Optimized and robust implementation of mobile networks confidentiality and integrity functions

Mahdi Madani, Camel Tanougast

Optimized and Robust Implementation of Mobile Networks Confidentiality and Integrity Functions

Mahdi Madani

Laboratory ImViA, University of Bourgogne Franche-Comté , Dijon, France
Email :mmadani49@gmail.com

Camel Tanougast

Laboratory LCOMS, University of Lorraine, Metz, France
Email :Camel.Tanougast@univ-lorraine.fr

Abstract

In mobile networks, sensitive information is transmitted over the radio link between the Base Station (BS) and the Mobile Station (MS). Therefore, a security mechanism based on confidentiality and integrity functions is designed to protect the anonymity of users'. More precisely, the confidentiality function was standardized by the Third Generation Partnership Project (3GPP) to protect users' data and the integrity function to protect control (signaling) data. However, both functions are based on the same kernel algorithm. For example, the functions used in the Universal Mobile Telecommunications System (UMTS) network are based on the algorithm, namely the KASUMI block cipher. In this work, we proposed an optimized function that combines the confidentiality and integrity functions in one Running-Block-Cipher and ensures the same functionalities as the basic functions (UMTS F8 and F9 functions). We used an architectural synthesis technique that allows for achieving a significant reduction in the area occupied on the hardware device. The designed architecture was implemented on Xilinx Virtex Field Programmable Gate Arrays (FPGA) technology. The synthesis results after a place-and-route and comparison with previous works show the good performance of the proposed architecture in terms of throughput, consumed energy, and occupied hardware logic resources.

Keywords: Mobile Networks, FPGA Implementation, Confidentiality, Integrity, Architectural synthesis

1. Introduction

Today, mobile networks and Internet applications (e-mail, e-commerce, e-learning, etc.) occupy an important part of our lives and transfer a lot of personal information via a radio link that is not protected physically. Consequently, service providers must protect all data exchanged on the channel used (wireless)

to preserve the anonymity of users despite the difficulties encountered. For this reason, security mechanisms based on encryption algorithms are used to ensure the security of data transmitted over mobile networks.

In this paper, we propose a Running-Block-Cipher to ensure the confidentiality and integrity of users' and control (signaling) data. The proposed architecture is based on the standardized F8 confidentiality and F9 integrity functions designed by the Third Generation Partnership Project (3GPP) for the Universal Mobile Telecommunications System (UMTS) network [1–3]. We fixed two main objectives. Firstly, we combined the internal blocks of the original F8 and F9 functions to reduce the required area for hardware implementation [4, 5]. The adopted technique is based on the reuse of common blocks for original functions [6, 7], principally KASUMI algorithm [8–10]. Secondly, we simplified the implementation of the internal functions (FL, FO, and FI) of the KASUMI block cipher (the core of standard F8 and F9 functions) to increase the encryption throughput. The connection between the different levels of hierarchy and the control of the internal signals in the proposed architecture relies on the use of Moors' Finite State Machines (FSM) [11, 12].

Register-Transfer-Level (RTL) abstraction is used in VHSIC Hardware Description Language (VHDL) to create high-level representations of the proposed architecture. The design has been implemented on the ML507 Virtex development platform. To prove the good performance of the proposed Running-Block-Cipher, we compared the synthesis results after a place-and-route with previous implementation [13], in terms of throughput and occupied hardware logic resources. To test and validate the proposed design, we compared the generated outputs with the 3GPP standardized documents [1, 2, 14]

The rest of this paper is organized as follows. In Section 2 the related work is discussed. In Sections 3, the standard F8 and F9 functions are briefly described. The proposed Running-Block is detailed in Section 4. The Field Programmable Gate Arrays (FPGA) implementation results, comparison of the proposed architecture with previous works, and discussion is the subject of Section 5. Finally, the paper ends with some conclusions in Section 6.

2. Related work

In modern cellular networks, the wireless link is widely used to transmit sensitive data. Therefore, security mechanisms are used to prevent unauthorized access to user's information exchanged over this radio channel. For example, the UMTS security is based on the KASUMI block cipher. Because of the limited logic resources and the high throughput required by real-time embedded applications (mobile phones), the hardware implementation of this algorithm becomes complicated. Despite that, several proposals have been published that use different approaches to implement KASUMI in hardware, ranging from reuse techniques, the addition of internal registers to reduce critical path, pipelined designs, and specialized processor core extension for the KASUMI encryption algorithm [9, 10, 15–21].

The KASUMI block cipher is principally used to ensure high levels of confidentiality and integrity of information. The F8 and F9 functions are the 3GPP algorithms designed for this focus. Contrarily to the works [9, 10, 15–21] that implementing the KASUMI block cipher, Helion's work [13] has successfully implemented the F8 and F9 algorithms on Xilinx FPGA devices.

The objective of this work is to realize an optimized and robust implementation that outperforms the Hellion's implementation [13] in terms of throughput and occupied hardware resources. Unlike Hellion's approach based on the implementation of standardized 3GPP algorithms, we used an architectural synthesis technique (work [8]). We have simplified the internal functions of the common KASUMI block. Then, we have combined the internal blocks of F8 and F9 functions to form a single algorithm that ensures both the original functionalities.

After implementation and validation, the comparison of the proposed architecture with the 3GPP test set vectors [14] proves the generation of the expected outputs, and the FPGA implementation results prove the achievement of high performance.

3. F8 and F9 architectures

The UMTS security mechanism is based on two main functions, confidentiality function named F8 and integrity function named F9 [1, 22]. Both functions are based on the KASUMI block cipher used as the internal kernel algorithm. In this section, we begin by presenting the architecture and the main functions used to form KASUMI block cipher. After that, we present the internal architecture and the execution process of each of the confidentiality and integrity functions, respectively.

