Comparison of the Hardware Implementation of Stream Ciphers

Michalis Galanis, Paris Kitsos, Giorgos Kostopoulos, Nicolas Sklavos, and Costas Goutis
Electrical and Computer Engineering Department, University of Patras, Greece

Abstract: *In this paper, the hardware implementations of five representative stream ciphers are compared in terms of performance and consumed area in an FPGA device. The ciphers used for the comparison are the A5/1, W7, E0, RC4 and Helix. The first three ones have been used for the security part of well-known standards, especially wireless communication protocols. The Helix cipher is a recently introduced fast, word oriented, stream cipher. W7 algorithm has been recently proposed as a more trustworthy solution for GSM, due to the security problems concerning A5/1. The designs were implemented using VHDL language. For the hardware implementation of the designs, an FPGA device was used. The implementation results illustrate the hardware performance of each stream cipher in terms of throughput-to-area ratio. This ratio equals to: 5.88 for the A5/1, 1.26 for the W7, 0.21 for the E0, 2.45 for the Helix and 0.86 for the RC4.*

Keywords: *Cryptography, security,s tream ciphers, hardware architecture, FPGA implementation.*

Received April 27, 200; accepted July 28200 4

1. Introduction

Cryptography works out with problems, which are associated with secrecy, authentication and integrity. Cryptography is also closely related with the meaning of protocol. A protocol consists of sequences of actions, which concern two or more sides, and it is designed to fulfill a goal. A protocol uses a cryptographic algorithm that its intention is to prevent attempts of thefts and invasions.

Cryptographic algorithms are divided between those that are *secret key* or *symmetric*, and those that are *public key* or *asymmetric*. With the latter one, the sender uses publicly known information to send a message to the receiver. Then, the receiver uses secret information to recover the message. In secret key cryptography, the sender and receiver have previously agreed on some private information that they use for both encryption and decryption.

Secret key cryptographic systems can be categorized into either *block* or *stream* ciphers. Block ciphers are memoryless algorithms that permute *N*-bit blocks of plaintext data under the influence of the secret key and generate *N*-bit blocks of encrypted data. Stream ciphers contain internal states and typically operate serially by generating a stream of pseudo-random key bits, the keystream (stream ciphers are also called *keystream generators*). The keystream is then bitwise XORed with the data to encrypt/decrypt.

Stream ciphers do not suffer from the error propagation, as in the block ones, because each bit is independently encrypted/decrypted from any other. They are generally much faster than block ciphers and they have greater software efficiency. Due to these features stream ciphers have been the choice for several communication protocols, especially wireless ones, like the IEEE 802.11b [10] and the Bluetooth [1].

The hardware implementation of cryptographic algorithms plays an important role because of the growing requirements for high-speed and high-level of secure communications. However, these algorithms impose tremendous processing power demands that can be a bottleneck in high-speed networks. Modern applied cryptography in the communication networks, demands high data processing rate to fully utilize the available network bandwidth. To follow the variety and the rapid changes in algorithms and standards, a cryptographic implementation also needs to support different algorithms and be upgradeable in field.

Field Programmable Gate Arrays (FPGAs) are a highly promising alternative to ASICs and general-purpose computers for implementing cryptographic algorithms. They are programmable devices, where the computation is performed by the logic cells and the connections among the cells are reconfigurable. A logic cell usually consists of Look-Up Tables (LUTs), carry logic, flip-flops, and programmable multiplexers.

Implementations of cryptographic algorithms in FPGA devices usually achieve superior performance when compared with software-based ones. The first main reason is that the fine-granularity of FPGAs matches extremely well the operations required by cryptographic algorithms (e. g., bit-permutations, bit-substitutions, Boolean functions). As a result, such operations can be executed more efficiently in FPGAs than in a general-purpose computer. The second reason is that the inherent parallelism of these algorithms can be efficiently exploited in FPGAs, as opposed to the

serial fashion of computing in a general-purpose processor.

There is a great number of stream cipher algorithms proposed both in academia and in industry. Five of them have been chosen, implemented in an FPGA device and compared in this paper. A5/1, E0 and RC4 are stream ciphers that they have been specified in popular communication standards and protocols; the A5/1 in GSM [4], the E0 in Bluetooth [1], and the RC4 in IEEE 802.11b. Helix is a word-oriented stream cipher, which also provides Message Authentication Code (MAC) function. Its functions are easily implemented and it is faster (in software implementations) than the best Advanced Encryption Standard (AES) implementation [5]. The W7 algorithm is a synchronous stream-cipher optimized for efficient hardware implementation at very high data rates [9]. W7 has been proposed in order to replace A5/1 in GSM security scheme, due to the security weaknesses of the A5/1 [3].

