SecureComm: A Secure Data Transfer Framework for Neural Network Inference on CPU-FPGA Heterogeneous Edge Devices

Tian Chen, Yu-an Tan, Chunying Li, Zheng Zhang, Weizhi Meng, Yuanzhang Li*

***Abstract*—With the increasing popularity of heterogeneous computing systems in Artificial Intelligence (AI) applications, ensuring the confidentiality and integrity of sensitive data transferred between different elements has become a critical challenge. In this paper, we propose an enhanced security framework called SecureComm to protect data transfer between ARM CPU and FPGA through DDR on CPU-FPGA heterogeneous platforms. SecureComm extends the SM4 crypto module by incorporating a proposed Message Authentication Code (MAC) to ensure data confidentiality and integrity. It also constructs smart queues in the shared memory of DDR, which work in conjunction with the designed protocols to help schedule data flow and facilitate flexible adaptation to various AI tasks with different data scales. Furthermore, some of the hardware modules of SecureComm are improved and encapsulated as independent IPs to increase their versatility beyond the scope of this paper. We implemented several ARM CPU-FPGA collaborative AI applications to justify the security and evaluate the timing overhead of SecureComm, and we also deploy SecureComm to non-AI tasks to demonstrate its versatility, ultimately offering suggestions for its use in tasks of varying data scales.**

***Index Terms*—Heterogeneous system, Edge devices, Data transfer, DDR, SM4, Message Authentication Code (MAC)**

I. Introduction

THE heterogeneous system, which integrates various processing elements such as CPUs, GPUs, DSPs, and FPGAs, plays a critical role in AI applications. The use of heterogeneous computing systems allows for efficient utilization of computational resources, improved performance, and flexibility to adapt to specific AI workloads, including real-time Neural Network (NN) inference. In heterogeneous systems, CPU-FPGA is one of the most popular architectures of edge devices. Xilinx and Intel have released various CPU-FPGA systems, such as Xilinx MPSoC [1] and Intel Cyclone V SoCs [2]. Also, FPGAs have been deployed in commercial cloud computing systems such as Microsoft Azure [3] and Amazon AWS [4] to accelerate computation-intensive tasks.

Although the CPU-FPGA architecture offers performance benefits for AI workloads, it faces significant security challenges. Much sensitive and secret data such as user biometric information are stored, manipulated, and exchanged between CPU and FPGA, which suffers from malicious attacks [5], [6]. The CPU, as the controlling center,has access to the physical resources of the system, allowing running illegitimate processes to generate malicious requests [7]–[9]. Sensitive information can be extracted from the memory, leading to risks such as leakage of user privacy data. [10]–[12]. On the FPGA side, previous work [13], [14] shows that a hardware trojan implemented in FPGA can compromise the Advanced eXtensible Interface (AXI) bus communication, even the entire system, since all data goes through the AXI bus and IP cores. Therefore, it is crucial to monitor and protect communications between CPU and FPGA to safeguard sensitive data, prevent unauthorized tampering, and defend against attacks before system corruption.

Trusted Execution Environments (TEEs), such as ARM TrustZone, have been developed to enable hardware-based enclaves for protecting trusted applications and sensitive data [15]. However, these TEEs cannot be directly applied to heterogeneous systems to safeguard data communication among different computing units. Recently, some researchers have been aiming to extend TEEs to protect commnunication between CPU and FPGA [16], [17] and constructing extended TEE. However, TEE-based solutions need to rely on the security features provided by the hardware and the firmware [18]. Moreover, lots of TEE-based solutions entail sophisticated modifications of hardware and have not been evaluated on real hardware.

Apart from constructing extended TEEs, some other researchers have concentrated on securing bus communication among different modules. [19], [20] have focused on detecting malicious components of the hardware to secure bus communication. The research in [21] checks and monitors the bus with specific security rules. Encrypting the bus communications of microprocessors [22]–[25], peripheral [26]–[28] using lightweight stream ciphers are also a strategy to secure systems. Most solutions focused only on encryption and decryption of the bus data, without considering data

Tian Chen is with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, 100081, China (email: chentian20@bit.edu.cn).

Yu-an Tan is with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, 100081, China (email: tan2008@bit.edu.cn).

Chunying Li is with the School of Computer Science and Guangdong Provincial Key Laboratory of Intellectual Property & Big Data, Guangdong Polytechnic Normal University, Guangzhou, 510665, China (email: ChunyingL@gpnu.edu.cn).

Zheng Zhang is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, 100081, China (email: zhangzheng@bit.edu.cn).

Weizhi Meng is with the Department of Applied Mathematics and Computer Science, Technical University of Denmark, Kongens Lyngby, 2800, Denmark (email: weme@dtu.dk).

Yuanzhang Li is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, 100081, China (email: popular@bit.edu.cn).

transmission protocols and performance optimization for tasks with different data scales. In real-time NN inference, the time overhead brought by security measures is critical, making performance optimization essential. Additionally, some solutions do not consider data integrity, which limits their comprehensiveness and requires further improvement.

In this paper, we proposed an flexible, efficient and enhanced security framework, SecureComm on the CPU-FPGA heterogeneous devices for NN inference to protect the sensitive data communication between the ARM CPU and FPGA through DDR. The main contributions of this work are summarized as follows.

- We proposed a transmission protocol to schedule data flow and established smart queues in the shared buffers of SecureComm, where elements can be of variable length and non-continuous. This enables flexible adaptation to various AI tasks with different data scales, depending on the type of data transferred to and from the neural networks.
- We fully optimized an SM4 stream cipher core as the crypto module on the FPGA side, collaborating with the SM4 crypto function provided by OpenSSL to protect data flows between the CPU and FPGA through DDR. Additionally, we proposed and incorporated a MAC with SM4 to ensure data integrity and detect replay attacks. The SM4 crypto is encapsulated as an independent, fully functional crypto IP, making it suitable for various scenarios beyond this paper.
- We implemented several ARM CPU-FPGA collaborative AI applications, such as VGG, ResNet, MobileNet, and YOLO, to justify the security of SecureComm and measure the timing overhead. We also deploy SecureComm to non-AI tasks to demonstrate its versatility. Finally, based on the experimental results, we provided suggestions for applying SecureComm to tasks with different data scales.

The rest of the paper is organized as follows. Section II introduces the related work and provides the preliminary of SM4 algorithm. Section III gives a detailed introduction to our proposed framework, including the threat model, data exchange framework SecureComm, the corresponding protocols, and the optimizations in the framework. Section IV shows the security analysis and the performance of our framework. Section V concludes this paper.

II. Background

A. Related Work

Several research works have focused on protecting data confidentiality and integrity on heterogeneous systems.

(1) **Encryption**. Fangyong Hou et al. [24] applied AES-GCM to make data encryption and authentication for both the shared bus and the shared memory of a multi-processor system. Thomas Hiscock et al. [23] achieved instruction-level encryption using Trivium stream ciphers to protect data confidentiality on a SoC system. Jesús Lázaro et al. [29] proposed a novel electronic core using Trivium stream cipher to encrypt and decrypt traffic between two digital modules through an AXI connection. Lopez et al. [30] secured internal communication of a heterogeneous SoC by implementing two lightweight stream ciphers without excessive overheads. Compared to the above schemes, SecureComm not only focuses on encryption for data confidentiality but also extends the SM4 method with verification, cooperating with protocols to schedule data flow between CPU and FPGA, and fend off data tampering and replay attacks on CPU-FPGA platforms.

(2) **TEEs**. Besides applying encryption, some researchers constructed or extended TEEs to enhance protection. On heterogeneous accelerators, [31]–[33] extended the CPU TEE to develop GPU TEEs. J. Zhu et al. [34] and R. Bahmani et al. proposed HETEE and CURE [35] TEE frameworks to protect various heterogeneous components and their communications. Ke Xia et al. [16] proposed SGX-FPGA, a trusted hardware isolation path by bridging SGX enclaves and FPGAs in the heterogeneous CPU-FPGA architecture. Different from the mentioned TEEs work, SecureComm does not rely on or extend the security features provided by TEEs to the framework or construct a new TEE, which is easy to deploy on the CPU-FPGA platforms.

B. SM4 Block Cipher

The SM4 block cipher is one of the lightweight symmetric block ciphers widely used in wireless sensor networks in China. It was released by the Chinese National Cryptography Administration (CNCA) in 2012 and became an ISO/IEC international standard in 2021. This paper implements and optimizes the SM4 block cipher as the crypto engine in SecureComm.

