2023 26th Euromicro Conference on Digital System Design (DSD) | 979-8-3503-4419-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/DSD60849.2023.00102

FPGA-based Encryption System for Cloud Security

Marios Papadopoulos, Paris Kitsos
ECSA Lab., Electrical and Computer Engineering Department
University of the Peloponnese, Greece

m.papadopoulos@go.uop.gr, kitsos@uop.gr

***Abstract*—The primary goal of this work is to introduce possible hardware solutions that can increase trust in cloud computing while maintaining data control in the hands of the data holder, in the framework of the collaborative environment provided by cloud computing. A software-based solution leads to possible anonymity or impersonation.**

***Keywords*— FPGA, Cloud Encryption, RSA and KAC techniques**

I. Introduction

Cloud Computing is an emerging technology in industry that enables access to a shared pool of computing resources for cloud users. Despite Cloud computing offers many benefits to users, there are some drawbacks that place restrictions on the usage of cloud computing. One of the most important is security. Data sharing is an important functionality in cloud storage [1]. One of the major problems is how the user can share encrypted data efficiently. In an inefficient way, a user can download the encrypted data from the storage, decrypt them and then send them to others users. But, by this way the user loses the value of cloud storage. Although, the users should be able to share partial data in cloud storage, this task is not easy in a security point of view. In cloud computing, this property is called, multitenancy which means that many customers of a cloud vendor use the same computer resources [2]. Although they share resources, cloud clients do not know each other and their data is kept completely separate.

In addition, FPGAs (Field Programmable Gate Arrays) is an excellent piece of the Cloud Computing toolbox. Since FPGAs that can offers many advantages such as reconfigurability, high throughput, predictable latency and low power consumption can be utilized as a general purpose computing resources on the cloud. They also can change on-the-fly their circuits offering functionality enhancement using Dynamic Partial Reconfiguration (DPR) capabilities [1]. This property is very usefull when a given FPGA is shared simultaneously by multiple tenants. It is obvious that any security approach proposed should take into account and support multifunctionality. So, an adoption of existing techniques, e.g. IP protection for a single tenancy do not address the issues' arises in cloud for practical deployment.

In this paper we investigate the efficiency in the hardware implementation point of view of a Key-Aggregation-Cryptosystem (KAC) proposed in [2]. The proposed methodology is a scalable framework for secure multi-tenant FPGA on the cloud.

The rest of the paper is organized as follows. In Section II, we discuss the Key-Aggregate Cryptosystem (KAC) [3] with the relative architecture. In Section III, we describe the system basic implementation. In section IV the implementation results are given while section V concludes the paper.

II. Security in a Multitenancy Environment

The heart of the proposed methodology is the Key-Aggregation-Cryptosystem (KAC) [2-3] that can offer both bitstream encryption / decryption for different tenants and efficient key-management. The scheme of the KAC is shown in Fig. 1.

According to [2] each plaintext message is associated with a unique identity *Id* that is encrypted with a common master public-key (*mpk)*, generated by the system administrator. The administrator, which is part of the cloud security system frontend and interacts with both the cloud security system and the user environment, also generates a master secret-key (*msk*), that is used to generate decryption keys for the plaintexts. The basic advantage of KAC is its ability to generate constant-size aggregate decryption keys corresponding to identities *Id1*, *Id2*, · · · , *Idn*. Then, it is possible to generate a constant-size aggregate decryption key *ski* for the *i*-th ciphertext. For example, in Fig. 1 the individual secret-keys *sk1* and *skn* for the identities *Id1* and *Idn* are compressed into a single aggregate-key *sk1,n*, which has the same size as either of *sk1* and *skn*, that can decrypt the ciphertexts *C1* and *Cn*, but not *C2*. KAC operation is based on Elliptic Curves.

