Hardware-Accelerated Encryption with Strong Authentication

Zdenek Martinasek[1], Jan Hajny[1], Lukas Malina[1] and Denis Matousek[2]

martinasek@feec.vutbr.cz, hajny@feec.vutbr.cz,
malina@feec.vutbr.cz, matousek@netcope.com

[1]Faculty of Electrical Engineering and Communication
Brno University of Technology
[2]Netcope Technologies, a.s.

Abstract

With the growing amount of data transferred over communication networks, the high-speed encryption systems are becoming a hot topic. The paper is focused on the design and implementation of a hardware-accelerated encryption system based on 100 Gbps FPGA (Field Programmable Gate Array) network cards. First, an AES (Advanced Encryption Standard)-based encryption system is designed and implemented on the FPGA platform using the VHDL (VHSIC Hardware Description Language). The AES core is implemented using the GCM (Galois/Counter Mode) so that both confidentiality and integrity of data are provided. The AES core is then integrated with a strong authentication subsystem based on programmable smart-cards used for storing sensitive cryptographic material. The authentication subsystem implements the IKE protocol using shared secrets. In contrast to existing implementations, the keys used for authentication never leave a tamper-proof device in our system, all cryptographic operations are implemented on the smart-cards. The use of smart-cards significantly increases the security of the system as the keys do not have to be stored on a shared vulnerable file system any more. The resulting system is compliant with IPsec specification and will be interoperable with existing implementations. The paper contains the description of the system, results of the implementation benchmarks on the NFB-40G2 (Xilinx, Virtex-7) cards and proposals for next development.

Keywords: AES, Authentication, Encryption, IPsec, FPGA, GCM, Security, Smart card.

1 Introduction

The amount of data transferred over communication networks and security attacks realized are constantly growing. Therefore, the high-speed encryption systems become more important. These systems can be applied in different areas of data communication in order to satisfy demands of security and privacy of information transferred through communication channels. Typical examples of utilization are secure remote connectivity for government, commercial and financial institutes, cloud providers, ISP (Internet Service Provider), secure internal datacenter connectivity and so on. This paper presents the first results of the currently running project that seeks to propose and implement a hardware-accelerated encryption system based on 100 Gb/s FPGA (Field Programmable Gate Array) network cards. FPGAs allow us to develop hardware-accelerated applications. The biggest advantage of using FPGAs is that the cards can be programmed after their manufacturing, which leads to their versatility. The main contribution of our system is the encryption speed 100 Gb/s. To our best knowledge, this speed has not been achieved in real device nowadays.

A prerequisite for achieving the high speed encryption is the utilization of a symmetric cryptographic algorithm, usually a block cipher. In our system, we employ AES (Advanced Encryption Standard) that is widely used symmetric cryptographic algorithm in many cryptographic applications [1]. The hardware-based implementation of the AES algorithm has been studied during the last 10 years due its advantages in comparison with its software-based implementation. In the following text, we summarize the most important work dealing with AES implementation on FPGA. One of the first articles, which describe the hardware implementation of the AES-128 algorithm (Xilinx Virtex-4), is the paper [2] where the authors reach encryption speed 30 Mb/s. Further, the articles [3, 4] focus on the efficient design of the hardware implementation of various cryptographic algorithms and prove that AES is a good choice. The following research focuses on efficient AES implementation from point of the application and the parameters such as used FPGA area, latency, throughput and power consumption [5, 6]. Hoang [7] presents an efficient implementation for high-speed applications and reaches the encryption speed of 1054Mb/s. Furthermore, authors in [8] explore pipelining techniques in order to increase throughput of an AES implementation on Xilinx Virtex-5 and their implementation accomplishes the maximum throughput of 73.737 Gb/s. The recent works [9, 10] propose the hardware optimized implementations on Xilinx Virtex-5 and Spartan-6 devices that achieve the throughput of 86 Gb/s and 113 Gb/s. Nevertheless, these achieved throughputs are only theoretical because using frequency 886.64 MHz is not currently applicable on real devices.

In this paper, we propose our system that achieves the throughput of 100 Gb/s by the concatenation of encryption/decryption components. Our components use the maximum frequency of 200 MHz that is acceptable in current production devices.