3.1. A general vision on KASUMI block cipher

KASUMI is a block cipher with a Feistel structure for eight rounds and developed on 3GPP. It operates in 64-bits block data and controlled by 128-bits Ciphering Key (CK). To generate output key-stream at each round, it executes two internal functions, FL and FO. It executes the FL then FO function in the odd rounds ((1, 3, 5, and 7), and the FO then FL function in the even rounds 2, 4, 6, and 8). The process begins by dividing the 64-bits input data into two 32-bits strings, left L0 and right R0. The outputs of the odd rounds are produced according to Equation 1 $for\ i = 1,\ 3,\ 5,\ and\ 7$, and the outputs of the even rounds are produced according to Equation 2 $for\ i = 2,\ 4,\ 6,\ and\ 8$.

$$L(i+1) = FO\ (FL\ (L(i),\ KLi\ ),\ KOi,\ KIi)\ xor\ R(i) \quad ; \quad R(i+1) = L(i) \quad (1)$$

$$L(i+1) = FL\ (FO\ (L(i),\ KOi,\ KIi),\ KLi)\ xor\ R(i) \quad ; \quad R(i+1) = L(i) \quad (2)$$

We not that CK is used at each round to derive 8 sub-keys of size 16-bits : $KLi1$ and $KLi2$ (used by FL function), $KOi1$, $KOi2$, and $KOi3$ (used by

FO function), $KIi1$, $KIi2$, and $KIi3$ (used by FI function). Where FI is an internal function of the FO function and based on S-box tables (S9 and S7). The algorithm architecture is shown in Figure 1.

For more details in the KASUMI components and specifications, please refer to [2].

3.2. Confidentiality function F8

As we said, the UMTS confidentiality is based on function F8. The ciphering process is based on the use of 5 input parameters : CK (128-bits), encryption sequence $COUNT-C$ (32-bits), the used channel identifier $BEARER$ (5-bits), $DIRECTION$ (1-bit), and the length of encrypted block $LENGTH$. The output of the F8 function is combined by a XOR operation with the plain-text to form the cipher-text. The receiver uses the same process to retrieve the plain-text. The confidentiality process is shown in Figure 2.

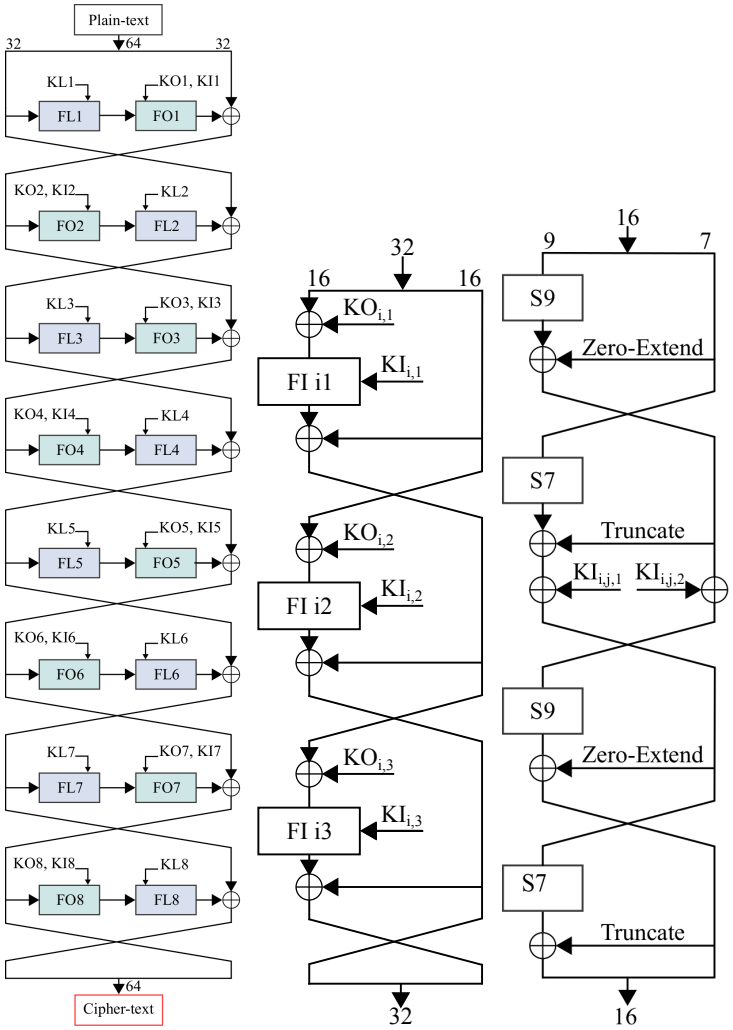
The internal architecture of the function F8 is presented in Figure 3. It is composed of several KASUMI blocks that are connected to generate the output keystream. The input to the first block is formed by concatenating the parameters $COUNT-C$, $BEARER$, $DIRECTION$, and $LENGTH$. Then, the output is saved on register A. After that, it is combined with the $BLKCNT$ counter using a bitwise XOR operation. The results form the input of the next block. Note that the first block is controlled by the combination of CK with a fixed sequence named $KM1 = 0x55\ldots5$ (128-bits) using a bitwise XOR operation. But the other blocks are controlled only by CK.

For more details in the function F8, please refer to [1].

3.3. Integrity function F9

UMTS integrity is based on the function F9. The integrity process is based on the use of 5 input parameters : Integrity Key IK (128-bits), integrity sequence $COUNT-I$ (32-bits), the message to protect $MESSAGE$, $DIRECTION$ (1-bit), and a generated random sequence $FRESH$ (32-bits). The function F9 generates a 32-bits Message Authentication Code Integrity ($MAC-I$) that is sent with the message. The receiver follows the same process to generate the authentication code corresponding to the received message ($XMAC-I$). If the expected $XMAC-I$ and received $MAC-I$ codes are similar, the receiver concludes that there is no alteration in the content of the received message (no added, no deleted, and no modified parts). The integrity process is shown in Figure 4.

