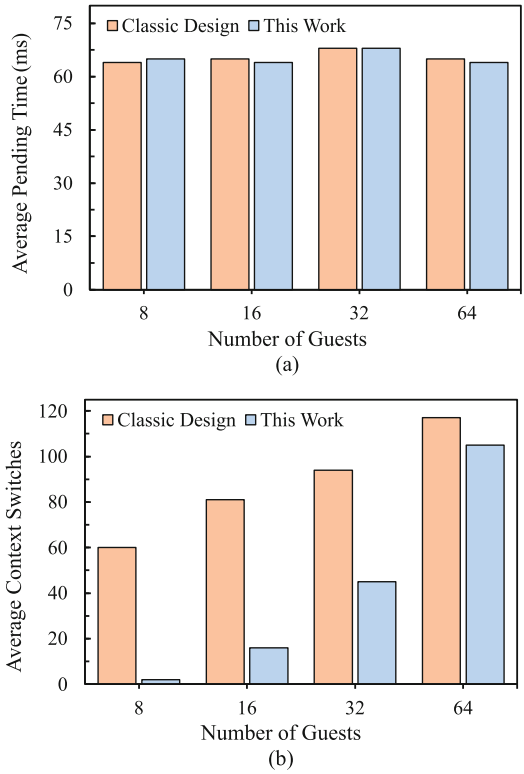

Fig. 10. Multi-guest performance evaluation in ECB mode in terms of (a) task average pending time and (b) average context switches.

the number of resources, our design no longer has a significant advantage, indicating that the user must instantiate more instances to further reduce context switches. In the operation mode that does not support block-wise parallelism, our design shows a clear performance advantage over the classic architecture in terms of pending time, reducing it by 75%. Classic pipelined designs typically perform poorly in this situation as they cannot fully utilize all the resources in the pipeline. In contrast, when operating in a mode that supports block-wise parallelism, both architectures exhibit similar performance in terms of average pending time. It is worth noting that in practical applications, the advantages of our design may be more pronounced because, in actual hypervisors or operating systems, context switches have greater software overhead.

6 Discussion

In multi-tasking virtualized environments, a critical job of the hypervisor/OS is to schedule tasks coming from different guests with limited hardware resources,

where context switching is inevitable. Frequent context switching could cause a loss in performance and, for cryptographic hardware, could potentially introduce security implications since sensitive information, including user keys, is conveyed more regularly via the bus. Designers of large-scale systems thus generally instantiate a substantial number of dedicated hardware to meet demand. Our work provides an alternative optimization perspective, demonstrating that this issue can be addressed by dividing resources into finer granularity at the hardware level.

The key to our design is increasing the available hardware resources from the user's perspective by making the pipeline stages independent. The independent pipeline stage brings advantages in two ways. First, it offers great flexibility to the hypervisor/OS level. The hypervisor/OS can apply different scheduling strategies, such as allowing each guest to occupy at most one resource in situations with a large number of guests to reduce context switching, or temporarily allowing a superior user to occupy all the resources of an instance in priority scheduling, which makes our design equivalent to the classic architecture. Second, it enhances the hardware utilization rate when the hardware operates in a mode that does not support block-wise parallelism.

Although the proposed structure is round-based, our design can be easily applied to an unrolled or partially unrolled structure to obtain a higher throughput, since the additional control information register in each pipeline stage is not a significant overhead compared to the 128-bit AES/SM4 state itself. Furthermore, our design strategy is not limited to specific cryptographic algorithms and can potentially be applied to more cryptography implementations, as long as it does not take excessive overhead to maintain the control information for key-related operations. An interesting example would be Ascon [5], the new lightweight cryptography standard selected by NIST, which adopts a sponge structure and only involves key-related operations at its initialization and finalization stage. This feature makes it natural for Ascon implementation to have independent pipeline structure.

7 Conclusion

This paper presented optimized hardware implementations for AES and SM4 algorithms, targeting multi-tasking virtualization scenarios. Evaluation on FPGAs shows that our designs can significantly reduce the average pending time and context switch demands in virtualized environments compared to classic architectures, while achieving competitive throughput/area efficiency. A prototype cryptosystem was further developed to showcase the practicality of our designs, indicating their scalability and potential applications in various scenarios. By enhancing hardware utilization rate and decreasing pending time in virtualized environments, our work demonstrates optimization strategies useful for a wider range of metrics and applications.

The source codes of the AES and SM4 modules are available on GitHub at https://github.com/bathtub-01/cluster-AES.

Acknowledgements. The work was supported by the National Natural Science Foundation of China (No. 62272348).