The mentioned related works do not take into account (and implement) the modes of the block cipher operation. Nonetheless, encryption devices deployed in the production environment require a proper cipher mode. In this paper, we propose and implement the FPGA encryption core with Galois Counter Mode (GCM) [11] in order to provide authenticated encryption. Furthermore, a crucial aspect for the secure symmetric encryption systems lies in key distribution and the authentication of communication entities. In our system, we propose an authentication subsystem based on programmable smart cards. In contrast to existing implementations (e.g. Strongswan) that use for example a pre-shared secret on a standard file system, the keys used during our authentication process never leave a tamper-proof device. The authentication keys are not stored on a shared vulnerable file system any more. This approach significantly increases the security of the system. The main contribution of this paper is threefold:

- We propose the architecture of the high-speed encryption system.
- We propose and implement an authentication subsystem employing smart cards.
- We present our AES-GCM implementation and achieved results of the encryption core in FPGA.

The rest of the paper is organized as follows: Section 2 presents the basic architecture of our proposed system. Section 3 describes our authentication subsystem based smart cards and its implementation. Section 4 presents our encryption subsystem based on FPGA and its implementation. Section 5 concludes the presented system and results.

2 Basic Architecture of Proposed High-Speed Encryption System

In this section, we present our design and implementation of the high-speed encryption system that can be deployed in the production environment of a datacentre. The example application is the encryption and decryption gateway(s) for end-users. Our proposed system deploys the IPSec protocol (Internet Protocol Security). IPSec is a protocol suite for secure Internet Protocol (IP) communications. IPSec provides the authentication and encryption of each IP packet in a communication session by utilizing the ESP (Encapsulating Security Payload) [12] and AH (Authentication Headers) [13] modes. However, the IPsec algorithms are computationally expensive and consume many CPU (Central Processing Unit) cycles. This usually limits data throughput. Hence, we implement cryptographic algorithms on the FPGA module in order to increase data throughput.

In order to understand the basic functionality of the proposed system, we first define the fundamental concepts of the IPsec protocol. ESP and AH take advantage of different symmetric cryptographic primitives and keys. Generally, the end station can have various cipher algorithms and keys established for different connections. This is defined as a Security Association (SA) in IPSec terminology and each station has this information stored in a special database. The initial entry in the database is inserted by the IKE (Internet Key Exchange) protocol [14]. At the beginning of the communication between two entities, the IKE protocol chooses the most secure supported cryptographic algorithm, authenticates the entities and establishes the set of secret keys for authentication and encryption services. Authentication can be realized by certificates with digital signatures or by HMAC (Keyed-hash Message Authentication Code) using a pre-shared key (PSK). The subsequent establishment of symmetric keys for ESP and HMAC is realized by the DH (Diffie-Hellman) protocol. The result is the IKE SA record in databases of stations that include supported cryptographic algorithms and established keys (also referred as master keys) from which all the necessary keys for security services are derived.

The basic block diagram of the proposed system is depicted in Figure 1. At the beginning of the communication, an initial record in SA database is created by using our application on smart card that is the implementation of the IKE protocol (an **authentication subsystem**). The SA record contains a master encryption key denoted as KME (Master Key Encryption), a master key for authentication (Key Master Authentication) and supported cryptographic primitives (eg. AES and SHA-2). In the next step, necessary information for ESP and AH are forwarded trough a defined API (Application Programming Interface) to the FPGA system block where the cryptographic algorithms are implemented. The encryption process is realized by the AES-GCM algorithm (a **cryptographic subsystem**). The FPGA block receives user's data through a virtual interface (VI) that provides formatting according to the IPSec specification. The receiving entity operates quite similarly. The stream of IPSec data is encrypted and forwarded trough VR to the application. Based on the basic architecture proposal, we analyzed the available implementations of IPSec and IKE protocols that can be emploed in the system. We selected StrongSwan as the appropriate implementation because it fulfills the following conditions: OpenSource, IKEv2 support, enable to use only IKE protocol, CentOS 6 support and PKCS #11 support (a smart card).