The internal architecture of the function F9 is presented in Figure 5. It is based on a string of KASUMI blocks. The input is formed by concatenating the parameters : $COUNT-I$, $FRESH$, $MESSAGE$, and $DIRECTION$. The result is divided into n 64-bits vectors $(PS0, PS1, \ldots, PSn)$. Where n is the number of 64-bits blocks formed by the concatenation sequence. Then, the outputs of the n blocks are combined by a bitwise XOR operations to form the input to an additional KASUMI block. The left half of the generated output forms the integrity code ($MAC-I$) of the processed message. Note that the



(a) Architecture of KASUMI block cipher

(b) The FO function

(c) The FI function

(d) The FL function

FIGURE 1: The Feistel structure of KASUMI.

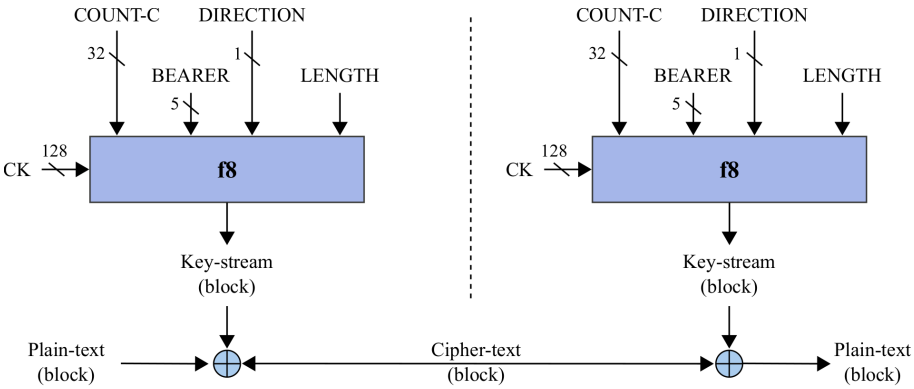


FIGURE 2: The UMTS confidentiality mechanism.

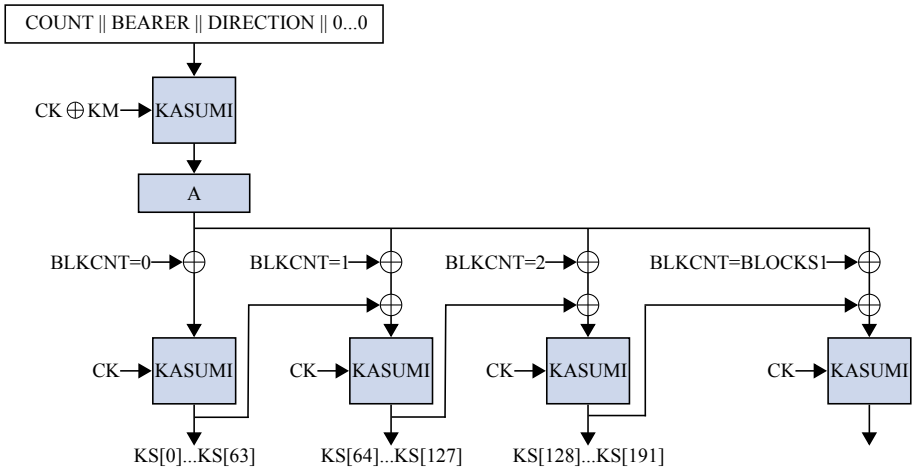


FIGURE 3: The F8 confidentiality function.

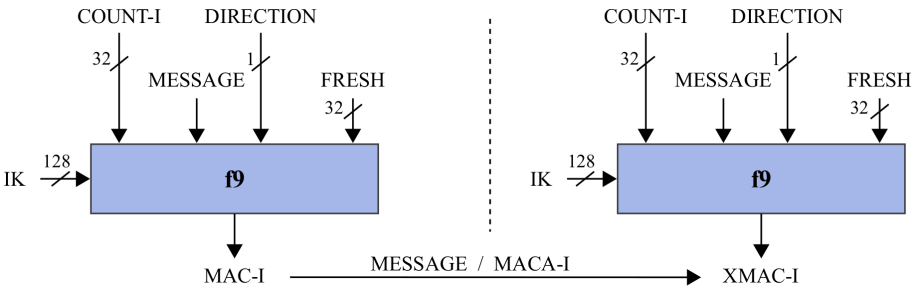


FIGURE 4: The UMTS integrity mechanism.

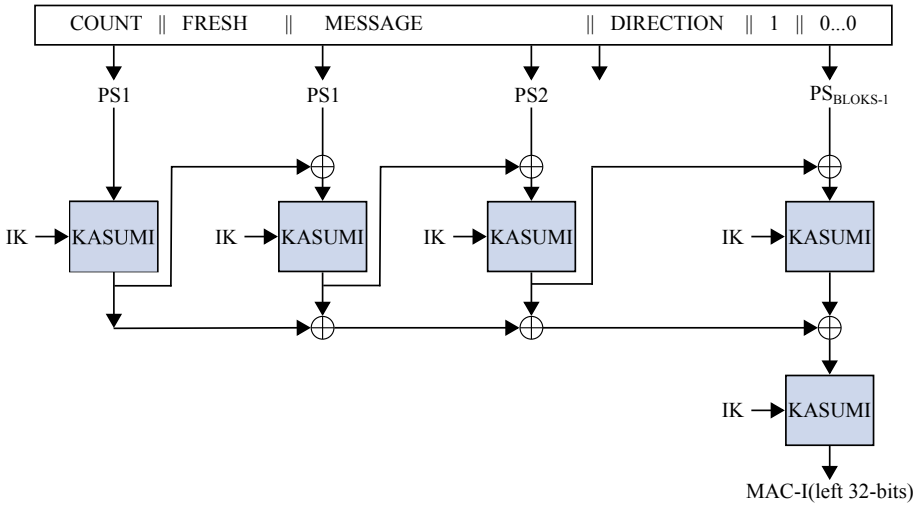


FIGURE 5: The F9 integrity function.

last block is controlled by the combination of IK with a fixed sequence named $KM2 = 0xAA \ldots A$ (128-bits) using a bitwise XOR operation. But the other blocks are only controlled by IK.

For more details in the F9 function, please refer to [1].