The rest of the paper is organized as follows. Section 2 describes the E0 cipher, while section 3 the A5/1. Sections 4 and 5 present the W7 and the Helix ciphers, respectively. RC4 cipher is presented in section 6. The hardware designs of the ciphers are presented in section 7, while the implementation results are analyzed in section 8. Finally, section 9 draws the conclusions for this stream cipher comparison.

2. E0 Cipher

The encryption of packet payloads in Bluetooth is performed by the E0 stream cipher [1], which consists of three components, as illustrated in Figure 1. The first component is the *payload key generator*, which performs the initialization (payload key generation). The second one, the *keystream generato*r, generates the keystream bits, and uses for this purpose four Linear Feedback Shift Registers (LFSRs), whose output is the input of a 16-state finite-state machine (called the *summation combiner*). The state machine output is the keystream sequence or the randomized initial start value during the initialization phase. The lengths L_i of the four LFSRs are 25, 31, 33, 39, and their feedback polynomials $f_i(x)$ are: $x^{25} + x^{20} + x^{12} + x^8 + 1$, $x^{31} + x^{24} + x^{16} + x^{12} + 1$, $x^{33} + x^{28} + x^{24} + x^4 + 1$, $x^{39} + x^{36} + x^{28} + x^4 + 1$, respectively, with $i = 1, 2, 3, 4$.

For the LFSRs initialization, the keystream generator needs to be loaded with an initial value for the four LFSRs (128 bits in total) and with 4 bits that specify the values of registers in the summation combiner. The 132-bit initial value is derived from four inputs by using the keystream generator itself. The input parameters are the encryption key K_c, a 128-bit random number, a 48-bit Bluetooth address, and the 26 master clock bits. Within the payload key generator, the K_c is modified into another key denoted K'_c, by using the polynomial modulo operation described in [1]. The maximum effective size of this key is factory preset and may be set to any multiple of eight; between one and sixteen (8-128 bits).

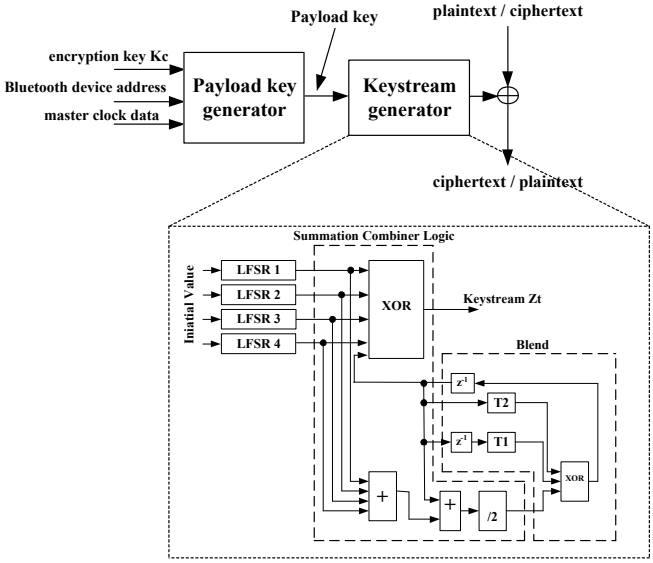

Figure 1. The E0 stream cipher architecture.

When the encryption key has been created, all the bits are shifted into the LFSRs, starting with the least significant bit. Then, 200 stream cipher bits are created by operating the generator. The last 128 of these bits are fed back into the keystream generator as an initial value of the four LFSRs. The values of the state machine are preserved. From this point on (i. e., after 239 cipher bits), the generator produces the encryption (decryption) sequence, when it is clocked. The produced sequence is bitwise XORed with the transmitted (received) payload data, in the *third component* of the cipher.

3. A5/1 Cipher

A5/1 is a stream cipher used for encrypting over the air transmissions in the GSM standard [4]. A GSM conversation is transmitted as a sequence of 228-bit frames (114 bits in each direction) every 4.6 millisecond. Each frame is XORed with a 228-bit sequence produced by the A5/1 keystream generator. The initial state of this generator depends on a 64-bit secret key K_c, which is fixed during the conversation, and on a 22-bit public frame number F_n.