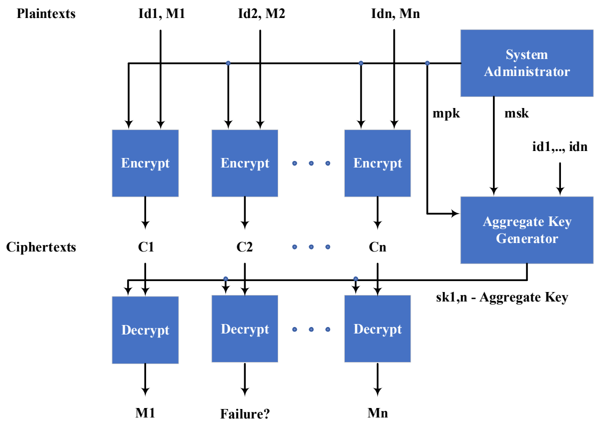

Fig. 1. Key-Aggremment Cryptosystem

In order to setup the security mechanism in a multitenant environment some steps are executed [2]. Firstly, the FPGA vendor sets up a KAC system using a master public key and a master secret key that are generated byself. Each FPGA divided into a *n* partitions, with each partition has a unique identity *Id*. Each partition corresponds to an independent virtual FPGA from the tenant point of view. In addition, it contains a Virtual Machine that can be used for the configuration of the tenant and a KAC decryption engine, that is pre-programmed to use a single aggregate decryption key *sk* that corresponds to the *Id* as a host. Then, each tenant encrypts its bitstream using an AES128 key. This AES128 key had been additionally encrypted using the master public key of the KAC. The second encryption is performed under the identity *Id* of the partition assigned to the tenant. The bitstream encryption occurs inside the FPGA in two steps. Each FPGA is provided with a single KAC decryption core, while each individual partition is provided with its own AES128 decryption core. The KAC decryption engine is first used to recover the AES128 key chosen by the tenant. The recovered key is subsequently used to decrypt any number of encrypted bitstreams and program the FPGA partition with the same.

Encryption and decryption keys are created for unique data that the user provides. Only a particular set of decryption keys are shared so that the data can be decrypted. A public–key encryption system which is called a Key-Aggregate cryptosystem (KAC) is presented. This system produces constant size ciphertexts. Any arrangement of secret keys can be aggregated and make them into a single key, which has the same power of the keys that are being used. This total key can then be sent to the others for decoding of a ciphertext set and remaining encoded documents outside the set stays private. The encrypted data is stored in the cloud, but the problem comes with how to share the keys securely so that all the unnecessary information should not be exposed to the user. If the sender encrypts all the information using a single key and sends it to the receiver, he is exposing all the important information to the receiver, which is a breach of privacy. Also, if he sends each key for the encrypted data, then he needs to send decryption keys for each encrypted data. This is acceptable when the number of keys is small, but for thousands of keys, separate storage needs to be maintained, and these must be sent through a secure channel, where there might be a chance for data leakage. So, to overcome this problem, a key aggregate cryptosystem concept is developed where a single aggregate key of constant size is formed from the set of keys and sent to the receiver [8]. The key aggregate cryptosystem is one of the most efficient, scalable and secured techniques that can be implemented in cloud storage. A key aggregate cryptosystem is a public key cryptosystem in which all of the sets of secret keys are aggregated into a single key, and the aggregated key has the power of all the secret keys [8]. proposed a leakage resilient key aggregate cryptosystem with auxiliary input based on the KAC scheme [9] .

We propose a complete user-cloud encryption system based on FPGA technologies. User security is based on encryption with RSA and KAC techniques that are exclusively in user hands.

III. BASIC IMPLEMENTATION

The initial design of the proposed system includes the encryption / decryption core which is based on the ZYNQ / ZEDBOARD platform [5], and the user interface environment.

TABLE 1: Comparison of KAC with Related Schemes [12]

System	*Size of Cipher Text*	*Size of Decryption Key*	*Type of Encryption*
Key assignment schemas	constant	non-constant	symmetric or public key
Symmetric key encryption with compact key	constant	constant	symmetric key
Identity based encryption with compact key	non-constant	constant	public key
Attribute based encryption	constant	non-constant	public key
KAC	constant	constant	public key

A. Aes Core

The operation of the system is based on the capabilities of the ZEDBOARD platform using the AES IP CORE subsystem and the ETHERNET interface it supports. This core contains an 128-bit data input, an 32 bit control signal input, an 128-bit data output, and a 32-bit control signal output.