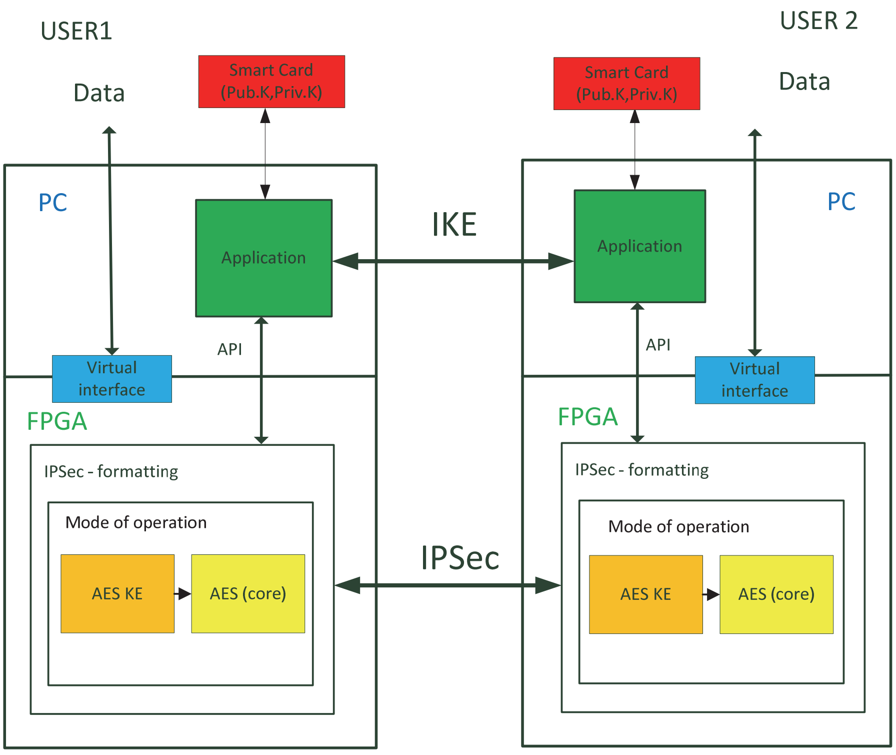


Figure 1: Block diagram of system proposed.

2.1 Authentication Subsystem Based on Programmable Smart-cards

In this section, we describe our proposed authentication subsystem and its experimental implementation. The main feature of the authentication subsystem is to verify the identities of the communication parties by using a pre-shared secret key which is stored on programmable smart cards. Smart cards work as Secure Access Modules (SAM) and provide the calculation of the encryption keys for the IPSec protocol. Smart cards host all critical operations and secret parameters in order to increase the security of the system. The connection between the smartcard and the card handler application is provided by APDU (Application Protocol Data Unit) messages via a USB (Universal Serial Bus) card reader. Our experimental implementation enables to establish a secure session key for a fast encryption FPGA module by using the IKEv2 Pre-shared Secret Key (PSK) authentication method. The authentication method between Initiator and Responder and interface connections are depicted in Figure 2.