The A5/1 cipher is composed by three LFSRs; R_1, R_2, and R_3 of lengths 19, 22, and 23 bits, respectively. Each LFSR is shifted, using clock cycles that are determined by a majority function m. The majority function uses three bits C_1, C_2, and C_3. Among these bits, if two or more of them are 0, then $m = 0$. Similarly, if two or more of these bits are 1, then $m = 1$. If $C_k = m$ then R_k is shifted, where $k = 1, 2, 3$. The feedback polynomials for R_1, R_2, R_3 are: $x^{19} + x^5 + x^2 + x + 1$, $x^{22} + x + 1$ and $x^{23} + x^{15} + x^2 + x + 1$,

respectively. At each cycle, after the initialization phase, the last bits of each LFSR are XORed to produce one output bit. The proposed architecture for the hardware implementation of the A5/1 cipher is shown in Figure 2.

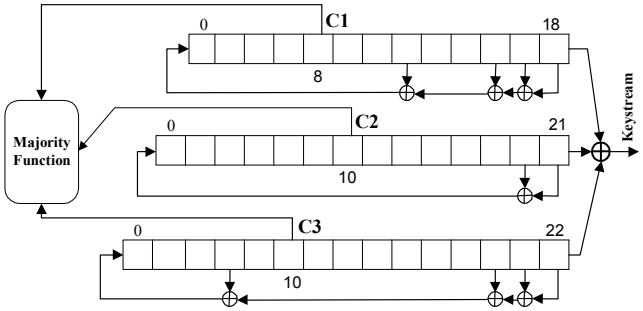

Figure 2. A5/1 stream cipher proposed architecture.

The process of generating the keystream bits from the key K_c and the frame number F_n is performed in four steps. In *step 1*, all the LFSRs are initialized to zero. Then the bits of K_c, starting from the least significant bit, are shifted into the three LFSRs in parallel, ignoring the majority function. During each cycle, the current bit from K_c is fed in and XORed with bit 0 of each LFSR.

In *step 2*, the 22 bits of F_n are fed in using the same process as in step 1. In *step 3*, 100 additional cycles are performed using the majority function, but without any output. Finally in *step 4*, another 228 cycles are required to get the 228 pseudo-random keystream bits.

4. W7 Cipher

The W7 algorithm is a symmetric key stream-cipher that supports key lengths of 128 bits. W7 cipher contains eight similar models, C1, C2, …, C8. Each model consists of three LFSR's and one majority function.

W7 architecture is composed by a control and a function unit. The function unit is responsible for the keystream generation. This unit contains eight similar cells. The proposed architecture for the hardware implementation of one cell is presented in Figure 3. Each cell has two inputs and one output. The one input is the key and it is the same for all the cells. The other input consists of control signals. Finally, the output is 1-bit long. The outputs of each cell compose the keystream byte.

Each cell consists of three LFSRs, 38-, 43- and 47-bit long, and a majority function. The initial state of the LFSRs, which is the same for all cells, is the symmetric encryption key. The 128 bits of the key map to the LFSRs' initial state as:

LFSRa (38-bit): $LFSR_0 = K_0, LFSR_1 = K_1, \ldots, LFSR_{36} = K_{36}, LFSR_{37} = K_{37}$

LFSRb (43-bit): $LFSR_0 = K_{38}, LFSR_1 = K_{39}, \ldots, LFSR_{41} = K_{79}, LFSR_{42} = K_{80}$

LFSRc (47-bit): $LFSR_0 = K_{81}, LFSR_1 = K_{82}, \ldots, LFSR_{45} = K_{126}, LFSR_{46} = K_{127}$

The three LFSRs together determine when each shift register is clocked. One bit in each register is designated as the clock tap for that register, as it is shown in Figure 3. At each clock cycle the majority value for these taps determines which LFSRs advance. Only the LFSRs, whose clock taps agree with the majority, advance. The output bit arises after a non-linear function in the register which is a combination of several bits in the LFSR, as presented in Figure 3. The non-linear function is a combination of logical-AND functions. The actual keystream output is taken as the exclusive-OR (XOR) of the three LFSRs. The keystream byte is the aggregation of each cell output.