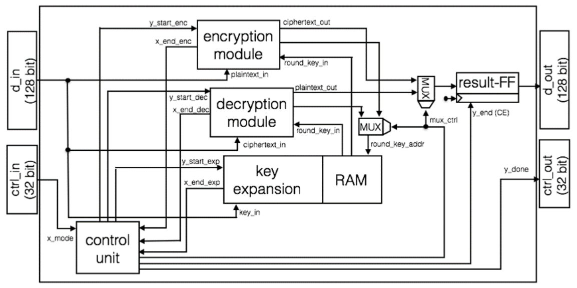

Fig. 2. Aes IP Core

B. Ethernet Interface

The encryption/decryption system interface is supported via Ethernet interface by the platform, the image diagram shows the ETHERNET system chipset which based on the integrated circuit of the Marvell 88E1518 PHY family and supports speeds up to 1000 Mbps. Ethernet chipset synchronize in 125 Mhz internal clock. There are two Ethernet MACs in the PS (Processing System) portion of the ZYNQ device. The first of these, ETH0 is connected, via the ZYNQ MIO interface, directly to Marvell Ethernet PHY on the ZEDBOARD using an RGMII interface. This Ethernet interface is 'free' on the ZEDBOARD and is what most applications and operating systems use to connect. The second Ethernet MAC in the ZYNQ PS, ETH1, cannot be connected directly via the MIO pins due to their multiplexed nature and the other peripherals on the ZEDBOARD connected to them. To use the second Ethernet MAC you would need to route the interface signals through the PL (Programmable Logic) portion of the ZYNQ device. Then you would need to connect these signals (as a GMII interface) to external ZYNQ pins connected to a PHY that you would have to provide on one of the external ZEDBOARD connectors such as the FMC connector. It is also possible to use a PHY with an RGMII interface by using a GMII to RGMII 'shim' IP within the ZYNQ PL. Unless you need a second Ethernet port, or have a specific interface requirement that the Marvell PHY on the ZEDBOARD does not support, you will not need to use the second Ethernet MAC

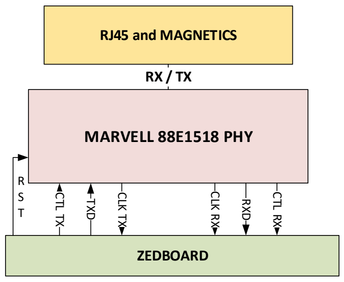

Fig. 3. Ethernet Chipset

C. *The System on Chip*

Figure 4 describes the overall architecture of our ZEDBOARD AES system. In the core part of the system implementation, both the Ethernet interface and the SD card are routed directly to the ZYNQ System on Chip (SoC) via MIO (Multipurpose IO) but do not go through PL (Programmable Logic). This enabled Xilinx libraries to be used directly at the PS (Processor System) level, allowing for easier integration of complex interfaces. The construction of CUSTOM IP [5] concerning the IP AES block contains several hardware blocks connected together and a State Machine. This IP block contains two interfaces: one interface for initializing the IP block by setting the key and mode (encryption or decryption), another for streaming the status via the AXI DMA protocol.

During the development process, each of the separate steps of the AES algorithm was designed as separate AXI4 IP blocks: using registers for its interface. The basis of the AES system consists of a well-known and reliable software implementation of the AES algorithm, Tiny AES in C [4], each hardware step was checked for accuracy on the ZEDBOARD at runtime. After all the components were completed, a state machine was designed to handle the flow of data that was running the software in the early stages of development.
Using this methodology helped to separate the implementation and completion stages, allowing full reliability in the final design of the block. The CBC mode was implemented exclusively by the software. Although it is more efficient and faster to process the additional XOR step in hardware, the additional complexity required in the given time frame of this project meant that there was no time for this optimization. Any client that supports the TCP/IP protocol can connect to the ZEDBOARD AES system. Once the connection is established, the host can send a file at any time. If the ZEDBOARD AES system is not ready, the connection will be rejected and no file transfer will occur. Currently, the system is limited to one connection at a time. ZEDBOARD AES is a completely self-contained system. A user can then navigate the device using the D-pad and OLED display. The system's ability to be network-enabled over Ethernet allows for flexibility in more applications, while leveraging the same AES encryption/decryption infrastructure as SD card functionality.