The SM4 block cipher mainly involves three computations: round function F, nonlinear transformation τ, and linear transformation L.

Round function F

SM4 consists of 32 rounds of non-linear iterations with both block size and key size in 128 bits. The 32 round keys are expanded from the key accordingly, each of which has 32 bits: $(rk^0, rk^1, ..., rk^{31})$, $rk^i \in \mathbb{Z}_2^{32}$, i is the nature number, whose range can be denoted as $0 \leq i \leq 32$. For round i, let input data is $(X_0^i, X_1^i, X_2^i, X_3^i) \in (\mathbb{Z}_2^{32})^4$, and the round key is $rk^i \in \mathbb{Z}_2^{32}$, then the round function is:

$$F(X_0^i, X_1^i, X_2^i, X_3^i) = X_0^i \oplus L(\tau(X_1^i \oplus X_2^i \oplus X_3^i \oplus rk^i)) \quad (1)$$

where τ is a non-linear transformation and L is a linear transformation.

Nonlinear transformation τ

Nonlinear transformation τ is an S-box lookup operation. Let the input of τ is

$$X_k^i = (a_0, a_1, a_2, a_3),\ a_i \in \mathbb{Z}_2^8 \quad (2)$$

where X_k^i is the kth 32-bit number of the input, $X_k^i \in \mathbb{Z}_2^{32}$, $k \in (0, 1, 2, 3)$, then

$$\tau(X_k^i) = (Sbox(a_0), Sbox(a_1), Sbox(a_2), Sbox(a_3)) \quad (3)$$

where $Sbox$ transforms the 8 bits input to an 8 bits output value based on the S-Box table.

Linear transformation L

Linear transformation mainly performs rotation operations. Let the input of L and is $B = \tau(X_k^i) = (b_0, b_1, b_2, b_3),\ b_i \in \mathbb{Z}_2^8$, then

$$L(B) = B \oplus (B << 2) \oplus (B << 10) \oplus (B << 18) \oplus (B << 24) \tag{4}$$

where $<<$ is left rotation.

To summarize, the encryption of SM4 block cipher is demonstrated in Algorithm 1. And decryption is similar to encryption, the only difference is using the round keys and round functions in the opposite order.

Algorithm 1: Encryption for the SM4 block cipher

Input: $rk = (rk_{31}, rk_{30}, ..., rk_0);\ X = (X_3, X_2, X_1, X_0)$
Output: $(X_{32}, X_{33}, X_{34}, X_{35})$
for $i \leftarrow 0$ **to** 31 **do**
  $X_{i+4} \leftarrow X_i \oplus L(\tau(X_1^i \oplus X_2^i \oplus X_3^i \oplus rk^i);$
end
return $(X_{32}, X_{33}, X_{34}, X_{35});$

For SM4 block cipher, a set of round keys are generated with round key expansion from the original secret key.

Round key expansion

As shown in Algorithm 2, round key expansion also involves S-Box lookup linear and non-linear transformations, where (MK_3, MK_2, MK_1, MK_0) is the key, $(rk_{31}, rk_{30}, ..., rk_0)$ are the expanded keys for Round Function F, $L'(B) = B \oplus (B << 13) \oplus (B << 23)$, and CK_i, FK_i are constant system parameters.

Algorithm 2: Round key expansion for the SM4 block cipher

Input: $(MK_3, MK_2, MK_1, MK_0),\ MK_k \in \mathbb{Z}_2^{32}$
Output: $(rk^0, rk^1, ..., rk^{31}),\ rk^i \in \mathbb{Z}_2^{32}$
$(K_3, K_2, K_1, K_0) \leftarrow (MK_3 \oplus FK_3, MK_2 \oplus FK_2, MK_1 \oplus FK_1, MK_0 \oplus FK_0)$
for $i \leftarrow 0$ **to** 31 **do**
  $rk_i \leftarrow K_{i+4} = K_i \oplus L'(\tau(K_{i+1} \oplus K_{i+2} \oplus K_{i+3} \oplus CK_i));$
end
return $(X_{32}, X_{33}, X_{34}, X_{35});$

In SecureComm, we use SM4 in fixed key mode, thus the round keys can be pre-calculated and stored in memory to eliminate the time of expanding.

III. Proposed Framework: SecureComm

A. Threat Model and Assumptions

Assume that users implement an FPGA-accelerated NN inference application consisting of an program run on an ARM CPU and an FPGA kernel. The application on ARM CPU switches data along DDR with the FPGA kernel, i.e. CPU puts the ingredients into DDR and notifies FPGA, then the FPGA fetches the data through the AXI bus and invokes the computation with logic circuits. When it finishes, it puts the results back into DDR through the AXI bus and informs the CPU to get the results. In this situation, protecting the integrity and confidentiality of the data transferred between CPU and FPGA is necessary. To achieve this goal, as shown in Figure 1, we design an enhanced data transfer framework SecureComm. In this paper, we only take the attacks on the AXI bus and the shared memory of DDR into consideration. We assume that both SecureComm and the accelerated kernel are immune to attacks; therefore, the AXI communications within SecureComm and between SecureComm and the accelerator kernel are considered secure. On the CPU side, we consider strong adversaries who can dump and tamper with the shared memory of CPU and FPGA on DDR. However, it cannot compromise the modules of SecureComm and the user APP, thus the normal communications will not be affected. On the FPGA side, we postulate that adversaries can plant malicious IPs to eavesdrop the AXI channels between CPU and SecureComm. But it cannot tamper with the contents on the channels. Moreover, it can perform arbitrary reads and writes on DDR through a self-established AXI channel out of SecureComm to manipulate the contents of the memory.

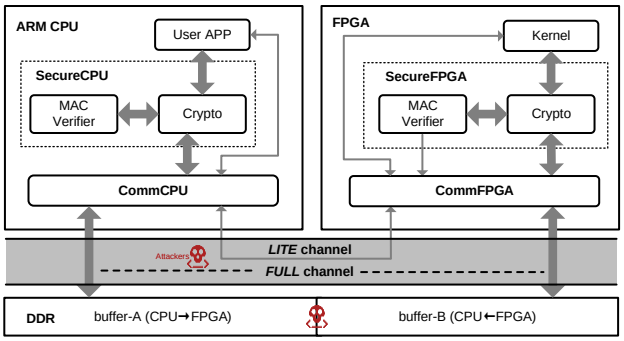


Fig. 1. The overview of SecureComm, where the thick arrow represents the data stream of large data that needs to be written into DDR or read from DDR through the *FULL* Channel, while the thin arrow represents the data stream of small data such as parameters that need to be transmitted through the *LITE* Channels

B. Architecture

Figure 1 shows the block diagram of our proposed framework SecureComm. SecureComm consists of three main components, ARM CPU, FPGA, and communication channels. On the ARM side, there are a SecureCPU module and a CommCPU module. In the SecureCPU module, the MAC verifier is responsible for MAC calculation and integrity verification, and the Crypto module takes charge of encrypting and decrypting data which sends to and received from FPGA through DDR. The CommCPU module includes drivers for transferring data directly or indirectly (through DDR) between CPU and FPGA. It can follow the parameters sent from FPGA and schedule the data flow of a task through the communication channels. On the FPGA side, similarly, the hardware SecureFPGA module is responsible for data integrity verification and data encryption/decryption. In our designs, we pick the SM4 algorithm as the crypto algorithm for demonstration and implementation. Noted that it can be replaced with

other cryptographic algorithms such as AES-GCM in practice. The hardware CommFPGA module is designed to generate AXI bus signals for accessing DDR memory and handles the data on the *LITE* channels between CPU and FPGA in collaboration with the driver in CommCPU.