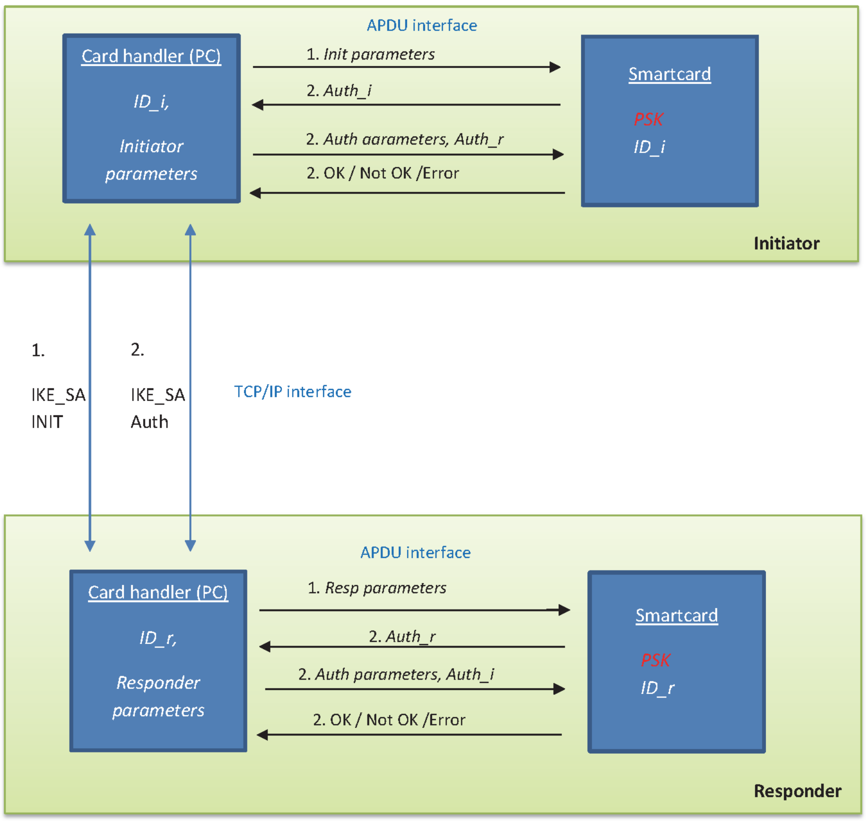
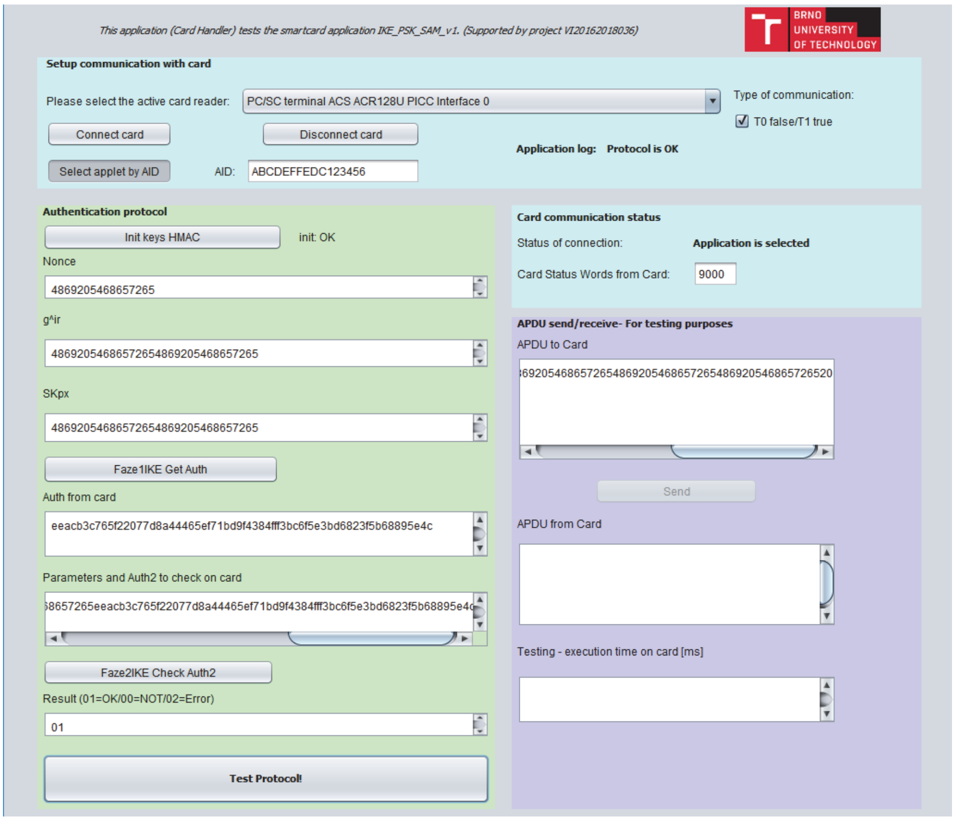


Figure 2: Proposed IKE_PSK authentication subsystem with smartcards.

Current programmable smart cards (JAVA Cards, MultOS Cards, .NET Cards, Basic Cards) differ in price, cryptographic APIs, a storage size, a card operation system security and a development support. We choose JAVA Card as an appropriate platform for our implementation. The experimental implementation can be divided on two parts:

- JAVA Card Handler PC application with graphical user interface (GUI),
- JAVA card applet running on JAVA Card OS JCAPI 2.2.2 or higher.

We implemented an experimental JAVA application Card Handler (PC) in order to establish APDU communication with a smart card via a USB interface-based reader (i.e. USB card reader ACR128). The graphical user interface of the application is presented in Figure 3. This application can test and manage the applet on a smart card.


This application (Card Handler) tests the smartcard application IKE_PSK_SAM_v1. (Supported by project VI20162018036)

Setup communication with card

Please select the active card reader: PC/SC terminal ACS ACR128U PICC Interface 0

Type of communication: [✓ T0 false/T1 true]

AID: ABCDEFFEDC123456

Application log: **Protocol is OK**

Authentication protocol

init: OK

Nonce: 4869205468657265

g^ir: 48692054686572654869205468657265

SKpx: 48692054686572654869205468657265

Auth from card: eeacb3c765f22077d8a44465ef71bd9f4384fff3bc6f5e3bd6823f5b68895e4c

Parameters and Auth2 to check on card: 68657265eeacb3c765f22077d8a44465ef71bd9f4384fff3bc6f5e3bd6823f5b68895e4c

Result (01=OK/00=NOT/02=Error): 01

Card communication status

Status of connection: **Application is selected**

Card Status Words from Card: 9000

APDU send/receive- For testing purposes

APDU to Card: 6920546865726548692054686572654869205468657265 20

APDU from Card

Testing - execution time on card [ms]

Figure 3: GUI of experimental JAVA Application - Card Handler (PC).