References

1. Alam, I., et al.: A survey of network virtualization techniques for internet of things using SDN and NFV. ACM Comput. Surv. (CSUR) **53**(2), 1–40 (2020). https://doi.org/10.1145/3379444
2. Bachrach, J., et al.: Chisel: constructing hardware in a Scala embedded language. In: Proceedings of the 49th Annual Design Automation Conference, DAC 2012, pp. 1216–1225. Association for Computing Machinery, New York (2012). https://doi.org/10.1145/2228360.2228584
3. Chen, Y., et al.: Exploring the high-throughput and low-delay hardware design of SM4 on FPGA. In: 2022 19th International SoC Design Conference (ISOCC), pp. 211–212 (2022). https://doi.org/10.1109/ISOCC56007.2022.10031393
4. Chu, J., Benaissa, M.: Low area memory-free FPGA implementation of the AES algorithm. In: 22nd International Conference on Field Programmable Logic and Applications (FPL), pp. 623–626 (2012). https://doi.org/10.1109/FPL.2012.6339250
5. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2: lightweight authenticated encryption and hashing. J. Cryptol. **34**(3), 1–42 (2021). https://doi.org/10.1007/s00145-021-09398-9
6. Dworkin, M.J., et al.: Advanced encryption standard (AES) (2001). https://doi.org/10.6028/NIST.FIPS.197
7. Guan, Z., Li, Y., Shang, T., Liu, J., Sun, M., Li, Y.: Implementation of SM4 on FPGA: trade-off analysis between area and speed. In: 2018 IEEE International Conference on Intelligence and Safety for Robotics (ISR), pp. 192–197 (2018). https://doi.org/10.1109/IISR.2018.8535613
8. Gui, C.Y., et al.: A survey on graph processing accelerators: challenges and opportunities. J. Comput. Sci. Technol. **34**, 339–371 (2019)
9. Guo, X., El-Hadedy, M., Mosanu, S., Wei, X., Skadron, K., Stan, M.R.: Agile-AES: Implementation of configurable AES primitive with agile design approach. Integration **85**, 87–96 (2022)
10. Harb, S., Ahmad, M.O., Swamy, M.N.S.: A high-speed FPGA implementation of AES for large scale embedded systems and its applications. In: 2022 13th International Conference on Information and Communication Systems (ICICS), pp. 59–64 (2022). https://doi.org/10.1109/ICICS55353.2022.9811140
11. Information technology - Security techniques - Encryption algorithms - Part 3: Block ciphers - Amendment 1: SM4. Standard, ISO/IEC 18033–3:2010/Amd 1:2021 (2021)
12. Kumar, T.M., Reddy, K.S., Rinaldi, S., Parameshachari, B.D., Arunachalam, K.: A low area high speed FPGA implementation of AES architecture for cryptography application. Electronics **10**(16) (2021). https://doi.org/10.3390/electronics10162023. https://www.mdpi.com/2079-9292/10/16/2023
13. Liu, Q., Xu, Z., Yuan, Y.: A 66.1 GBPS single-pipeline AES on FPGA. In: 2013 International Conference on Field-Programmable Technology (FPT), pp. 378–381 (2013). https://doi.org/10.1109/FPT.2013.6718392

14. Mansouri, Y., Babar, M.A.: A review of edge computing: features and resource virtualization. J. Parallel Distrib. Comput. **150**, 155–183 (2021). https://doi.org/10.1016/j.jpdc.2020.12.015. https://www.sciencedirect.com/science/article/pii/S0743731520304317
15. Maximov, A., Ekdahl, P.: New circuit minimization techniques for smaller and faster AES SBoxes. IACR Trans. Crypt. Hardw. Embed. Syst. **2019**(4), 91–125 (2019). https://doi.org/10.13154/tches.v2019.i4.91-125. https://tches.iacr.org/index.php/TCHES/article/view/8346
16. Oukili, S., Bri, S.: High speed efficient advanced encryption standard implementation. In: 2017 International Symposium on Networks, Computers and Communications (ISNCC), pp. 1–4 (2017). https://doi.org/10.1109/ISNCC.2017.8071975
17. Peccerillo, B., Mannino, M., Mondelli, A., Bartolini, S.: A survey on hardware accelerators: taxonomy, trends, challenges, and perspectives. J. Syst. Architect. **129**, 102561 (2022). https://doi.org/10.1016/j.sysarc.2022.102561
18. Rautakoura, A., Hämäläinen, T.: Does SOC hardware development become agile by saying so: a literature review and mapping study. ACM Trans. Embed. Comput. Syst. **22**(3) (2023). https://doi.org/10.1145/3578554
19. Shahbazi, K., Ko, S.B.: High throughput and area-efficient FPGA implementation of AES for high-traffic applications. IET Comput. Digit. Tech. **14**(6), 344–352 (2020)
20. Shang, M., Zhang, Q., Liu, Z., Xiang, J., Jing, J.: An ultra-compact hardware implementation of SMS4. In: 2014 IIAI 3rd International Conference on Advanced Applied Informatics, pp. 86–90 (2014). https://doi.org/10.1109/IIAI-AAI.2014.28
21. Ueno, R., et al.: High throughput/gate AES hardware architectures based on datapath compression. IEEE Trans. Comput. **69**(4), 534–548 (2020). https://doi.org/10.1109/TC.2019.2957355