The performance gain from using hardware acceleration over a pure software application has been measured to be about 9.8 times faster when the ARM CPU was clocked at 200 MHz [5]. While this gain in performance is significant, it could be improved even further. Currently the implementation uses DMA to transfer 4 words each cycle, this requires many calls to the DMA transfer function and is the major bottleneck of our system. Ideally we could implement a streaming interface where the DMA would be instructed to continuously transfer the entire file instead of 16 byte chunks as in the current implementation. However, in its current state, it takes about 4 seconds to encrypt or decrypt 10 MB, which is a relatively satisfactory performance. The maximum file size that can be processed in the system is 100 MB. Regardless, the entire application can reach a maximum of 512 MB, which is the maximum DRR memory available on the ZEDBOARD. The AES IP Block is constructed from a large subset of self-contained elements, each of which represents a step in the AES encryption/decryption algorithm.

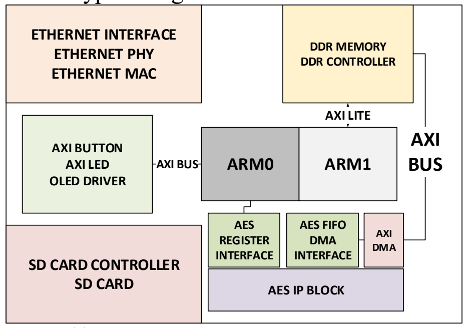

Fig 4. System Architecture

In the file structure and throughout the VHDL code, these functions remain self-contained and are held together by a higher-level state machine. In this design, both encryption and decryption are performed in parallel and the final data output depends on the mode selected. The hardware performs one round of encryption per cycle, receives 32 bits of stream data per cycle, and outputs 32 bits of stream data per cycle. Therefore, it takes 4 clock cycles to read and send each 16-byte state. There are many implementations of AES on FPGA on the web. In this implementation, the AES algorithm, Tiny AES in C [4] was used as our main reference. Abdeladim Sadiki and Jevin Sweval's [6] implementations were also used as secondary references in this component design [6] [7].

IV. Implementation Results

The implementation results are shown in Table 2. The analysis describes the FPGA system resources that were committed.

TABLE 2: HARDWARE RESOURCES

Site Type	*Used*	*Available*	*Util%*
Slice LUTs	13171	53200	24.76
LUT as Memory	626	17400	3.60
Slice Registers	7917	106400	7.44
Slice	4354	13300	32.74
LUT as Logic	12545	53200	23.58
Block RAM tile	2.5	140	1.79

A. Communication with user interface environment

Streaming interface follows the AXI DMA protocol, with 32-bit wide data, and signal registers to start and end transfers. The AES state machine controls the "ready" signals that the DMA protocol uses to send and receive data. An entire 16 bytes state must be encrypted or decrypted before the next state of data can be processed. In the current implementation, the IP block must be reset after completion of each state in order to return it to the initial step in the state machine. The encryption/decryption system communication is implemented through an ECHO SERVER that supports TCP operations and relies on the lwip libraries available. During the stage of sending information to be encrypted/decrypted, the user interface system sends the data to the system in the form of a packet as follows:

TABLE 3 :TRANSMIT PACKET FORM

File Size (4bytes)	***Mode (4bytes)***	***Raw Data (max 100 MB)***
The first 4 bytes include the size of the data (Raw Data) in bytes including the 4 bytes of the File Size header.	4 bytes that make up the mode selector (Mode Select Switch), if their value is zero (0) the operation of the system concerns the encryption of the input data, while in any other situation it concerns the decryption of the input data.	Data with a maximum length of 100 MB.

After the encryption process, the system returns to the user interface the results in the following format.

TABLE 4: RECEIVE PACKET FORM

File Size (4bytes)	***Raw Data (max 100 MB)***
The first 4 bytes contain the size of the data (Raw Data) in bytes received.	Received Data with a maximum length of 100 MB.

B. Use Case

In case of using encryption of a text file containing the phrase "HELLO WORLD", the packet to be transmitted will contain the header and data information as follows:
The size of the file in 11 bytes, the FILE SIZE header is 4 bytes, therefore the content of the File Size header is 15 bytes (x0F). The operation in this particular case concerns information encryption, therefore the Mode header is zero (x00). The information of the phrase "HELLO WORLD" that will be sent to the system has the form: 48454c4c4f20574f524c44. The packet to be sent via the TCP protocol has the following format (See Table 5). Immediately after sending the packet, the system returns the encrypted response via the TCP protocol.