2.2 Encryption Subsystem Based on FPGA implementation

In this section, we present the components of our encryption subsystem. Our encryption subsystem uses Advanced Encryption Standard (AES) that is based on the Rijndael algorithm which replaces older Data Encryption Standard (DES). AES is a symmetric block cipher that works iteratively (in rounds) and supports key sizes of 128, 192, 256 bits and the block size of 128 bits. The use of larger key sizes increases the cryptographic strength but requires more rounds. In our system, we focus only on the AES implementation with the key size of 128 bits because it is sufficient for most applications that are commonly used. In the following text, we do not provide detailed information about AES because it is a well-known standard described in [1]. We focus solely on the integration of our implementation to a production system (i.e. Netcope FPGA card).

Our FPGA implementation can be divided in to the following components:

- an encryption component: AES cipher that consists all functions of AES (AddRoundKey, SubBytes, ShiftRows and MixColumns),
- a key expansion component that provides the Key Expansion function,
- a GCM component that provides the mode of operation.

The AES core has been already described in more details in the article [15]. In the following text, we provide the description of Key expansion, GCM and whole integration.

2.3 Key Expansion Component

In this subsection, we describe the key expansion component that creates round keys from the main 128-bit secret AES key. The 128-bit key is divided into sixteen 8-bit round keys that are written to the 4x4 matrix. The key expansion process is depicted in Figure 4. Each 8-bit field in the matrix is denoted as K *X*,*Y*. The columns are represented as W (word) with 4-byte size. The last word W 3 is used as an input to the function *f* depicted in Figure 5. The function *f* consists of 3 main operations: the RotWord operation that rotates the column up by 1 byte, the SubWord operation that substitutes input bytes with values from the S-box table and the Rcon operation that performs the xor operation with input values and with round constant values defined in Table 1. The output of the *f* function is xored with W 0 and the result is written into the W 0 register. This new value is xored with the W 1 and written to the W 1 register. W 2 and W 3 are computed similarly as W 1.

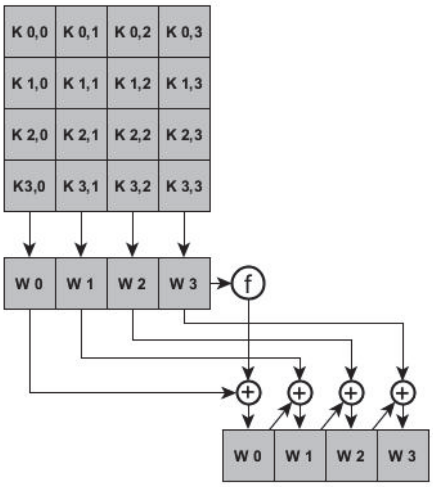


Figure 4: Key expansion process.

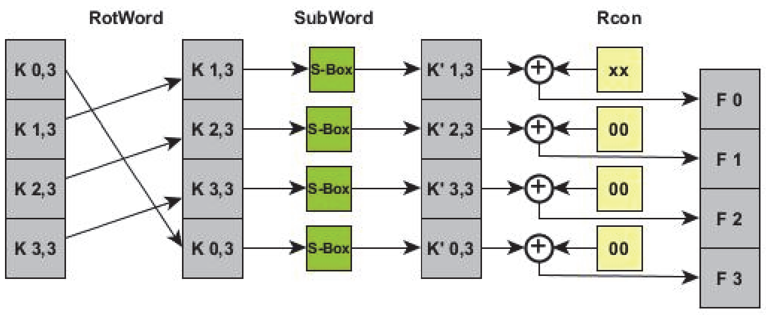


Figure 5: *f* function process.

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Table 1: Round constants Rcon.

In the following text, we show Example code 1 that represents our key expansion implementation in the VHDL language.