4. Proposed Running-Block

In this section, we present the proposed Running-Block architecture. The purpose is to reduce the required logic resources to hardware parallel processing on limited systems. To achieve this objective, we have developed an architecture that allows confidentiality F8 and integrity F9 functions to be executed using a single common kernel (KASUMI block cipher). Data routing and switching between the confidentiality and integrity operating modes in the proposed architecture is provided by a control block encoded as an FSM. The proposed architecture is illustrated in Figure 6, and the processing steps are presented as follows.

4.1. Confidentiality mode

The confidentiality mode is selected by the control unit. To generate the same output as the regular function F8, we begin by uploading the confidentiality input parameters ($COUNT - C$, $BEARER$, $DIRECTION$, and $LENGTH$) to $VecConf$ and initializing $BLKCNT$ and register A to zero. In the first round, the combination of CK and $KM1$ using a bitwise XOR operation is selected as the control key and $VecConf$ as the input to KASUMI. The output is saved in register A (update of register A). In the second round, the combination of

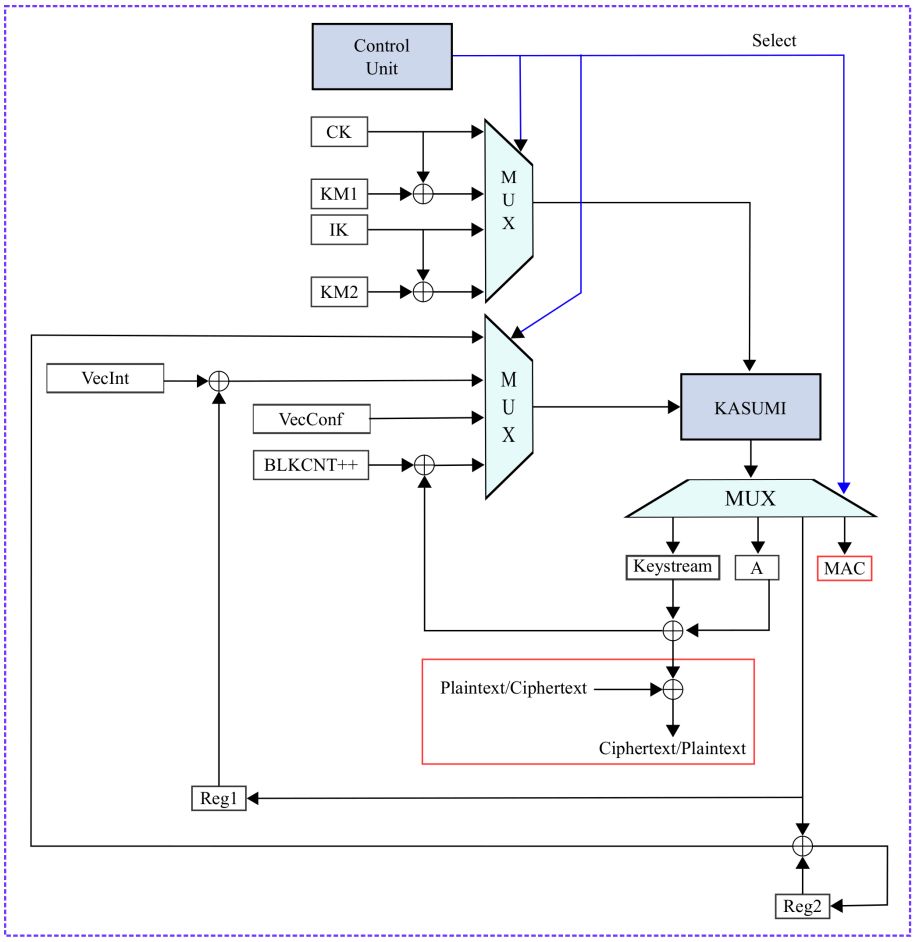


FIGURE 6: The proposed Running block.

register A and $BLKCNT$ using a bitwise XOR operation forms the new input to KASUMI, and the CK is selected as the new control key. The output is saved in $Keystream$ register. In the next rounds, the combination of register A, $Keystream$, and $BLKCNT$ forms the new input to KASUMI, and CK is always selected as the control key. Note that $BLKCNT$ is incremented at each round. The cipher-text is generated by combining the plain-text blocks (64-bits) with KASUMI output (from the second round) using a bitwise XOR operation. The process is repeated n times, where n is the number of 64-bits blocks to encrypt. The detailed process is shown in Figure 6.

4.2. Integrity mode

Like the confidentiality, the integrity mode is selected by the control unit. To generate the same output as the regular function F9, we begin by uploading the integrity input parameters ($COUNT-I$, $FRESH$, $MESSAGE$, and $DIRECTION$) to $vecInt$ and initializing the registers $Rge1$ and $Reg2$ to zero. Then, the $VecInt$ is divided into n 64-bits blocks (the last block is zero-extended if bits are missing). In the first round, IK is selected as the control key and the combination of register $Reg1$ with the first 64-bits block ($VecInt(1)$) using a bitwise XOR operation as input to KASUMI. The output is saved in the register $Reg1$ (updates of register $Reg1$) and then combined with register $Reg2$ using a bitwise XOR operation (updates of register $Reg2$). In the second round, the same process is repeated until the last 64-bits bock ($VecInt(n)$). At each cycle the registers $Reg1$ and $Reg2$ are updated. After that, an additional cycle will be executed. The last content of register $Reg2$ forms the input to KASUMI and the control key switches to applying the combination of IK with $KM2$ using a bitwise XOR operation as the new key. The left half of the generated output forms the integrity code $MAC-I$ (32-bits) of the processed message.

4.3. FSM control unit

To realize the Running-Block, we opt for a hardware behavior description based on FSM. This system describes the numerical resolution based on a control unit designed to manage the inputs (key and value) and output of the KASUMI block cipher. Thereby, the description is implemented in a digital way suitable for FPGA using the FSM system coded in VHDL language and detailed in Figure 7. This figure describes the FSM system corresponding to the internal states of the Running-Block. Detailed description at each state of the FSM system is presented as follows.