As for data transferring, SecureComm provides two kinds of communication channels. One is the *FULL* channel for transferring large-scale data between CPU and DDR, DDR and FPGA. Compared to FIFO and BRAM, DDR provides a large amount of memory with high data transfer rates. Also, it is more flexible for users to design data flow. The other channels are the *LITE* channels, which are for transmitting address, nonce, and some other small-size configurations and parameters between CPU and FPGA directly without DDR. As the data exchange center, the shared memory of DDR is divided into two parts, involving deposit data from CPU to FPGA and FPGA to CPU respectively. When a host APP needs to communicate with an FPGA kernel, it needs to store the data in DDR through the *FULL* channel and notify FPGA of the configurations and parameters of the transfer through the *LITE* channels. Then the kernel loads the data from DDR through the *FULL* channel based on the parameters and executes the computation. After finishing, it stores the result back to DDR and informs CPU. In SecureComm, we allocate two shared buffers in DDR, one is for storing data from CPU to FPGA, and the other is for storing data from FPGA to CPU. Before writing data frames to these two buffers, the parameters of the data frames, including the starting address and length, are first sent to the receiver. Therefore, the two buffers can be considered as two circular queues, where the elements, i.e., the data frames, do not need to be contiguous or of the same size. Both the sender and receiver use the rear and front to determine the status of elements in the queue. Users can adjust the storage address of the data according to their needs, making it highly flexible. The head and tail of the queue are updated through *LITE* channels. To ensure the interoperability between different hardware and software components of SecureComm, we propose protocols to define a standardized way of communication, which will be introduced in detail in the next part.

C. Protocols

Based on the above architecture, we proposed protocols that defend confidentiality, integrity and also offer protection against replay attacks on transfers between CPU and DDR as well as DDR and FPGA. The proposed protocols can mainly be divided into the following two procedures according to the directions of the data flow.

1) CPU → DDR → FPGA: Figure 2 illustrates the protected transfer from CPU to FPGA through DDR that executes in three phases. 1. **Construct and encrypt a data frame.** SecureCPU first generates a nonce and computes the MAC of the data from a user APP in the MAC verifier according to Equation 6-9. Then it constructs a data frame, whose data structure is shown in Figure 3, and encrypts the data frame with the Crypto module. 2. **Data transfer.** CommCPU first checks the value of the front index to determine if the

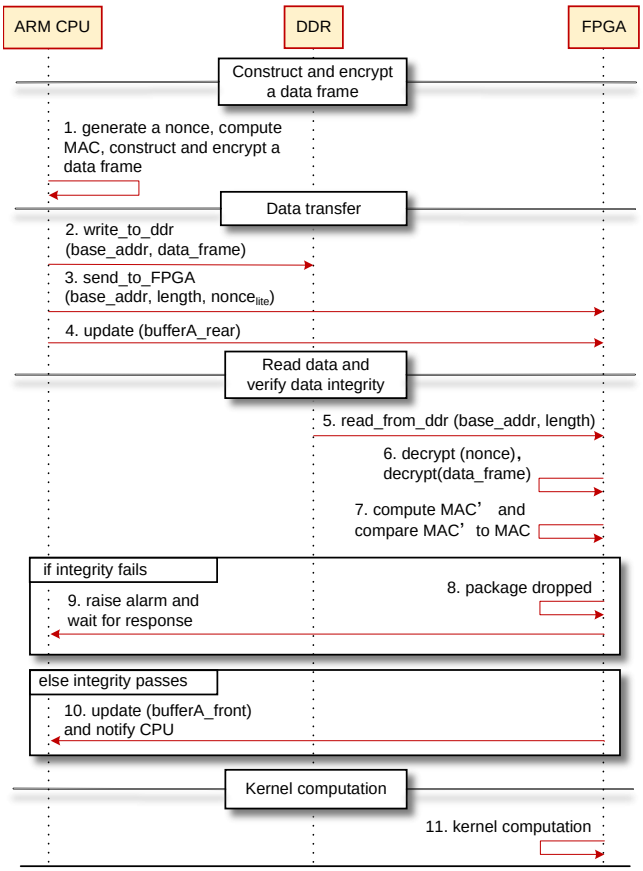


Fig. 2. Protected transfer from CPU to FPGA through DDR

FPGA has read the data frame. If the value of front changes, it means that the corresponding space can be reclaimed. Then, CommCPU searches for an available area in bufferA and writes the data frame into it. Afterwards, it sends the parameters, including base address, length, and the sliced nonce to CommFPGA directly through the LITE channels, and updates the rear index of bufferA with the formula $(rear = rear + 1)\%MAX\_SIZE$, where MAX_SIZE denotes the maximum number of data frames that can be stored in the buffer. 3. **Read data and verify data integrity.** When bufferA is not empty, CommFPGA first reads the data frame from bufferA through the *FULL* channel according to the parameters received in the previous step. Then the hardware Crypto and MAC verifier of SecureFPGA decrypt the data frame and compute the MAC. Finally, the verifier compares the calculated MAC against the MAC value stored in the data frame. If it is different, i.e., integrity fails, SecureFPGA drops the frame, sets a fail flag, and raises an alarm to the User APP through CommFPGA and CommCPU. If the integrity passes, the front of bufferA will be updated by CommFPGA with the formula $(front = front + 1)\%MAX\_SIZE$. Then the decrypted data frame will be delivered to the kernel in FPGA for further computation and the User APP will get the notification of the success of verification.

2) FPGA → DDR → CPU: After the kernel finishes computation, it needs to store the result back to DDR. As shown in Figure 4, similar to Procedure 1, the protected transfer

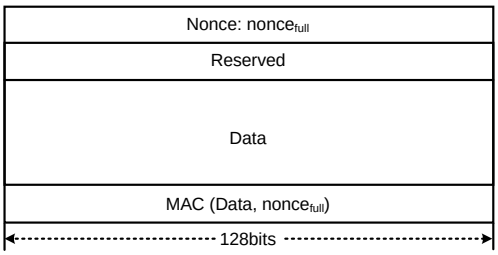


Fig. 3. The data structure of a data frame

also executes in three phases. 1. **Construct and encrypt a data frame.** Since the nonce is encrypted when transferring, the attackers will not get the plaintext with eavesdropping. Thus, SecureFPGA does not need to generate a new nonce for data transfer. Before constructing a result data frame, the MAC verifier of SecureComm needs to compute MAC with Equation 6-9 and the nonce sent from CPU. After constructing the frame, the Crypto module encrypts it and delivers to CommFPGA. 2. **Data transfer.** CommFPGA first reclaims the memory that CPU has read and generates the AXI bus signals to write the data frame to bufferB of the shared memory. Then, it updates the rear index of bufferB and sends the base address, and the length of the data frame to CommCPU through the *LITE* channel. 3. **Read data and verify data integrity.** When bufferB is not empty, SecureCPU reads the data frame from bufferB through CommCPU and decrypts it with the sealed key. Then the MAC verifier verifies the integrity of the data by comparing the MAC value calculated by the host against the MAC value stored in the data frame. If the integrity passes, the front of bufferB will be updated through CommCPU and the *LITE* channels. If the integrity fails, the MAC verifier will raise an alarm to the user and waits for the user's response.

By following the aforementioned protocols, CPU and FPGA can establish reliable and efficient communication and collaboration. Moreover, constructing queues on the shared buffers and sending parameters including base address and length to the receiver allows users to arrange data flow conveniently and flexibly. Noted that, for a clear understanding of the data flow, the above demonstration is serial. In our framework, some phases of the procedure are paralleling and pipelining to improve the efficiency and the latency of SecureComm.

In SecureComm, the modules of the ARM CPU and FPGA are designed and implemented as software and hardware respectively. Figure 7 demonstrates the hardware design and data flow of SecureComm on the FPGA side. To decouple the timing requirements of different IP blocks, we utilize FIFOs in SecureComm to connect with modules with different clock domains. The designs for each module and the optimizations will be discussed below.

D. CommCPU and CommFPGA

In SecureComm, CommCPU and CommFPGA are designed to intercommunicate through the *LITE* channels and access DDR through the *FULL* channel.