Example code 1: KeyExpansion.

```
--Rotword
rotword_col3 <= std_logic_vector(unsigned(col3) rol 8 );
--SubWord
Sub : process (rotword_col3)
begin
  sub0 <= SBOX(conv_integer(rotword_col3(31 downto 24)));
  sub1 <= SBOX(conv_integer(rotword_col3(23 downto 16)));
  sub2 <= SBOX(conv_integer(rotword_col3(15 downto 8)));
  sub3 <= SBOX(conv_integer(rotword_col3(7 downto 0)));
end process;
subbytes_col3 <= sub0 & sub1 & sub2 & sub3;
--RoundKey
rkey_col0 <= ((col0 xor subbytes_col3 ) xor rcon );
rkey_col1 <= col1 xor rkey_col0;
rkey_col2 <= col2 xor rkey_col1;
rkey_col3 <= col3 xor rkey_col2;
round_key <= rkey_col0 & rkey_col1 & rkey_col2 & rkey_col3 ;
```

2.4 GCM Cipher Mode Component

The basic scheme of the GCM cipher mode component is depicted in Figure 6. Firstly, the counter value is set to its initialization value which is added to an initialization vector value. This counter value *ctr* is then used as the input in the AES encryption component (E_K) that also uses the secret key. The output from the AES component is xored with the 128-bit data block of a plain text in order to produce the cipher text. The cipher text is also used as the input value in the mult_H multiplication function that creates the authentication tag denoted as *mult_H*. For the next data block, the counter value is incremented by 1 and the input for the mult_H function is created by the xor operation of the cipher text and the previous authentication tag (*prev mult_H*). The length of cipher text is exactly the length of the plain text. The authentication tag provides data authentication and integrity and is added as an additional tag behind the cipher text. The length of the authentication tag can be between 0 and 128 bits. The decryption operation is similar to the encryption operation. The decryption operation has five inputs: secret key, IV, cipher text, an additional authenticated data (unencrypted data), and the authentication tag. The output is the plain text and an indicator of data authenticity.

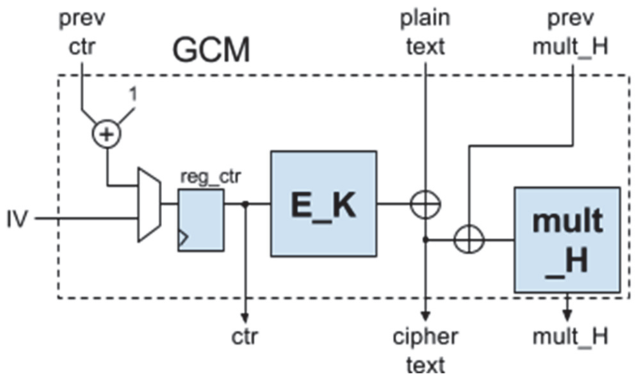


Figure 6: The scheme of GCM cipher mode component.

2.5 Integration of Individual Components on FPGA Cards

The encryption component computes all AES rounds. To achieve high throughput, it is necessary to compute individual rounds in parallel, namely through pipelined processing (illustrated in Figure 7). For example, Round 2 with the first data block is concurrently performed with computing the second block in Round 1. In general, block *N* is computed in Round 1 in parallel with computing the block *N*-1 in Round 2, block *N*-2 in Round 3, etc. By utilizing pipelined processing, our component computes one block per one clock cycle. The key expansion process can be also implemented in pipelined processing. Processing one block in the first cycle requires only the first part of the expanded key.

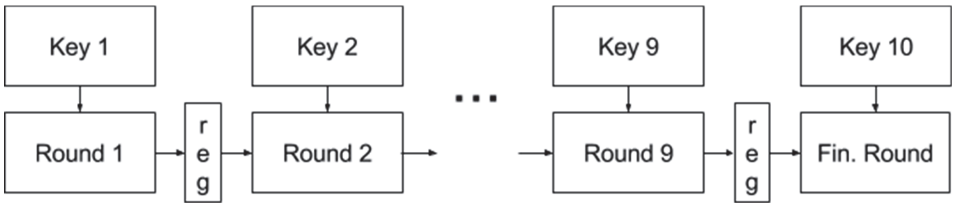


Figure 7: AES cipher component with pipelined processing.

The disadvantage of pipelined processing is the necessity to increase the amount of logic on a chip. More parallel rounds mean more logic levels. The minor drawback is the increased latency. The latency can be reduced by calculating several rounds within one clock. The register (Reg) is inserted after a few rounds. However, this approach reduces the maximum frequency at which the circuit is able to operate.