— State-init : It is the initialization state. All outputs are set to zero, and the confidentiality and integrity vectors (VecConf and VecInt) are initialized with the corresponding input parameters, as follows.
$VecConf = COUNT-C \| BEARER \| DIRECTION \| 0 \cdots 0$
$VecInt = COUNT-I \| FRESH \| MESSAGE \| DIRECTION \| 10 \cdots 0$
If we want to execute the confidentiality mode, the next state is State-Conf1, and if we want to execute the integrity mode, the next state is State-Int1.

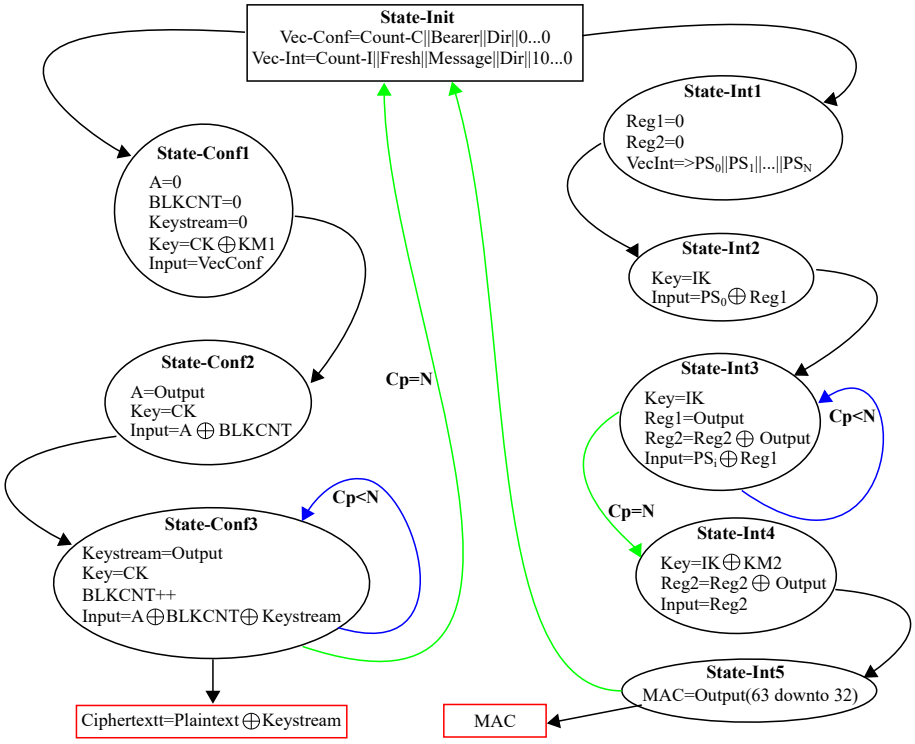


FIGURE 7: Description of the FSM system performing the proposed Running-Block.

— State-Conf1 : In this state, we load the input parameters to the KASUMI block. The control key is $CK \oplus KM1$ and input value is $VecConf$. The Content of the registers A, $Keystream$, and $BLKCNT$ are always set to zero. At the next rising edge of the clock signal, it jumps to the next state without any condition.
— State-Conf2 : The content of the register A is updated by the first generated output of the KASUMI block. After that, the new input parameters are modified and loaded to the KASUMI block. The control key is CK, and the input value is the result of the operation $A \oplus BLKCNT \oplus Keystream$. At the next rising edge of the clock signal, it jumps to the next state without any condition.
— State-Conf3 : The content of the register $Keystream$ is updated by the generated output of the KASUMI block. The ciphertext is then generated by combining a 64-bits block of the plaintext with a keystream, $ciphertext = plaintext \oplus Keystream$. After that, the content of $BLKCNT$ is incremented, and the input parameters (CK and $A \oplus BLKCNT \oplus Keystream$) are loaded to the KASUMI block. At the next rising edge of the clock signal, if $Cp < N$, then it repeats the State-Conf3 cycle to generate a new keystream. But, if $Cp = N$, which indicates that all the required 64-bits blocks of plaintext are encrypted, the FSM jumps to State-Init to wait for a new order of confidentiality or integrity function execution. Note that Cp is a command to control the end of plaintext blocks, and N is the number of 64-bits blocks to be encrypted.
— State-Int1 : In this state, we divide the content of $VecInt$ to N 64-bits blocks ($PS_0, PS_1, \cdots, PS_N$). The content of the registers $Reg1$ and $Reg2$ are always set to zero. At the next rising edge of the clock signal, it jumps to the next state without any condition.
— State-Int2 : We load the input parameters to the KASUMI block. The control key is IK and input value is $PS_0 \oplus Reg1$. At the next rising edge of the clock signal, it jumps to the next state without any condition.
— State-Int3 : The generated output is used to update to content of the registers $Reg1$ ($Reg1 = Output$) and $Reg2$ ($Reg2 = Reg2 \oplus Output$). After that, the input parameters (IK and $Reg1 \oplus PS_1$) are loaded to the KASUMI block. At the next rising edge of the clock signal, if $Cp < N$, then it repeats the State-Int3 cycle to generate a new keystream. But, if $Cp = N$, which indicates that all the required N 64-bits blocks of divided $VecInt$ are processed, the FSM jumps to State-Int4. Note that Cp is a command to control the end of processing the expected N formed 64-bits blocks.
— State-Int4 : In this state, the inputs parameters are modified then loaded to the KASUMI block. The control key is the result of $IK \oplus KM2$, and the input value is the last content of $Reg2$. At the next rising edge of the clock signal, it jumps to the next state without any condition.
— State-Int5 : The left half of the generated output is selected as the integrity code $MAC - I$ (32-bits) of the processed message. At the next rising edge of the clock signal, the FSM jumps to State-Init to wait for a

new order of confidentiality or integrity function execution.
Note that parameter $\oplus$ is used to design a bitwise XOR operation.

5. Implementation, comparison and discussion

In this section, we begin by presenting FPGA implementation results. After that, we give a comparison of the proposed architecture with previous works in terms of throughput, occupied logic resources, and generated output. Finally, we discuss the obtained results.