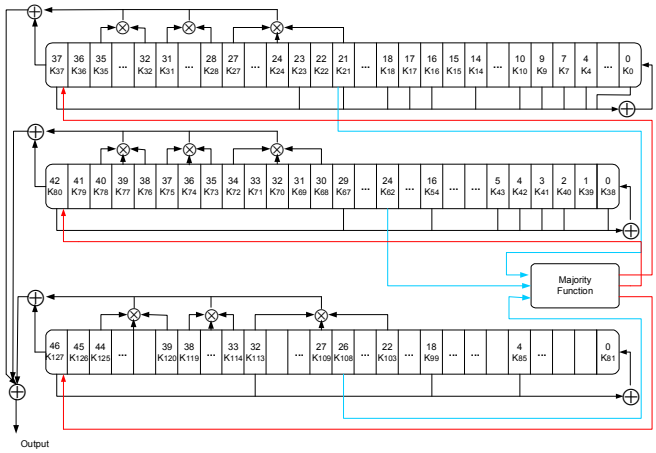

Figure 3. Proposed architecture for a W7 cell.

5. Helix Cipher

Helix [5] is a combined stream cipher and MAC function that directly provides the authenticated encryption functionality. By incorporating the plaintext into the stream cipher state, Helix can provide the authentication functionality without extra costs.

Helix uses a 256-bit key and a 128-bit parameter (called *nonce*). The key is secret, and the nonce is typically public knowledge. All operations in Helix are on 32-bit words. These operations are addition modulo 2^{32} (denoted $\boxplus$), XOR (denoted $\oplus$), and left rotation by fixed numbers of bits (denoted <<<). The design philosophy of Helix can be summarized as "many simple rounds". Helix has a state that is composed by 5 words (Z_0 to Z_4) of 32 bits each. A single round of Helix consists of adding (or XORing) one state word into the next, and rotating the first word.

Multiple rounds are applied in a cyclical pattern to the state. The horizontal lines of the rounds wind themselves in helical fashion through the five state words. Twenty rounds make up one block. Helix actually uses two interleaved helices; a single block contains two full turns of each of the helices. The critical path through the block function consists of six modulo 2^{32} additions and five XORs. In Figure 4, the

half of the block of the Helix cipher is illustrated. The other half of the block is the same as the part shown in Figure 4.

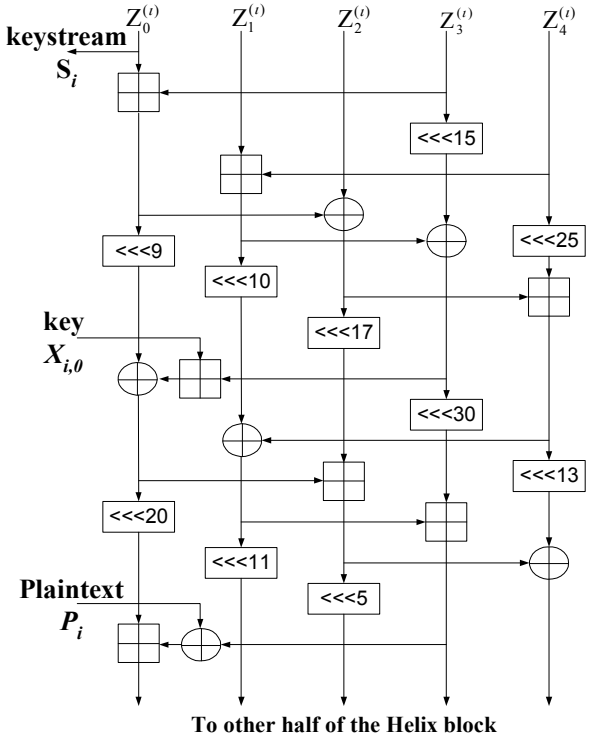

Figure 4. The half of the block of the Helix cipher

6. RC4 Cipher

RC4 is a variable key-size stream cipher developed by Ron Rivest for RSA Data Security, Incorporation. The RC4 stream cipher has two phases, the *key set-up* and the *keystream generation*. Both phases must be performed for every new key. During an *n*-bit key set-up, the encryption key is used to generate an encrypting variable using two arrays - the state and the key array - and *n*-number of mixing operations [2].

RC4 works in Output Feedback (OFB) mode [2] of operation. In RC4 there are two 256-byte arrays, the State (S)-box and the Key (K)-box. The S-box is linearly filled, such as $S_0 = 0$, $S_1 = 1$, $S_2 = 2$, ..., $S_{255} = 255$. The K-box consists of the key repeated as many times in order to fill the array.