1) Intercommunication: On the FPGA side, in CommFPGA, we use AXI-LITE bus to construct *LITE* channels to

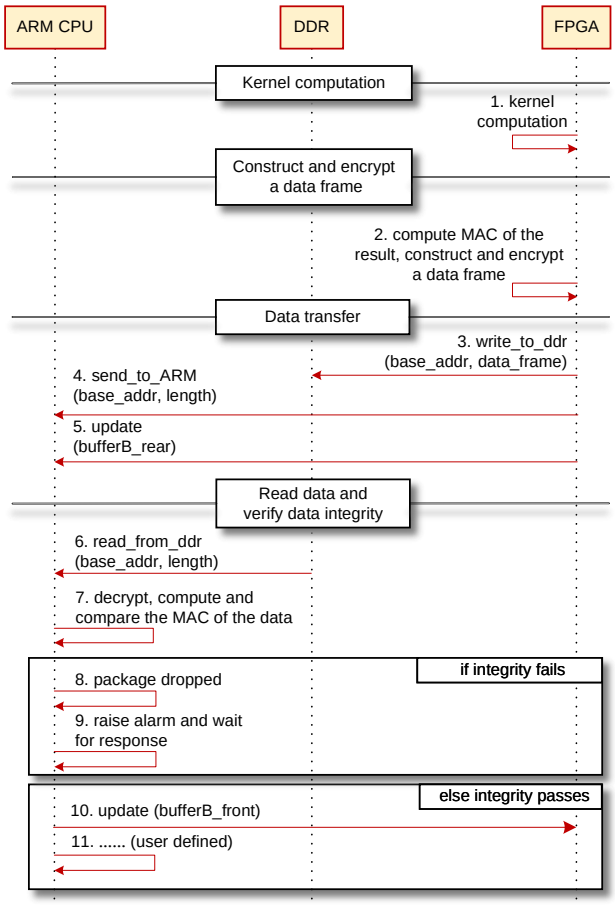


Fig. 4. Protected transfer from FPGA to CPU through DDR

communicate with CommCPU. The data width of the channels is configured to 32 bits. On the CPU side, an extended driver for gpiochips to read and write multi-bit ports is implemented in CommCPU to interact with AXI GPIOs of CommFPGA. SecureComm provides four separate *LITE* channels for transmitting different messages, which are shown in Table I. As for Channel 1 and 2, FPGA or CPU parses data according to the highest 3 bits and extracts the corresponding messages. Channel 1 delivers the message from CPU to FPGA, while Channel 2 transmits the message from FPGA to CPU. The parameters delivered on Channel 1 are extracted and kept temporarily in a FIFO of CommFPGA, while the parameters sent from FPGA on Channel 2 are extracted and saved in the allocated memory of CommCPU. As for Channel 3, the rear (16 bits) of bufferA and the front (16 bits) of bufferB are concatenated to a 32 bits number. To avoid data race, these two variables are maintained by CPU, i.e., only CPU can modify their values, while the FPGA can only read them. As for Channel 4, similarly, the front (16 bits) of bufferA and the rear(16 bits) of bufferB are concatenated to a 32 bits number. Only FPGA can modify the value.

To detect replay attacks, we construct a nonce pair

$$(nonce_{lite}, nonce_{full}) = (N[28:0], encrypt(N)) \quad (5)$$

to involve in MAC calculation, where N is a nonce, $nonce_{lite}$ is the 0th bit to 28th bit of N which is transferred on Channel

TABLE I
THE MESSAGE DELIVERED ON DIFFERENT *LITE* CHANNELS.

Channel	Data(32bit)		Message
1 (CPU→FPGA)	001		base_addr_H
	010		base_addr_L
	011	XXXX...XXXX	data_len
	100		nonce[28:0]
	...		reserved
2 (FPGA→CPU)	001		base_addr_H
	010	XXXX...XXXX	base_addr_L
	011		data_len
	111	0000...0000	integrity passed
		0000...0001	integrity failed
	...	XXXX...XXXX	reserved
3	XXXX...XXXX (bufferA_rear)	XXXX...XXXX (bufferB_front)	CPU maintains
4	XXXX...XXXX (bufferA_front)	XXXX...XXXX (bufferB_rear)	FPGA maintains

1, and $nonce_{full}$ is the 128 bits encrypted N which is packed in the data frame. When SecureFPGA received a data frame, it encrypts and extracts the complete 128 bits nonce N, and compares it to $nonce_{lite}$. If it is matched, the MAC verifier of SecureFPGA continues to verify MAC.

2) Accessing the Shared Memory of DDR: On the FPGA side, the parameters through the LITE Channel 1 from CPU are extracted in the CommFPGA. As shown in Figure 7, to access the shared memory of DDR based on the parameters, we design and encapsulate an IP named FDMA integrated into CommFPGA to generate AXI bus signals. FDMA encapsulates the AXI-FULL bus interface and defines a simple APP interface for users to use the AXI4 bus for data transfer flexibly and conveniently.

On the CPU side, since DDR is managed by the kernel through its memory management subsystem, to release the control of the shared buffer of DDR, we set up a device tree to define the reserved memory area of DDR. Meanwhile, we design and implement a memory driver integrated into CommCPU to manage the shared buffer in the user applications, offering APIs for reading and writing to the shared memory.

E. High-performance Crypto

SecureFPGA and SecureCPU are the most significant modules of SecureComm designed to protect the confidentiality and integrity of the transferred data. On the FPGA side, we design and implement a high-performance SM4 Crypto module. Compared to the SM4 implementations in [36], [37], and [38], we not only implement an encryption module for encryption, but we also develop a specific module for key inversion to invert the round key and reuse the encryption module for decryption. In addition to the above two modules, we design a top module to schedule data flow for encryption or decryption. The Crypto in SecureFPGA works in fixed-key mode, and the fixed encryption round keys are sealed in the top module and passed directly to the encryption module for both encryption and decryption. The installation and the protection of the key are out of the scope of this paper. In the case of decryption, the round key needs to be put into the key inversion module first to obtain the inverted key, which is then passed to the encryption module for data decryption.

The core of the SM4 is 32 rounds of nonlinear iterations. As shown in Figure 5, in the encryption module, we use registers to divide critical paths and store the results of each iteration for the next iteration. All combined logic blocks work in parallel within a clock cycle and construct a pipeline for encryption and decryption. Therefore, one number can be encrypted or decrypted for each clock cycle.

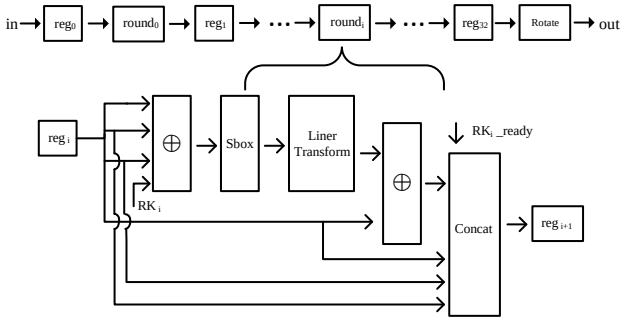


Fig. 5. The 32 rounds of nonlinear iterations of SM4

As for the key inversion module, which is involved in decryption, we use upper triangular registers to store and inverse the round keys efficiently. As shown in Figure 6, for each clock cycle, the key inversion module gets the undermost round key of a column, outputs to the decryption module in the right order, and move to the next column in the next cycle.

In our SM4 implementation, we adopt a speed-first fine-grained parallel pipeline optimization within and between modules, trading FPGA area for performance, which reduces encryption and decryption latency and improves throughput, ensuring that it will not become performance bottlenecks for SecureComm. In practical applications, the resource consumption of the FPGA can be reduced by decreasing the number of pipeline stages. However, this comes at the cost of slower encryption/decryption speed. Therefore, it is crucial to find a balance between performance and resource consumption, taking into account the specific requirements of the application.

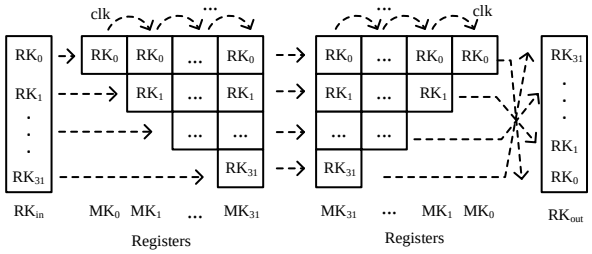


Fig. 6. Key inversion module

Although the framework uses fixed-key mode to encrypt and decrypt data, we still extend the crypto module and implement a key expansion module to support the rolling-key mode to fit in more scenarios outside this paper. Similar to the encryption module, to enable the handling of key changes without causing pipeline stalls, the key expansion module

uses additional registers to store the first four encryption round keys required for the next round of iteration for each key, then pipelines the nonlinear iterations to improve the performance. Thus 32 different round keys from different keys for encryption will be generated in each clock cycle. The rolling-key mode requires passing the key to the key expansion module to get the round key, and then sending it to the encryption module for encryption. On FPGA side, we instantiate one Crypto module for encryption and decryption. The Encryptor and Decryptor in Figure 7 are the same module, which is described separately for better distinguishing between encryption and decryption. On CPU side, we make use of the SM4 engine of OpenSSL, which uses ARM's SM4 instruction set to accelerate the SM4 crypto algorithm.