5.1. Implementation results

The proposed architecture has been described using the VHDL language and has been implemented on Xilinx Virtex FPGA technology [23–25]. Integrated Synthesis Environment (ISE) 13.2 of Xilinx tools have been used for this digital implementation allowing us to obtain the logic resource requirements and the associated real-time constraints. The proposed Running-Block synthesis results after a place-and-route are presented in Table 1. Note that the throughput is calculated using Equation 3.

$$Throughput = \frac{Block\ size \times Clock\ frequency}{Latency\ cycle} \tag{3}$$

Device	Function mode	Frequency Mhz	Latency cycle	Throughput Mbps	Area slice
Virtex-E	f8	91.208	3	1945.771	1434
	f9		11	265.332	
Virtex-II	f8	138.466	3	2953.941	1426
	f9		11	402.810	
Virtex-5	f8	274.771	3	5861.781	722
	f9		11	799.334	
Virtex-6	f8	372.815	3	7953.387	586
	f9		11	1084.553	

Table 1: FPGA implementation results of the proposed Running-Block.

From the obtained results, we remark that good performances are achieved by the proposed implementation in terms of throughput (7953.387 Mbps and 3 latency cycles with the mode confidentiality, 1084.553 Mbps and 11 latency cycles with the mode integrity), occupied hardware logic resources (586 slices), and low power consumption (estimated by 1.46 Watts on FPGA device).

All these results prove that the proposed architecture is suitable for embedded applications designed for real-time communications (mobile networks). Consequently, we conclude that the simplified Running-Block can ensure both the confidentiality and integrity functions using an optimized architecture with low resources and efficiently occupy the device.

5.2. FPGA implementation comparison

The FPGA implementation results of proposed and previous works are presented in Table 2. From the results obtained, we conclude that the proposed architecture is faster than the previous work in terms of maximum frequency and throughput achieved. The designed implementation is 8.4 times faster in Virtex-5, and 9.8 times faster in Virtex-6 than work in [13]. In terms of logic resources, the proposed and work in [13] occupies almost the same hardware area.

By analyzing the comparison results presented in Table 2 and the addressed remarks, it is clear that the proposed architecture achieves the highest throughput due to the high processing clock rate and low latency cycles. Finally, we conclude that the proposed Running-Block can provide fast protection (confidentiality and integrity) of users' data, which is considered as an important parameter in real-time applications, like mobile network communications.

Source	Device	Function	Freqency MHz	Throughput Mbps	Area slice
Helion[13]	Virtex-5	f8	186	700	548
		f9	186	700	
	Virtex-6	f8	215	808	564
		f9	221	830	
This work	Virtex-5	f8	274.771	5861.781	722
		f9	274.771	799.343	
	Virtex-6	f8	372.815	7953.387	586
		f9	372.815	1084.553	

TABLE 2: FPGA comparison implementations.

5.3. Generated outputs comparison

To test and validate the generated outputs, we used the 3GPP test set vector [14] designed to help implementers in their realization. This document provides test data for the algorithms as well as details on the internal states of the algorithms when they process the input data. For example, the test set vectors 1 is defi

ned as follows.

— The functions F8 and F9 inputs are presented in Table 3.

— The plaintext and the corresponding ciphertext are presented in Table 4.

The ISE Simulator (ISim) has been used to test the exactness of generated outputs compared to the 3GPP reference data presented in Tables 3 and 4 . The simulation results (VHDL test bench waveform) are shown in Figures 8 and 9.

By comparing outputs generated by the original F8 and F9 functions presented in Tables 4 and Table 3 with the outputs generated by the proposed Running-Block presented in Figures 8 and 9, respectively, it is clear that the proposed implementation generates the same and expected outputs (ciphertext and integrity code $MAC-I$) like the original ones. Consequently, we conclude

Confidentiality function F8 inputs		
Key	=	$2BD6459F82C5B300952C49104881FF48$
$Count-C$	=	$72A4F20F$
$Bearer$	=	$0C$
$Direction$	=	1
$Length$	=	$798bits$
Integrity function F9 inputs/outputs		
$Count-I$	=	$38A6F056$
$Fresh$	=	$05D2EC49$
$Direction$	=	0
$Length$	=	$189bits$
$MAC-I$	=	$F63BD72C$
$Message$	=	$6B227737296F393C8079353EDC87E2E805D2EC49A4F2D8E0$

Table 3: 3GPP Test set vector1 confidentiality/integrity inputs/outputs.

Plaintext	Ciphertext
7EC61272743BF161	D1E2DE70EEF86C69
4726446A6C38CED1	64FB542BC2D460AA
66F6CA76EB543004	BFAA10A4A093262B
4286346CEF130F92	7D199E706FC2D489
922B03450D3A9975	1553296910F3A973
E5BD2EA0EB55AD8E	012682E41C4E2B02
1B199E3EC4316020	BE2017B7253BBF93
E9A1B285E7627953	09DE5819CB42E819
59B7BDFD39BEF4B2	56F4C99BC9765CAF
484583D5AFE082AE	53B1D0BB8279826A
E638BF5FD5A6061	DBBC5522E915C120
3901A08F4AB41AAB	A618A5A7F5E89708
9B134880	9339650F

Table 4: 3GPP Test set vector1 plaintext/ciphertext.

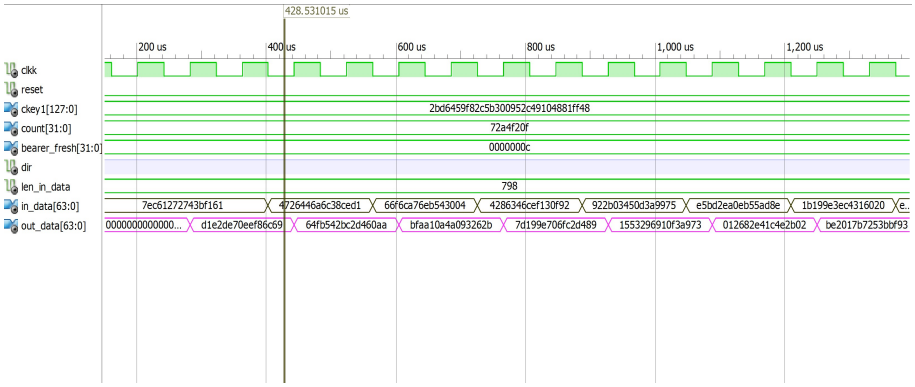


FIGURE 8: ISim F8 simulation results.