Table 2 shows resources required for the encryption component and the component of the key expansion on the chip from the Virtex7 family. Level of pipelined processing indicates how many rounds are performed in a single clock. Frequency denotes the maximum frequency at which the unit is able to run. LUT indicates the number of Look-Up Tables. FF denotes the number of Flip-Flop registers on a FPGA card.

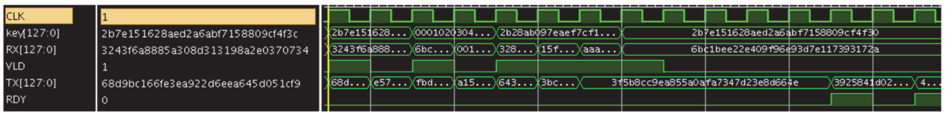
Component	**Number of rounds perfor-med in a one clock**	**Frequency (MHz)**		**LUT**		**FF**	
encryption	1	378	378	7245	10125	1538	2946
key_expansion	1	568		2880		1408	
encryption	2	217	217	7241	10281	898	1666
key_expansion	2	326		3040		768	
encryption	4	114	114	7233	9912	514	898
key_expansion	4	199		2679		384	

Table 2: Chip resources and parameters for encryption and key expansion components.

The throughput *t* of encryption is calculated as follows (*f* represents the frequency and N_b is the block size in bits):

$$t = f * N_b \text{ [b/s]} \quad (1).$$

Our implementation on FPGA module has been tested for different values of input data and keys. The correctness of the encrypted data is checked against the existing implementations of AES. We have also tested the accuracy of coping with the intermittent validity of the input data. These tests check if the unit gives the valid output of encrypted data corresponding to the valid input data. Figure 8 shows the part of waveform from our simulations.



```
CLK         1
key[127:0]  2b7e151628aed2a6abf7158809cf4f3c   2b7e151628...  0001020304...  2b28ab097eaef7cf1...  2b7e151628aed2a6abf7158809cf4f30
RX[127:0]   3243f6a8885a308d313198a2e0370734   3243f6a888...  6bc...  001...  328...  15f...  aaa...  6bc1bee22e409f96e93d7e117393172a
VLD         1
TX[127:0]   68d9bc166fe3ea922d6eea645d051cf9   68d...  e57...  fbd...  a15...  643...  3bc...  3f5b8cc9ea855a0afa7347d23e8d664e  3925841d02...  4...
RDY         0
```

Figure 8: Simulation of encryption component.

Our encryption subsystem based on FPGA consists of an encryption component, a key expansion component and a GCM cipher mode component. The GCM mode enhances the encryption by data integrity and authentication. The subsystem is plugged into NDK (Netcope Development Kit)[1] which provides all the auxiliary features such as communications software, receiving and sending packets over Ethernet interfaces and so on. Our encryption subsystem is illustrated in Figure 9. The subsystem is running at the frequency of 200MHz. According to the formula 1.11, it is necessary to use parallel bus of the width of 512b to achieve the throughput of 100Gbps. Input packets are delivered through parallel bus from network modules or DMA engine of NDK. To achieve full throughput, data are distributed into four data paths, each processing 128b of the original 512b data word. Each of these data paths contains an encryption unit (AES) and these units are interconnected with block-mode units (GCM) in a circular topology. Based on the values from the Table 2, it is necessary to use encryption units that perform at most two AES rounds in a single clock cycle to meet timing constraints for the frequency of 200MHz. To ensure proper synchronization of incoming data and encryption units, each data path contains buffers with appropriate capacity in front of and after encryption units. Output data of encryption units are merged on a parallel bus.

[1] More information about NDK on http://www.netcope.com/en/products/fpga-development-kit.