RC4 cipher uses two counters, *i* and *j*, which are initialized to zero. In the key set-up phase, the S-box is being modified according to the following pseudo-code:

Key set-up phase:

for i = 0 to 255
$j = (j + S_i + K_i) \bmod 256$
swap S_i *and* S_j

Once the key set-up phase is completed, the second phase encrypts or decrypts a message. The keystream generation phase is described by the following pseudo code:

Keystream generation phase:

$i = (i + 1) \bmod 256$
$j = (j + S_i) \bmod 256$
swap S_i *and* S_j
$t = (S_i + S_j) \bmod 256$
$K = S_t$

For producing the ciphertext/plaintext, the generated keystream is XORed with the plaintext/ciphertext.

Figure 5 shows the block diagram of the aforementioned RC4 phases.

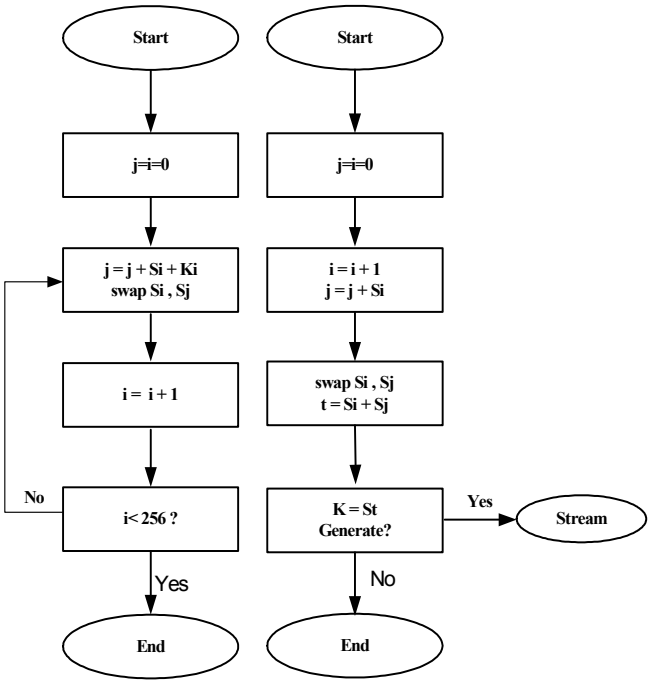

Figure 5. Block diagram of two RC4 phases.

7. Implementation Issues

The hardware implementations of A5/1, W7, Helix and E0 stream ciphers are quite straightforward, since their hardware architectures are well defined in this paper. For the implementation of the E0 cipher, the implementation of [7] is adopted.

For the RC4 cipher, an efficient implementation which is parameterized in order to support variable key lengths, is proposed. The key length could be 8 up to 128-bit, opposed to the previous designs [6, 8] that support only fixed key lengths.

The proposed architecture of the RC4 stream cipher consists of a control and a storage unit and it is shown in Figure 6. The storage unit is responsible for the key set-up and keystream generation phases. The operation of the storage unit is synchronized by the control unit. The control unit generates the appropriate clock and control signals.

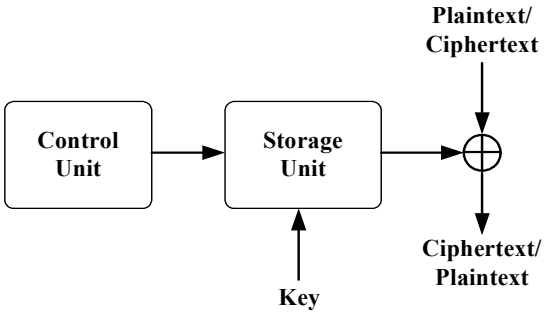

Figure 6. General architecture for the RC4 cipher.

The implementation of the storage unit is shown in Figure 7. The storage unit contains memory elements for the S-box and K-box, along with 8-bit registers, adders and one multiplexer.

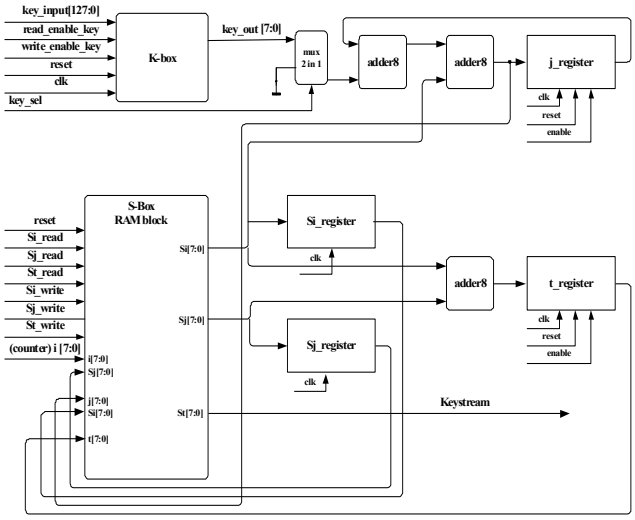

Figure 7. Storage unit implementation for the RC4 stream cipher.