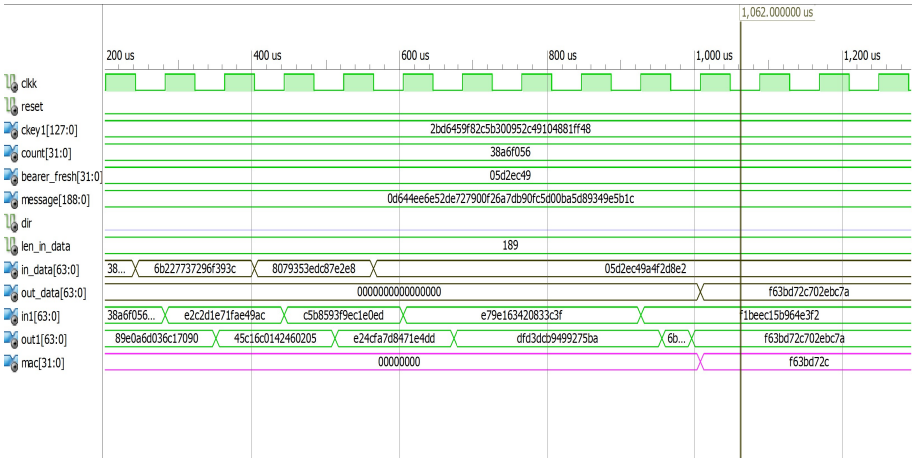


FIGURE 9: ISim F9 simulation results.

that the design proposed in this study can ensure the same functionalities compared to the original F8 and F9 functions.
Note that the proposed algorithm satisfies all of the test data sets provided in [14].

5.4. Discussion

According to the proposed architecture illustrated in Figure 6 and results presented in Tables 1 and 2, we remark that the proposed Running-Block requires low hardware logic resources (slice registers) and provides high throughput. Besides, it ensures the same functionalities that the original confidentiality F8 and integrity F9 functions, and generates the expected outputs fixed in the 3GPP test vector set1 [14], as presented in Figures 8and 9.

Indeed, the proposed architecture presents a simple, efficient, and less complicated implementation in a hardware device (FPGA technology). Also, the characterizing good performances prove that proposal architecture is more suitable for use in UMTS networks. Consequently, the proposed Running-Block architecture can replace both the old confidentiality F8 and integrity F9 functions while satisfying the network requirements (time constraint, size of data...). Additionally, it is suitable for embedded applications (handsets), in particular for a mobile station in our case study.

In terms of security, the proposed architecture is robust against the linear and nonlinear cryptanalysis, like the kernel (KASUMI) of original F8 and F9 functions. However, in the last decade, KASUMI was the subject of many cryptanalysis attacks, such as [26–31]. Therefore, the improvement of the KASUMI block security to resist the existing cryptanalysis attacks and to upgrade the knowing drawbacks is the main perspective of our future works.

5.5. Potential uses

The proposed design can be used in the UMTS security mechanism to replace the separated confidentiality F8 and integrity F9 functions.

6. Conclusion

The main contribution of this work was the implementation of UMTS confidentiality F8 and integrity F9 functions in the same hardware architecture. The technique used consists of keeping the kernel of the original functions (KASUMI block encryption) as the core of the proposed Running-Block. Then, we developed a control unit based on an FSM to ensure the routing of data in two operating modes (confidentiality and integrity modes) to generate the same outputs as the original functions.

The proposed architecture has been performed and implemented on an FPGA Virtex technology using VHDL structural description. The synthesis implementation results after place-and-route show the good performance of the optimized Running-Block. Consequently, the final proposed architecture allows us to increase data protection (confidentiality and integrity) and to decrease the occupied area in the handset compared to the original functions. By considering the

obtained good trade-off between time performance and logic resources, compared with previous works, we conclude that the proposed design forms the best FPGA implementation designed to ensure data transmitted over the third generations of mobile networks.

Finally, we conclude that this proposed architecture is still satisfying the real-time and embedded systems properties while preserving the 3GPP standardized requirements (key length, data size, etc.). Therefore, it is possible to replace the old F8 and F9 functions with the proposed optimized Running-Block, which can ensure the same functionalities with more performances.

Références

[1] 3G Security ; Specification of the 3GPP Confidentiality and Integrity Algorithms ; Document 1 : f8 and f9 specification, Technical Specification (TS) TS 35.201 V15.0.0, 3GPP (Jun 2018-06).

[2] 3G Security ; Specification of the 3GPP Confidentiality and Integrity Algorithms ; Document 2 : KASUMI specification, Technical Specification (TS) TS 35.202 V15.0.0, 3GPP (Jun 2018-06).

[3] A. Sankaliya, V. Mishra, A. Mandloi, Implimentation of Cryptographic Algorithm for GSM and UMTS Systems, International Journal of Network Security & Its Applications 3 (6) (2011) 81–88 (2011).

[4] K. Vikas, N. Bhushan, B. Vinayak, k. S. K. Narayan, Next Generation Encryption using Security Enhancement Algorithms for End to End Data Transmission in 3G/4G Networks, Procedia Computer Science 79 (2016) 1051 – 1059 (2016). doi:https://doi.org/10.1016/j.procs.2016.03.133.

[5] S. David, H. Jan, M. Zdenek, Comparative Analysis of Different Implementations of Encryption Algorithms on FPGA Network Cards, IFAC-PapersOnLine 51 (2018) 312–317 (2018). doi:https://doi.org/10.1016/j.ifacol.2018.07.172.

[6] M. Madani, C. Taougast, Combined and Robust SNOW-ZUC Algorithm Based on Chaotic System, in : The International Conference on Cyber Security and Protection of Digital Services (Cyber Security 2018), IEEE Conference Publications, 2018 (2018).