C. User Web Interface

To implement the user interface environment, using UBUNTU 20 operating system, the WEBMIN platform, which includes Apache 2, MariaDB, Firewall, all this configuration was installed on a VPS (ESXI). The user interface allows uploading a file with a maximum size of 2MB, sending it to the AES IP CORE for encryption, and then storing the encrypted file in the user's Home Directory in the Cloud (which is simulated in the web interface). Also, the interface allows loading an encrypted file with a maximum size of 2 MB, sending it to the AES IP CORE for decryption, and then saving the decrypted file to the user's local disk. Cloud System tested in the following URL: http://parmenion.marios.gr/

TABLE 5: TRANSMIT / RECEIVE PACKET FORM EXAMPLE

File Size (4bytes)	***Mode (4bytes)***	***Raw Data (max 100 MB)***
0F 00 00 00	00 00 00 00 TRANSMIT	. 48 45 4C 4C 4F 20 57 4F 52 4C 44
10 00 00 00	11 11 11 11 RECEIVE	0C C5 20 95 8C 1C 23 21 07 6A E9 9D 79 38 52 A0

V. CONCLUSION

In this work, the implementation of a frontend encryption/decryption system for files in cloud environments is described in detail. The core of the system is based on ZEBOARD FPGA platform, on which SoC was implemented and support security, encryption and decryption functions via SD card or ETHERNET. The basis of the AES system consists of a well-known and reliable software implementation of the AES algorithm, Tiny AES in C. The web user interface allows uploading and download a file from or to the user's Home Directory in the Cloud. It should be noted that an important approach of the work is also the definition of the encryption/decryption model (KAC) based on the needs and characteristics of the computing cloud, which will also constitute the road map for the further approach

REFERENCES

[1] K. Vipin, S. A. Fahmy, "FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications", *ACM Computing Surveys*, Vol. 51, No. 4, 2019.

[2] A. Bag, S. Patranabis, D. Basu Roy and D. Mukhopadhyay, "Cryptographically Secure Multi-Tenant Provisioning of FPGAs", *Security, Privacy, and Applied Cryptography Engineering. SPACE 2020*, Lecture Notes in Computer Science, Vol. 12586, 2020.

[3] S. Patranabis, Y. Shrivastava and D. Mukhopadhyay, "Provably Secure Key-Aggregate Cryptosystems with Broadcast Aggregate Keys for Online Data Sharing on the Cloud", *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 891-904, 1 May 2017.

[4] "Tiny AES in C," [Online]. Available: https://github.com/kokke/tiny-AES-c.

[5] T. jason, D. Dixon "AES File Encryption and Decryption" [Online]. Available: https://github.com/jasonrtsang/ZEDBOARD_aes

[6] A. Sadiki, "A VHDL implementation of the AES algorithm", [Online]. Available:https://github.com/AbdeladimSadiki/AES-VHDL.

[7] J. Sweval, "A VHDL implementation of 128 bit AES encryption with a PCIe interface", [Online]. Available: https://github.com/jevinskie/aes-over-pcie

[8] C. K. Chu, S. S. Chow, W. G. Tzeng, J. Zhou, R. H. Deng, "Key-aggregate cryptosystem for scalable data sharing in cloud storage", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, No. 2, 2014.

[9] Cui, B., Liu, Z., Wang. "Key-aggregate searchable encryption (KASE) for group data sharing via cloud storage", *IEEE Transactions on Computers*, Vol. 65, No. 8, 2016.

[10] Q. Gan, X. Wang, D. Wu, "Revocable key-aggregate cryptosystem for data sharing in cloud. Security and Communication Networks", *Security and Communication Networks*, Vol. 2017, 2007.

[11] F. Guo, Y. Mu, Z. Chen, "Identity-based encryption: How to decrypt multiple ciphertexts using a single decryption key", In Proceedings of *Pairing-Based Cryptography 2007*, Lecture Notes in Computer Science, Vol 4575, 2007.

[12] K. Kate and S. D. Potdukhe, "Data sharing in cloud storage with key-aggregate cryptosystem", *International Journal of Engineering Research and General Science*, Vol. 2, No. 6, 2014.