The block diagram of the S-box RAM is shown Figure 8. It consists of three 256 bytes RAM blocks. Each RAM block has four inputs and one output. The two inputs are the read and write signals, while the other two ones are the address and data signals. Also, all the three RAM boxes have the same signals for clock and reset.

The operation of the RAM blocks is quite simple. If the reset signal is activated, the blocks are linearly initialized. For each block, if the write signal is activated, new data are stored in the address position. On the other hand, if the read signal is activated, the data in the address position are available on the output of the block. The two first blocks *i*, *j* of Figure 7 are used for the swapping of the values of the third block *t*. The final values (i. e., the keystream) that are used for the algorithm are produced by the *t* block.

The key set-up is divided in two steps. In the first step, the S-box is filled. The S-box is linearly initialized, such as $S_0 = 0$, $S_1 = 1$, $S_2 = 2$, …, $S_{255} = 255$ when the reset state occurs.

In the second step of the key set-up, the S-box is randomly filled. For the S-box, a 3x256-bytes RAM memory is used as it is shown in Figure 8. The *Si_* and *Sj_* registers in Figure 7 are used for the swappings imposed by the algorithm. The *j_* and *t_* registers in Figure 7 are used in order to temporarily store all the intermediate variables that are produced.

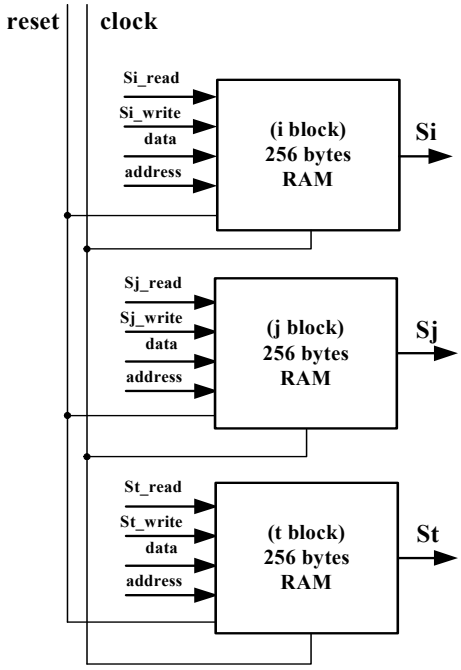

Figure 8. The S-box RAM of the RC4 cipher.

At the *first* clock cycle, the value of counter *i* (Figure 7) is used as address in the first RAM block. The value of *Si* (stored in the *Si_register*) is used for the computation of the new value of *j* as it is shown in Figure 7. The two adders are used for the computation of the new value of *j*. They accept as input the values of *Ki* and *Si*. At the *second* clock cycle, the new produced value *j* is used as an address for the second RAM block. The stored value in this address is temporarily stored in the *Sj_register*. At the *third* cycle, the contents of the *Si_register* and *Sj_register* are written at the *j* and *i* addresses, respectively. With this procedure, the swapping is achieved.

The first phase needs three clock cycles per iteration. So, the total time that is required in the key set-up phase is 256.3 = 768 clock cycles.

The second phase (keystream generation) is quite similar to the first one. So, the same hardware is being re-used. The difference in this phase is that the values of the K-box are not used. After the completion of the first phase, the multiplexer in Figure 7 selects the zero value input. Also, the *j_register* is initialized to zero so as to be ready for the second phase. After the two aforementioned actions, the procedure of keystream generation can begin.

The operations at the first three steps are similar to those of the key set-up phase except that the S-box is already initialized. At the *first* step, the value of *i* is used as address in the first RAM block and the value of *Si* is stored in the *Si_register*. Also, the new value of *j* is computed. At the *second* step, the new value of *j* is

used as address of the second RAM block and the value of *Sj* is stored in the *Sj_register*. In this step, the values of *Si* and *Sj* are being added and the result of the addition is stored in the *St_register*. At the *third* step, the contents of the *Si_register* and *Sj_register* are written at the *j* and *i* addresses, respectively, and the value of the *t_register* is being used as address for the third RAM block. So, the value of *St* is also produced in the third step. This value of *St* is the generated keystream byte.