We also encapsulate our SM4 crypto as an independent and full-function crypto IP to fit in more scenarios outside this paper, including secure communication between different IPs using AXI protocols in FPGA. In situations where designers lack control over IPs that could intercept AXI communication, such as the AXI Interconnect, utilizing SM4 cryptography can safeguard the communication between the user's trusted AXI master and slave. This can be achieved by inserting an SM4 cryptographic pair between the Master and AXI Interconnect, as well as between the AXI Interconnect and Slave for encrypting and decrypting data. Even if the AXI Interconnect acts maliciously, it will be unable to access the actual message being transmitted along the bus.

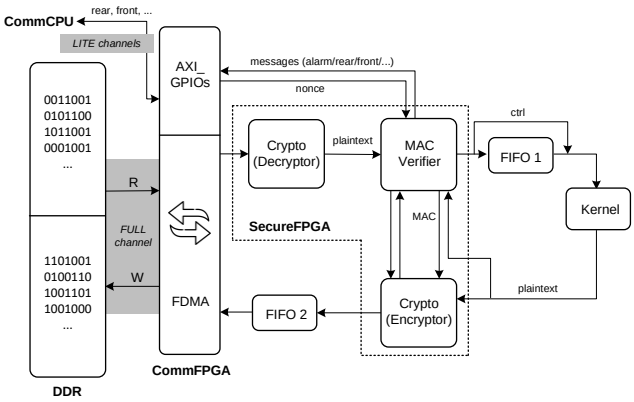


Fig. 7. The architecture and dataflow of SecureComm on the FPGA side, where Decryptor and Encryptor are the same module. The separate description is only for better distinguishing between encryption and decryption

F. MAC Verifier

In SecureComm, SM4 provides confidentiality for the data being encrypted, but it does not provide any integrity or authenticity guarantees. To ensure data integrity and authenticity, a separate MAC is proposed in conjunction with SM4. Moreover, to detect replay attacks, a nonce is introduced in the generation of MAC.

To calculate MAC, the verifier (1) pads the message to a multiple of the block size (128 bits) and divides the padded message into blocks, denoted as $M = (M_0, M_1, ..., M_{n-1})$, where $M_k \in \mathbf{Z}_2^{128},\ k \in [0, n-1]$.

(2) computes the bitwise XOR of all blocks:

$$T = M_0 \oplus M_1 \oplus ... \oplus M_{n-1} \tag{6}$$

(3) computes the bitwise XOR of T and $nonce$:

$$T' = T \oplus nonce \tag{7}$$

where $nonce$ is a random 128 bits number generated on the ARM side.

(4) encrypts T' with SM4:

$$C = encrypt(T'),\ C \in \mathbf{Z}_2^{128} \tag{8}$$

(5) expands C in 4 bits a unit. Converting each 4 bits corresponding hexadecimal to a character, then saving the corresponding hexadecimal value to complete the expansion. For example, suppose a unit is 4'b1010, which is 4'hA. The ASCII value of the character A is 65, which is 8'h41, thus the expansion of 4'b1010 is 8'h41. With the expansion, C is transformed to $C' = (C_0, C_1),\ C_k \in \mathbf{Z}_2^{128}$.

(6) does a bitwise exclusive OR operation on C_0 and C_1, encrypts and outputs the result as MAC:

$$MAC = encrypt(C_0 \oplus C_1) \tag{9}$$

To verify the integrity of the transferred data and detect replay attacks, the framework introduces nonce pairs and adapts to the calculation of MAC. On both sides, we use Encrypt-and-MAC, E&M mode to encrypt data and calculate MAC, i.e., the encryption and MAC calculation can be parallelized to accelerate. For verification, the ciphertext needs to be decrypted first before calculating the corresponding MAC. These two operations are also pipelined in SecureFPGA to reduce the calculation latency. Both encryption/decryption and verification share the same fixed key in SecureComm. The recipient can decrypt the data frame, extract the nonce and plaintext, and compare the calculated MAC with the one stored in the frame. If the values vary, it suggests that the frame has been tampered with by attackers. The nonce introduced in MAC generation provides randomness, thereby effectively detecting replay attacks.

On the FPGA side of SecureComm, we utilize FIFOs to enable asynchronous operation of different IPs and simplify the design by isolating them from each other. FIFO1 in Figure 7 is used to accumulate the decrypted input of the kernel, FIFO2 is used to store the encrypted output of the kernel. As for the encryption steps within the verifier, we schedule and reuse the SM4 Crypto to reduce hardware resources. We also parallelize and pipeline steps on the FPGA size to reduce latency. For example, the output of the kernel will be sent to the verifier and the encryptor at the same time for MAC calculation and data encryption. In SecureComm, we choose a balance between resource consumption and performance. In practice, we can use more hardware resources and finer-grained parallel pipelining optimizations to further improve the performance of SecureComm.

IV. Experiment

We implemented a prototype system for SecureComm on the ALINX AXU5EV-P development board with a XAZU5EV-SFVC784-1-i chip, i.e., Xilinx Zynq UltraScale+ MPSoC, which is a highly integrated and versatile platform that combines programmable logic (FPGA) with processing subsystems based on ARM Cortex-A53 and Cortex-R5 cores. The FPGA and the processing subsystems are connected with the AXI buses. Based on the prototype system, we conducted three types of experiments to analyze the effectiveness and performance of SecureComm: (1) Security analysis, in which we analyzed the resistance of SecureComm against the threat models demonstrated in Section III. (2) Performance analysis, in which we evaluated the performance of the SM4 crypto in SecureComm with other implementations on FPGA. Moreover, we explored the impact of data size and the kernel's computational intensity on the performance of SecureComm. In addition, we evaluated the timing overhead caused by SecureComm by comparing it with the framework that disables all security protection on three benchmark applications. (3) Resource overhead, in which we evaluated the resource overhead on the FPGA side caused by the hardware modules of SecureComm.

A. Security Analysis

In this section, we evaluated the resistance against two types of attacks mentioned in Section III that come from malicious IPs on the FPGA side and malicious applications on the ARM CPU side respectively.

1) The Attacks from Malicious IPs: Because the data stored in and retrieve from DDR are encrypted within SecureComm, and the encryption key is sealed in the CPU controller and the crypto module on FPGA, only the crypto module of FPGA can decrypt data, and transfer it to the FPGA kernel on the FPGA side. Therefore, there is no clear data on the AXI-FULL buses. Even if an attacker plants a malicious AXI Interconnect IP (or any other malicious AXI infrastructure elements) to intercept the data on the AXI bus between SecureComm and the CPU, they would be unable to access the sensitive data without the encryption key. If a malicious IP overwrites the data in the shared memory via a self-established channel outside of SecureComm, it will be unable to decrypt and access the nonce contained in the data frame necessary to compute the corresponding MAC for the tampered data. Consequently, this leads to a failure in the integrity verification process of SecureComm.

2) The Attacks from Malicious Applications: Given that the data stored in the shared memory is securely encrypted, any information extracted by malicious applications will only appear as ciphertext. These malicious applications lack the decryption key, rendering them unable to decipher the sensitive data. In the event that a malicious application endeavors to manipulate the sensitive data within the shared memory, it will face obstacles in decrypting the data to access the nonce necessary for falsifying the corresponding MAC. Consequently, any alterations made to the shared memory will be identified by the MAC verifier on the FPGA. Moreover, in cases where a malicious application tries a replay attack, any tampering with the nonce transmitted through the *LITE* channel will result in discrepancies in the nonce pair elements, prompting the MAC verifier to reject the transmission.

TABLE II
Resource usage and performance of our SM4 implementation.

Reports		**SM4 Crypto of SecureFPGA**
Resources Usage	LUTs	7171
	FFs	8643
Max Frequency (MHz)		462
Throughput (Gb/s)		59.14

To validate our claim, we simulated two test cases, where we compromised the OS to perform replay attacks and planted a malicious hardware module on the FPGA to use random numbers to overwrite the data in the shared memory. In both scenarios, we observed that SecureComm successfully detected the tampering of data and denied the transmission, which proves that it can defend the attacks from malicious IPs and malicious applications, protecting the integrity and confidentiality of the sensitive data stored in the shared memory.