[7] T. Liu, C. Tanougast, S. Weber, A Framework of Architectural Synthesis for Dynamically Reconfigurable FPGAs, in : IEEE Circuits and Systems Society (Ed.), IEEE International SOC Conference 2008, IEEE Circuits and Systems Society, Newport Beach-California, USA, 2008, pp. 283–286 (2008).

[8] M. Madani, S. Chitroub, C. Tanougast, Two KASUMI Components for an Optimal Implementation of the A5/3 Algorithm, in : The International Conference on Circuits, System and Simulation (ICCSS 2017), IEEE Conference Publications, London, UK July 14-17, 2017, pp. 124–128 (2017).

[9] Y. Dai, I. Kouichi, Y. Jun, A Very Compact Hardware Implementation of the KASUMI Block cipher, in : LNCS 6033, 2010, pp. 293–307 (2010).

[10] P. Kitsos, M. D. Galanis, O. Koufopavlou, High-speed hardware implementations of the KASUMI block cipher, in : ISCAS 2004, Vol. 2, 2004, pp. 549–552 (2004).

[11] M. Madani, I. Benkhaddra, C. Taougast, S. Chitroub, L. Sieler, Digital Implementation of an Improved LTE Stream Cipher SNOW-3G based on Hyperchaotic PRNG, Security and Communication Networks 2007 (2017). doi:https://doi.org/10.1155/2017/5746976.

[12] M. Madani, I. Benkhaddra, C. Taougast, S. Chitroub, L. Sieler, FPGA Implementation of an enhanced SNOW-3G Stream Cipher based on a Hyperchaotic System, in : The 4th international conference on Control, Decision and Information Technologies (CoDIT'17), IEEE Conference Publications, 2017 (2017).

[13] DATASHEET-3GPP KASUMI f8 and f9 Cores for Xilinx FPGA, Ash House, Breckenwood Road, Fulbourn, Cambridge CB21 5DQ, England Revision 1.2, Helion Technology Limited (Sep 2009-09).

[14] 3G Security ; Specification of the 3GPP Confidentiality and Integrity Algorithms ; Document 3 : Implementors' test data, Technical Specification (TS) TS 35.203 V15.0.0, 3GPP (Jun 2018-06).

[15] B. T, C. R, An efficient reuse-based approach to implement the 3GPP KASUMI block cipher, in : first international conference on electrical and electronics engineering and 10th conference on electrical engineering ICEEE/CIE, Acapulco, Mexico, 2004, pp. 113–18 (2004).

[16] B. T, C. R, An efficient hardware implementation of the KASUMI block cipher for third generation cellular networks, in : global signal processing conference, GSPx , Santa Clara, CA, 2004 (2004).

[17] T. Balderas-Contreras, R. Cumplido, C. Feregrino-Uribe, On the design and implementation of a RISC processor extension for the KASUMI encryption algorithm, Computers & Electrical Engineering 34 (2008) 531–546 (2008). doi:https://doi.org/10.1016/j.compeleceng.2007.11.003.

[18] X. ZHAO, S.-x. GUO, High-Performance Hardware Implementation of the 3GPP Algorithm KASUMI, The Journal of China Universities of Posts and Telecommunications 13 (2006) 60–62 (2006). doi:https://doi.org/10.1016/S1005-8885(07)60082-X.

[19] K. Marinis, N. Moshopoulos, F. Karoubalis, K. Pekmestzi, On the hardware implementation of the 3GPP confidentiality and integrity algorithms, in : the 4th International Conference for the Information Security, ISC 2001, Malaga, Spain, 2001, p. 248–265 (October 1-3 2001).

[20] H. Kim, Y. Choi, M. Kim, H. Ryu, Hardware Implementation of 3GPP KASUMI Crypto Algorithm, in : The 2002 International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), Phuket, Thailand, 2002, p. 317–320 (July 16-19 2002).

[21] A. Satoh, S. Morioka, Small and high-speed hardware architectures for the 3GPP standard cipher KASUMI, in : the 5th International Conference Information Security, ISC 2002 (2002), Lecture Notes in Computer Science 2433 Springer-Verlag, Sao Paulo, Brazil, 2002 (September 30 - October 2 2002).

[22] N. Valtteri, K. Nyberg, UMTS Security, John Wiley & Sons Ltd, England, 2003 (2003).

[23] Virtex devices specification, `http://www.xilinx.com/`.

[24] Xilinx, Virtex^{TM}-E 1.8 V Field Programmable Gate Arrays, Production Product Specification, DS022-1 (v3.0) March 21, 2014 (DS022-1 (v3.0) March 21, 2014).

[25] Virtex 5 FPGA Configuration User Guide, ug702, Xilinx.

[26] Y. Wentan, C. Shaozhen, Multidimensional zero-correlation linear cryptanalysis of the block cipher KASUMI, IET Information Security 10 (2016).

[27] Z. Wang, X. Dong, K. Jia, J. Zhao, Diffeential Fault Attack on KASUMI Cipher Used in GSM Telephony, Mathematical Problems in Engineering 2014 (2014) 7 pages (2014). `doi:http://dx.doi.org/10.1155/2014/251853`.

[28] O. Dunkelman, N. Keller, A. Shamir, A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony, Journal of Cryptology 27 (2014).

[29] K. Jia, L. Li, C. Rechberger, J. Chen, X. Wang, Improved Cryptanalysis of the Block Cipher KASUMI, in : International Conference on Selected Areas in Cryptography SAC 2012, Lecture Notes in Computer Science, volume 7707, Windsor, Canada, 2012, pp. 222–233 (Aug 2012).

[30] E. Biham, O. Dunkelman, N. Keller, A related-key rectangle attack on the full KASUMI, in : International Conference on the Theory and Application of Cryptology and Information Security ASIACRYPT 2005, LNCS, volume 3788, 2005, pp. 443–461 (2005).

[31] M. Blunden, A. Escott, Related Key Attacks on Reduced Round KASUMI, in : International Workshop on Fast Software Encryption FSE 2001, LNCS, volume 2355, 2002, pp. 277–285 (2002).