After the completion of this phase, each byte in the keystream can be generated and used for encryption/decryption. The encryption/decryption is achieved by the bitwise XORing of the keystream with the plaintext/ciphertext.

The time needed for the keystream generation phase is 3.*n* cycles, where *n* is the number of bytes of the plaintext or ciphertext. So, the total time for both RC4 phases is 768 + 3.*n* clock cycles.

8. Implementation Results

The results of performance (in terms of throughput) and of consumed area (in terms of FPGA CLB slices), for the implemented stream ciphers, are presented in Table 1. All the designs were synthesized in a Xilinx Virtex-II™ 2V250FG256 FPGA [11], for having a common hardware device for the comparison. The selected FPGA has 18K-bit selectRAM™ blocks. Each block is synchronous and it can be easily configured in 256-byte RAM blocks. The proposed RC4 implementation utilizes a 3.256-byte RAM block.

Table 1. Performance and area comparison.

Cipher	Area (slices)	Frequency (MHz)	Throughput (Mbps)	Throughput / Area
A5/1	32	188.3	188.3	5.88
W7	608	96.0	768.0	1.26
E0	895	189.0	189.0	0.21
Helix	418	32.0	1024.0	2.45
RC4	140	60.8	120.8	0.86

As illustrated in Table 1, the Helix stream cipher achieves the largest throughput that it is measured in Mega bits per second (Mbps). Also, it has the second best throughput-to-area ratio. This ratio is a measure of the hardware performance of the ciphers. The A5/1 cipher has the best throughput-to-area ratio. This is a rather expected outcome since A5/1 has a quite simple architecture that consumes the least FPGA area.

The results for the throughput-to-area ratio for all ciphers are graphically shown in Figure 9. The E0 cipher has the smallest ratio, while the A5/1 has the largest one. So, A5/1 achieves the best hardware performance. The throughput of W7 implementation is much better compared with the one that the A5/1 implementation achieves. However, this comes with an area cost.

The time required for the key set-up phases (initialization phases) of the presented stream ciphers are: 12.6 μs, 1.26 μs, 0.99 μs, 0.25 μs and 0.01 μs for the RC4, E0, A5/1, Helix and W7, respectively. So, RC4 has the largest start-up time and W7 the smallest one. The respective clock cycles for the key set-up phases are: 768, 239, 188, 8 and 1 cycles for RC4, E0, A5/1, Helix and W7, respectively.

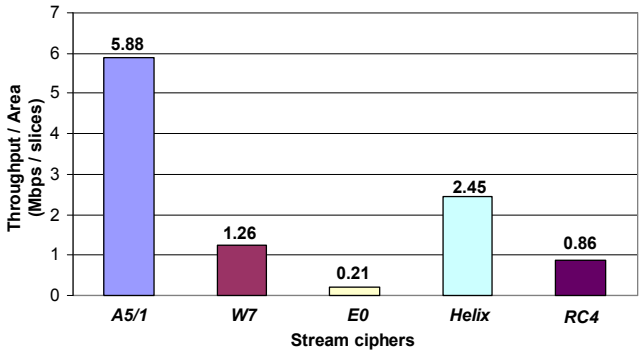

Figure 9. Throughput to area ratio results.

To the best of our knowledge there are no published hardware implementations results for the Helix, A5/1 and W7 ciphers, which can be compared with our respective implementations. The implementation results for RC4 are comparable with the ones in [6]. Our implementation is faster and consumes less area, since in [6] the area was 255 CLB slices and the throughput was 17.76 Mbps. So, our design outperforms their RC4 implementation.

9. Conclusions

In this paper, five representative stream ciphers are implemented in hardware and compared in terms of performance and consumed FPGA area. These ciphers were coded in VHDL language and synthesized in an FPGA device. The largest throughput-to-area ratio has been achieved by the A5/1 cipher and is equal to 5.88 Mbps/slice. The Helix cipher achieves the largest throughput (1024 Mbps). The throughput of the hardware implementation of the Helix cipher proves that this cipher is indeed fast, as it was shown in the comparison of its software implementation with other ciphers [5]. The W7 has the smallest key set-up time (0.01 μs). Also, the performance of the W7 cipher is greater than the one of the A5/1. Finally, our developed RC4 architecture outperforms previous published designs both in terms of performance and of consumed area.

Acknowledgements

Michalis D. Galanis would like to thank the Alexander S. Onassis Public Benefit Foundation for financially supporting his PhD thesis.