B. Performance Analysis

1) SM4 Crypto: We evaluated the resource usage, maximum operating frequency, and throughput of our SM4 crypto implementation and compared it with other SM4 implementations on FPGA. As shown in Table II, our implementation greatly optimizes timing by inserting a large number of registers and dividing critical paths, resulting in a higher frequency of 462MHz and an improved throughput of 59.14Gb/s with limited resource overhead. In our SM4 implementation, we adopts a speed-first fine-grained parallelism to achieve high-performance encryption and decryption. In scenarios with limited resources, the parallelism can be reduced to decrease resource consumption. In SecureComm, the crypto core operates in 150MHz to work efficiently with other modules.

2) Runtime Timing Evaluation:

① The exploration of the impact of data size and the kernel computational intensity.

In SecureComm, the data size (both input and output) and the computational intensity of the kernel are two decisive factors in determining computation time. To evaluate the influence of the data size, we implemented the calculation of the square of a matrix as the kernel using High-level Synthesis (HLS). We fixed the compute matrix size to 56x56 (int), and adjusted the input and output size to measure the additional timing overhead of one loop (ARM CPU→FPGA kernel→ARM CPU). If the input size was larger than 56x56, we only took 56x56 of the input data and the rest was discarded. As shown in Figure IV-B2, we observed that the timing overhead posed by the framework increased approximately linearly with the input/output data size. That is because data encryption, decryption, mac calculation, and data transfer contribute significantly to the time, which highly depends on the data size. On average, the framework incurred an overhead

TABLE III
TIMING EVALUATION OF MAC GENERATION AND VERIFICATION, WHERE ENC DENOTES ENCRYPTION, ENC + MAC DENOTES ENCRYPTION AND MAC GENERATION, DEC DENOTES DECRYPTION AND DEC + MAC DENOTES DECRYPTION AND MAC VERIFICATION.

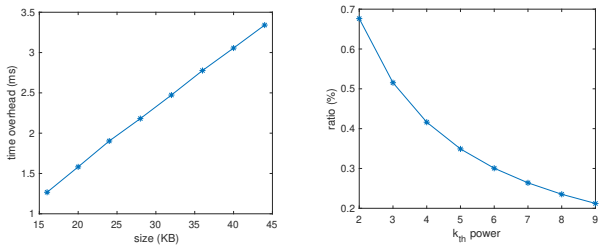
Size	CPU(ms)				FPGA(*10-5 ms)			
	Enc	Enc + MAC	Dec	Dec + MAC	Enc	Enc + MAC	Dec	Dec + MAC
16KB	0.641	0.677	0.564	0.595	0.705	0.733	0.727	1.461
32KB	0.660	0.697	0.568	0.597	1.388	1.416	1.410	2.826
48KB	1.769	1.856	1.695	1.777	2.071	2.099	2.093	4.191

TABLE IV
TIMING EVALUATION USING TWO AI APPLICATIONS AND TWO NON-AI APPLICATIONS (MS), WHERE SC. DENOTES THE CASE THAT APPLIES SECURECOMM, AND BASELINE DENOTES THE CASE WITHOUT THE SECURITY PROTECTION

Apps	16KB		32KB		48KB		64KB		80KB	
	Baseline	SC.	Baseline	SC.	Baseline	SC.	Baseline	SC.	Baseline	SC.
LeNet	12.7889	14.0909	26.5818	27.8449	37.9977	41.6418	50.5886	55.3897	63.2395	69.2046
VGG-5	63.1393	64.4194	127.2107	128.5017	188.9600	192.6031	251.9163	256.7474	314.9027	320.9078
K-means	1.9112	3.1842	4.6623	5.9624	4.7954	8.4005	6.4806	11.2547	7.9658	13.9429
Smith Waterman	70.2251	71.4971	148.4983	149.7993	224.3924	228.0254	294.4725	299.2916	371.5896	377.5337

of 0.074ms with an increase of 1KB in data size for both input and output. We also evaluate the impact of MAC generation and verification in SecureComm. As shown in Table III, MAC generation and verification do not significantly pose extra time overhead compared to pure encryption and decryption. The influence brought by MAC on system performance is negligible.

To evaluate the influence of the computational intensity of the kernel, we set the size of the input/output matrix to 16KB (56x56) and adjusted the computational intensity by performing matrix power operations with different exponents. Noted that no additional optimization was applied to the HLS program, so the computational intensity increased approximately linearly with the exponent. As shown in Figure IV-B2, the additional time overhead generated by SecureComm decreases inversely with the exponent, as a proportion of the total time. In practice, it is necessary to adjust the data size and kernel complexity based on the actual need to obtain better performance.



(a) The additional time overhead with different input/output size.

(b) The proportion of additional time overhead to total time with different exponents.

Fig. 8. The impact of data size and the kernel computational intensity

② The evaluation of the runtime performance of SecureComm.

To evaluate the overall performance of SecureComm, we adopted two AI CPU-FPGA applications, LeNet [39] and VGG-5 [40] with different input and output sizes. Also, we implemented two non-AI applications, K-means [41] and Smith Waterman [42], which demonstrate the versatility of our framework. In the experiment, the kernel computed all the input data in a loop. All cases are conducted 20 times to reduce random errors, and the result of each case is the average of the measurements. Table IV shows the results comparing SecureComm with the baseline where there is no security protection for the CPU-FPGA applications. As we can see, for low-computation programs like K-means, the total computation time after applying the SecureComm framework is still short and acceptable. For computation-intensive applications like LeNet, VGG-5 and Smith Waterman, the additional overhead is negligible.

To evaluate the runtime performance of SecureComm with large size of input data, we employed SecureComm to protect the data streams of YOLOv4, which is illustrated in Figure 9. Also, we implement three NNs, namely Inception-v1, ResNet50 and MobileNet-v2, and compare their performance with and without employing SecureComm. As shown in Table V, we tested three YOLO tasks: vehicle recognition, pedestrian recognition, and concrete crack detection. We compared the average time for 50 frames and the frame rate before and after applying SecureComm. It is evident that the inference time significantly surpasses the additional time introduced by SecureComm, leading to a slight decrease in frame rate. Regrettably, due to the limited CPU performance and FPGA resources of our development board, real-time inference cannot be achieved, with a frame rate of approximately 3 fps regardless of SecureComm deployment. In our experiments, the large input image contributes to a predominant SM4 encryption time on the CPU within SecureComm's cryptographic operations, averaging around 37 ms per image.

Theoretically, employing a more powerful SoC, such as the

TABLE V
PERFORMANCE EVALUATION OF THREE YOLOV4 TASKS AND THREE NNS, WHERE SC. DENOTES THE CASE WITH SECURECOMM, AND BASELINE DENOTES THE CASE WITHOUT SECURITY PROTECTION.

Model		Input Size (N, H, W, C)	Time (ms)		Speed (fps)		Time Overhead / Time (SC.)
			Baseline	SC.	Baseline	SC.	
YOLO v4	Vehicle	(1, 608, 608, 3)	274.8669	311.4089	3.45	3.06	0.1173
	Pedestrian		306.6804	342.4513	3.26	2.92	0.1045
	Concrete		289.8551	326.8880	3.45	3.06	0.1133
Inception-v1		(1, 224, 224, 3)	21.5998	26.5102	46.30	37.72	0.1852
ResNet50			52.5805	57.8929	19.02	17.27	0.0918
MobileNet-v2			13.6570	18.5909	73.22	53.79	0.2654

Xilinx Versal SoCs, along with further accelerating the inference process through fine-grained parallelism, could lead to a greater reduction in the performance impact of SecureComm. For neural networks like MobileNet and Inception, which have short inference times, the deployment of SecureComm may decrease system performance; however, this is still acceptable and meets real-time requirements.

Overall, with small input and output data scales, SecureComm does not cause significant performance loss. However, with large input and output data scales, SecureComm may lead to an overall performance decrease due to the long encryption and decryption times on the CPU side. This issue can be mitigated through approaches such as and employing a more powerful SoC or using encryption algorithms that are faster or better supported by hardware/instruction set acceleration.

C. Resources Overhead

We further evaluated the resource overhead of SecureComm on FPGA, considering the Crypto, MAC verifier and CommFPGA modules as the three major components on the FPGA side. Table VI demonstrates the resource consumption of the Crypto, MAC verifier, and CommFPGA implementations in terms of flip-flops (FFs), lookup tables (LUTs), block RAMs (BRAMs) and lookup table RAMs (LUTRAMs) reported in Vivado. The results indicate that SecureComm is lightweight, from which the crypto module takes up most of the resources. This is because we have optimized the SM4 module in a very fine-grained way to maximize performance. In practice, in some cases where performance requirements are not high, the optimization can be appropriately adapted according to specific needs, thereby reducing resource usage.