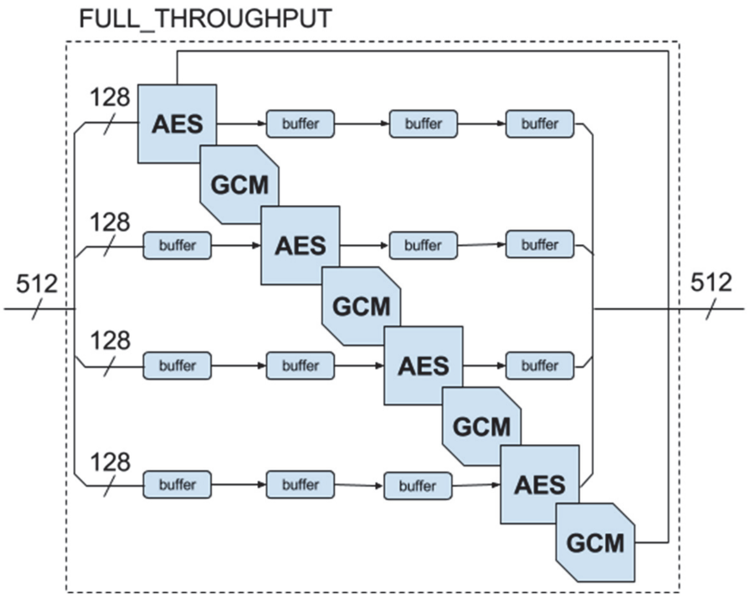


Figure 9: Encryption subsystem based on FPGA.

3 Conclusion

In this paper, we present our design and implementation of a hardware-accelerated encryption system based on 100 Gbps FPGA network cards. We design and implement the AES encryption subsystem with Galois/Counter mode for providing data confidentiality and integrity onto FPGA cards. Our implementation is based on the 4 duplicated encryption cores and employs pipelined processing in order to achieve full 100Gbps encryption speed. We also propose the authentication subsystem based on JAVA smart-cards. The smart cards are used for secure storing preshared secret keys that are employed in the mutual authentication and establishment of session secret keys. Our implemented authentication subsystem is integrated with the encryption subsystem. Our hardware-accelerated encryption system provides strong high-speed encryption, data authenticity and integrity, secure mutual authentication and key establishment, and it is compliant with the IPSec specification.

Our future work aims at testing the proposed encryption system in high-speed data communications and conducting research into different secure cipher modes providing authenticated encryption.

Acknowledgment

Research described in this paper was financed by the Ministry of Interior program "Program bezpečnostního výzkumu České republiky 2015-2020" under grant VI20162018036.

References

[1] NIST FIPS. 197: Advanced encryption standard (AES). Federal Information Processing Standards Publication 197.441 (2001): 0311

[2] Wiebe, J. H. (2007, December). AES-128 Implementation on a Virtex-4 FPGA. In Signal Processing and Information Technology, 2007 IEEE International Symposium on (pp. 68-73). IEEE.

[3] Saggese, G., Mazzeo, A., Mazzocca, N., Strollo, A., 2003. An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm. 2778, 292–302.

[4] A. Elbirt, W. Yip, B. Chetwynd, C. Paar An FPGA-based performance evaluation of the AES bolck cipher candidate algorithm finalists IEEE Trans. Very Large Scale Integr. VLSI Syst., 9 (2001), pp. 545–557

[5] Rouvroy, G., Standaert, F.X., Quisquater, J.-J., Legat, J., 2004. Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications. In: Proceedings of the International Conference on Information Technology: Coding and Computing. ITCC 2004, vol. 2, pp. 583–587.

[6] J. Van Dyken, J.G. Delgado-Frias FPGA schemes for minimizing the power-throughput trade-off in executing the advanced encryption standard algorithm J. Syst. Archit., 56 (2010), pp. 116–123

[7] Hoang, T., Nguyen, V.L. An efficient FPGA implementation of the advanced encryption standard algorithm. In: 2012 IEEE RIVF International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), pp. 1–4.

[8] M.I. Soliman, G.Y. Abozaid FPGA implementation and performance evaluation of a high throughput crypto coprocessor, J. Parallel Distrib. Comput., 71 (2011), pp. 1075–1084

[9] R.R. Farashahi, B. Rashidi, S.M. Sayedi, FPGA based fast and high-throughput 2-slow retiming 128-bit AES encryption algorithm

[10] Farooq, U., & Aslam, M. F. (2016). Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA. Journal of King Saud University-Computer and Information Sciences.

[11] McGrew, D., Viega, J. The Galois/counter mode of operation (GCM). Submission to NIST Modes of Operation Process 20 (2004).

[12] Kent, S. (2005). RFC 4303: IP Encapsulating Security Payload (ESP),

[13] Kent, S. RFC :4302: IP authentication header. 2005.

[14] Sheffer, Y., & Fluhrer, S. (2013). Additional Diffie-Hellman Tests for the Internet Key Exchange Protocol Version 2 (IKEv2) (No. RFC 6989).

[15] Smekal, D., Frolka, J., & Hajny, J. (2016). Acceleration of AES Encryption Algorithm Using Field Programmable Gate Arrays. IFAC-PapersOnLine, 49(25), 384-389.