Additionally, the FPGAs used in commercial scenarios typically have more hardware resources than the MPSoC used in our experiments. Therefore, due to SecureComm's consistent resource consumption across different FPGAs, its overall resource usage (as a percentage) is expected to be lower and negligible.

TABLE VI
RESOURCE USAGE OF CRYPTO, MAC AND COMM ON AXU5EV-P.

Modules	LUT	FF	BRAM	LUTRAM
Crypto (SM4)	7171	8643	0	0
MAC verifier	399	665	0	0
CommFPGA	1320	2588	8	4
Total	9811	13819	8	4
Ratio	8.38%	5.90%	5.56%	0.01%

V. CONCLUSIONS

In this paper, we proposed a security-enhanced framework called SecureComm on the CPU-FPGA heterogeneous edge devices for the protection of data confidentiality and integrity for NN inference. SecureComm uses DDR as a data transport center between ARM CPU and FPGA, supporting a larger size of data transferred compared to BRAM and FIFO. In SecureComm, we established smart queues on the shared buffer, which support elements of different sizes and non-continuous placements.

To protect data flows between the ARM CPU and FPGA through DDR, we implemented and optimized an SM4 crypto to encrypt and decrypt data streams to ensure confidentiality. Moreover, the SM4 crypto is encapsulated as an independent and full-function crypto IP to fit in more scenarios outside this paper, supporting high-performance encryption and decryption. Collaborated with security-enhanced protocols and a MAC verifier module, our framework can detect data tampering and replay attacks, thus protecting data integrity.

Meanwhile, our evaluation of applying SecureComm to real-world AI and non-AI applications shows that, with small input and output data sizes, the performance impact of SecureComm can be negligible. SecureComm does not rely on any special security features and does not require extensive modifications to the system, making it easy to deploy on commodity hardware. Moreover, it can be referred to and adapted for use in other heterogeneous systems such as Intel's Cyclone SoC FPGAs, to enhance the security of chip-to-chip communication. When implementing SecureComm on resource-constrained boards, reducing the number of pipeline stages can mitigate resource consumption. In time-critical situations, substituting SM4 with faster symmetric encryption algorithms such as AES is advisable. In SecureComm, to support non-uniform and non-contiguous data frames, both the data sender and receiver need to monitor the status of the data frames written to the shared memory. The data receiver needs to read the data frames stored in the DDR, while the data sender needs to reclaim the data frames that have been read by

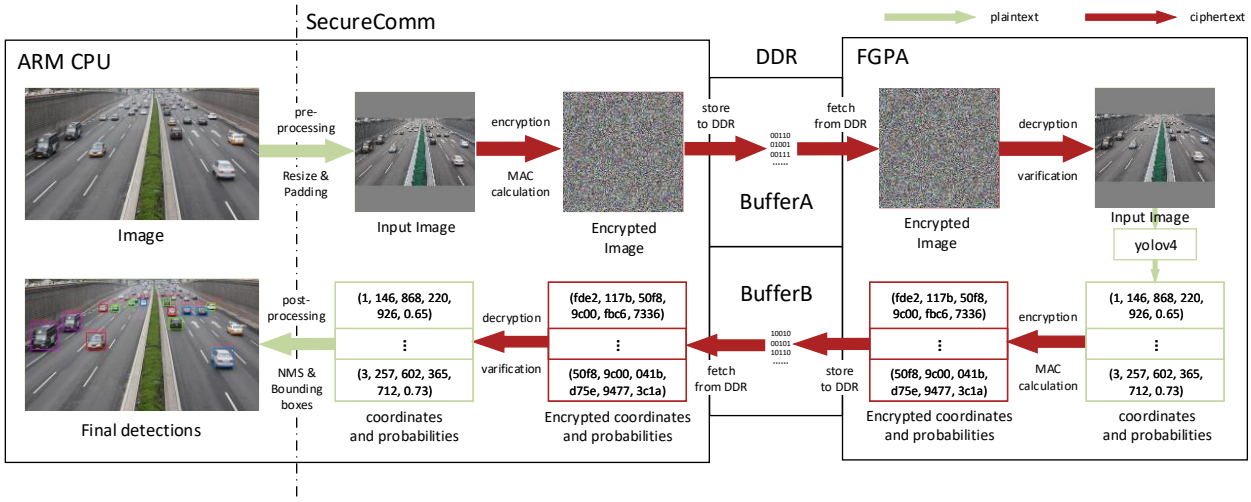


Fig. 9. Illustration of Securing YOLOv4 data flow with SecureComm

the receiver to allocate space for new data frames. In practice, enforcing fixed-length and continuous frame writing can help reduce code complexity and further improve efficiency.

In this paper, we restricted our attention to the data interaction scenario involving a singular host application and kernel. To broaden the generality and usability of SecureComm, we intend to concentrate on the data interaction involving multiple host applications and kernels in our forthcoming research. Additionally, we envisage the adaptation of our framework to a variety of heterogeneous platforms in future work. Currently, the protection provided by SecureComm is limited since it is lightweight, potentially inadequate for real-world deployments with complex security needs. Therefore, as part of our future work, we plan to explore the integration of a TEE to enhance the overall security of SecureComm. This integration will fortify protection and cater to the diverse security demands of intricate scenarios.

Declarations

- The authors have no conflicts of interest to declare that are relevant to the content of this article.

References

[1] Xilinx, "Zynq-7000 SoC," Xilinx Inc. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html, 2017, accessed: July 19, 2023.

[2] I. Corporation, "Cyclone® v fpga and soc fpga," Intel Inc. https://www.intel.com/content/www/us/en/products/details/fpga/cyclone/v.html, 2013, accessed: July 19, 2023.

[3] Microsoft, "Deploy ml models to field-programmable gate arrays (fpgas) with azure machine learning," Microsoft Inc. https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-fpga-web-service, 2019, accessed on July 19, 2023.

[4] A. W. Services, "Amazon ec2 f1 instances: Enable faster fpga accelerator development and deployment in the cloud," Amazon Inc. https://aws.amazon.com/ec2/instance-types/f1/, 2019, accessed: July 19, 2023.

[5] L. Deng, D. Li, Z. Cai, and L. Hong, "Smart iot information transmission and security optimization model based on chaotic neural computing," *Neural Computing and Applications*, vol. 32, pp. 16 491–16 504, 2020.

[6] M. K. Chowdary, T. N. Nguyen, and D. J. Hemanth, "Deep learning-based facial emotion recognition for human–computer interaction applications," *Neural Computing and Applications*, pp. 1–18, 2021.

[7] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (mpsoc) technology," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 27, no. 10, pp. 1701–1713, 2008.

[8] G. Lin, S. Wen, Q.-L. Han, J. Zhang, and Y. Xiang, "Software vulnerability detection using deep neural networks: a survey," *Proceedings of the IEEE*, vol. 108, no. 10, pp. 1825–1848, 2020.

[9] X. Feng, X. Zhu, Q.-L. Han, W. Zhou, S. Wen, and Y. Xiang, "Detecting vulnerability on iot device firmware: A survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 1, pp. 25–41, 2022.

[10] J. Zhang, L. Pan, Q.-L. Han, C. Chen, S. Wen, and Y. Xiang, "Deep learning based attack detection for cyber-physical system cybersecurity: A survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 3, pp. 377–391, 2021.

[11] J. Zheng, Y. Zhang, Y. Li, S. Wu, and X. Yu, "Towards evaluating the robustness of adversarial attacks against image scaling transformation," *Chinese Journal of Electronics*, vol. 32, no. 1, pp. 151–158, 2023.

[12] Q. Zhang, W. Ma, Y. Wang, Y. Zhang, Z. Shi, and Y. Li, "Backdoor attacks on image classification models in deep neural networks," *Chinese Journal of Electronics*, vol. 31, no. 2, pp. 199–212, 2022.

[13] E. Benhani, L. Bossuet, and A. Aubert, "The security of arm trustzone in a fpga-based soc," *IEEE Transactions on Computers*, vol. 68, no. 8, pp. 1238–1248, 2019.

[14] E. M. Benhani, C. Marchand, A. Aubert, and L. Bossuet, "On the security evaluation of the arm trustzone extension in a heterogeneous soc," in *2017 30th IEEE International System-on-Chip Conference (SOCC)*, 2017, pp. 108–113.

[15] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, 2015, pp. 57–64.

[16] K. Xia, Y. Luo, X. Xu, and S. Wei, "Sgx-fpga: Trusted execution environment for cpu-fpga heterogeneous architecture," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 301–306.

[17] M. Zhao, M. Gao, and C. Kozyrakis, "Shef: Shielded enclaves for cloud fpgas," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 1070–1085.

[18] A. Proulx, J.-Y. Chouinard, P. Fortier, and A. Miled, "A survey on fpga cybersecurity design strategies," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 16, no. 2, pp. 1–33, 2023.

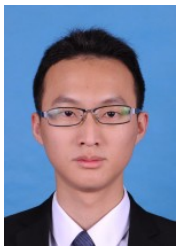
[19] W. Saad, A. Sanjab, Y. Wang, C. A. Kamhoua, and K. A. Kwiat, "Hardware trojan detection game: A prospect-theoretic approach," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 9, pp. 7697–7710, 2017.

[20] B. Li, M. Liu, and D. Lin, "Fpga implementations of grain v1, mickey 2.0, trivium, lizard and plantlet," *Microprocessors and Microsystems*, vol. 78, p. 103210, 2020.

[21] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: Threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[22] R. Elbaz, L. Torres, G. Sassatelli, P. Guillemin, C. Anguille, C. Buatois, and J. Rigaud, "Hardware engines for bus encryption: a survey of

existing techniques," in *Design, Automation and Test in Europe*, 2005, pp. 40–45 Vol. 3.
[23] T. Hiscock, O. Savry, and L. Goubin, "Lightweight instruction-level encryption for embedded processors using stream ciphers," *Microprocessors and Microsystems*, vol. 64, pp. 43–52, 2019.
[24] F. Hou, H. He, N. Xiao, F. Liu, and G. Zhong, "Efficient encryption-authentication of shared bus-memory in smp system," in *2010 10th IEEE International Conference on Computer and Information Technology*, 2010, pp. 871–876.
[25] X. Chen, R. P. Dick, and A. Choudhary, "Operating system controlled processor-memory bus encryption," in *Proceedings of the conference on Design, automation and test in Europe*, 2008, pp. 1154–1159.
[26] W. A. Farag, "Cantrack: Enhancing automotive can bus security using intuitive encryption algorithms," in *2017 7th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO)*, 2017, pp. 1–5.
[27] J. Lázaro, A. Astarloa, A. Zuloaga, U. Bidarte, and J. Jiménez, "I2csec: A secure serial chip-to-chip communication protocol," *Journal of Systems Architecture*, vol. 57, no. 2, pp. 206–213, 2011.
[28] Y. Li and L. Shi, "Design and implementation of encryption filter driver for usb storage devices," in *2011 Fourth International Symposium on Computational Intelligence and Design*, vol. 1, 2011, pp. 356–359.
[29] J. Lázaro, A. Astarloa, L. Muguira, U. Bidarte, and J. Jiménez, "Encryption axi transaction core for enhanced fpga security," *Electronics*, vol. 11, no. 20, p. 3361, 2022.
[30] E. M. Benhani, C. M. Lopez, and L. Bossuet, "Secure internal communication of a trustzone-enabled heterogeneous soc lightweight encryption," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, 2019, pp. 239–242.
[31] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on {GPUs}," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 681–696.
[32] I. Jang, A. Tang, T. Kim, S. Sethumadhavan, and J. Huh, "Heterogeneous isolated execution for commodity gpus," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 455–468.
[33] T. Hunt, Z. Jia, V. Miller, A. Szekely, Y. Hu, C. J. Rossbach, and E. Witchel, "Telekine: Secure computing with cloud {GPUs}," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 817–833.
[34] J. Zhu, R. Hou, X. Wang, W. Wang, J. Cao, L. Zhao, F. Yuan, P. Li, Z. Wang, B. Zhao *et al.*, "Enabling privacy-preserving, compute- and data-intensive computing using heterogeneous trusted execution environment," *arXiv preprint arXiv:1904.04782*, 2019.
[35] R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A.-R. Sadeghi, and E. Stapf, "{CURE}: A security architecture with {CUstomizable} and resilient enclaves," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1073–1090.
[36] X. Gao, E. Lu, L. Xian, and H. Chen, "Fpga implementation of the sms4 block cipher in the chinese wapi standard," in *2008 International Conference on Embedded Software and Systems Symposia*, 2008, pp. 104–106.
[37] Y. Jin, H. Shen, and R. You, "Implementation of sms4 block cipher on fpga," in *2006 First International Conference on Communications and Networking in China*, 2006, pp. 1–4.
[38] H. S. H. Fenghua;, "Optimization and implementation of the sm4 on fpga," *Journal of Xidian University*, vol. 48, no. 03, pp. 155–162, 2021. [Online]. Available: https
[39] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[40] H. D. Kabir, M. Abdar, A. Khosravi, S. M. J. Jalali, A. F. Atiya, S. Nahavandi, and D. Srinivasan, "Spinalnet: Deep neural network with gradual input," *IEEE Transactions on Artificial Intelligence*, 2022.
[41] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan, "Fpga implementation of k-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data," in *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2011, pp. 248–255.
[42] Y. Yamaguchi, H. K. Tsoi, and W. Luk, "Fpga-based smith-waterman algorithm: Analysis and novel design," in *Reconfigurable Computing: Architectures, Tools and Applications*, A. Koch, R. Krishnamurthy, J. McAllister, R. Woods, and T. El-Ghazawi, Eds., 2011, pp. 181–192.

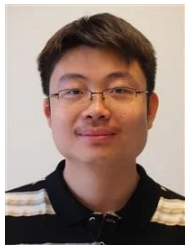


Tian Chen received his master degree in electronic and communication engineering from Institute of Acoustics, Chinese Academy of Sciences. He is currently a Ph.D. candidate in the School of Cyberspace Security, Beijing Institute of Technology. His primary research interests include parallel computing and systems security.

Yu-an Tan received the B.Eng. degree in Computer Software in 1991, Ph.D. in Computer Science in 2003. He has got teaching and research experience of more than 26 years, and has been a Professor in Beijing Institute of Technology since 2010. He is a senior member of the China Computer Federation. He contributes for peer reviewed 50+ journal papers and conference papers, including 5 papers of the top 1% of Essential Science Indicators. He has received over 20 research funds from National Natural Science Foundation of China, National Key Research and Development Program of China, etc. His research areas include Artificial Intelligence Security, Cybersecurity and Storage Sub-system.

Chunying Li received her Ph.D. in Service Computing Theory and Technology from South China Normal University in 2016 and her M.Eng. in Software Engineering from Beijing Institute of Technology in 2007. Now she is a Professor at the School of Computer Science, Guangdong Polytechnic Normal University, where she also serves as the Associate Dean. She is a senior member of the China Computer Federation (CCF). Her research interests include intelligent education, knowledge graphs, and recommendation algorithms.

Zheng Zhang received the master degree in instrument science and technology from Beijing University of Aeronautics and Astronautics. He is currently a Ph.D. candidate in the School of Computer Science, Beijing Institute of Technology. His primary research interests include parallel computing, systems security, including trustworthy execution and control-flow integrity.



Weizhi Meng is currently an Associate Professor in the Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Denmark. He obtained his Ph.D. degree in Computer Science from the City University of Hong Kong (CityU), Hong Kong. Prior to joining DTU, he worked as a research scientist in Institute for Infocomm Research, Singapore, and as a senior research associate in CityU. He won the Outstanding Academic Performance Award during his doctoral study, and is a recipient of the HKIE Outstanding Paper Award for Young Engineers/Researchers in both 2014 and 2017. His primary research interests are cyber security and intelligent technology in security including intrusion detection, smartphone security, biometric authentication, HCI security, cloud security, trust management, malware detection, blockchain, and IoT security. He is a senior member of IEEE.

Yuanzhang Li received the B.Eng. degree, M.Eng degree and Ph.D. degree in software and theory of computer in 2001, 2004, and 2015 from Beijing Institute of Technology. Now he has been an Associate Professor in Beijing Institute of Technology. His main research interests focus on mobile computing and information security.