References

[1] Bluetooth SIG, "Specification of the Bluetooth System," vol. 1.1, February 2001, http://www.bluetooth.org/spec/, 2004.

[2] Dworkin M., *Recommendation for Block Cipher Modes of Operation. Methods and Techniques*, *National Institute of Standards and Technology (NIST)*, Technology Administration, U.S. Department of Commerce, Special Publication, http://csrc.nist.gov/publications/nistp ubs/800-38a /sp800-38a.pdf, 2004.

[3] Ekdahl P. and Johansson T., "Another attack on A5/1" *IEEE Transactions on Information Theory*, vol. 49, no. 1, pp. 284-289, January 2003.

[4] European Telecommunications Standards Institute (ETSI), "Recommendation GSM 02.09," Security Aspects.

[5] Ferguson N., Whiting D., Schneier B., Kelsey J., Lucks S., and Kohno T., "Helix: Fast Encryption and Authentication in a Single Cryptographic Primitive," *Lecture Notes in Computer Science* (LNCS), Springer-Verlag, Berlin, Germany, vol. 2887, pp. 330-346, 2003.

[6] Hamalainen P., Hännikäinen M., Hamalainen T., and Saar J., "Hardware Implementation of the Improved WEP and RC4 Encryption Algorithms for Wireless Terminals," *in Proceedings European Signal Processing Conference (EUSIPCO)*, Tampere, Finland, pp. 2289-2292, September 2000.

[7] Kitsos P., Sklavos N., Papadomanolakis K., and Koufopavlou O., "Hardware Implementation of Bluetooth Security," *IEEE Pervasive Computing*, vol. 2, no.1, pp. 21-29, January-March 2003.

[8] Kundarewich P. D., Wilton S. J. E., and Hu A. J, "A CPLD-Based RC4 Cracking System," *in Proceedings of the Canadian Conference on Electrical and Computer Engineering*, May 1999.

[9] Thomas S., Anthony D., Berson T., and Gong G., "The W7 Stream Cipher Algorithm," *Internet Draft*, April 2002.

[10] Weatherspoon S., "Overview of IEEE 802.11b Security", *Intel Technology Journal Q2*, Network Communications Group, Intel Corporation 2000.

[11] *Xilinx Inc*., San Jose, California, Virtex-II 2.5V FPGAs, http://www.xilinx.com, 2004.

Michalis Galanis received his BSc in physics and MSc in electronics from the Department of Physics, University of Patras in 2000 and 2002, respectively. Since October 2002, he has been working towards his PhD degree in the domain of reconfigurable computing at the VLSI Design Laboratory of the Department of Electrical and Computer Engineering. In 2003, he received a scholarship for his PhD studies from the Alexander S. Onassis Public Benefit Foundation for his excellent academic studies in past years. He has authored or co-authored 18 research papers, presented (or to appear) in international conferences and journals.

Paris Kitsos obtained a BSc degree in physics from University of Patras, Greece, in 1999. In March 2004, he received his PhD degree from the Department of Electrical and Computer Engineering of the University of Patras. His research interests include efficient implementation of public-key algorithms, symmetric algorithms, security protocols for wireless systems, efficient polynomial basis Galois field arithmetic, VLSI design, and computer arithmetic. He has authored or co-authored 27 research articles.

Giorgos Kostopoulos received his Diploma in electrical & computer engineering from the Electrical & Computer Engineering Department, University of Patras, Greece, in 2003. Since then, he has been working towards his PhD degree in the Department of Electrical and Computer Engineering of the University of Patras. He has been a member the Technical Chamber of Greece since October of 2003. He has authored or co-authored 5 research papers, presented (or to be appeared) in international conferences.

Nicolas Sklavos received the Diploma in electrical & computer engineering and the PhD degree in electrical & computer engineering in 2000 and 2004, respectively, both from the Electrical & Computer Engineering Deptartment, University of Patras, Greece. His research interests include cryptography, wireless communications security, VLSI design, and reconfigurable computing architectures. He holds an award for his PhD thesis on "VLSI Designs of Wireless Communications Security Systems", from IFIP VLSI SOC 2003. Dr. Sklavos is a member of the IEEE, the Technical Chamber of Greece, and the Greek Electrical Engineering Society. He has authored or co-authored more than 60 scientific articles, book chapters, tutorials and reports, in the areas of his research.

Costas Goutis was a research assistant and research fellow in the Department of Electrical and Electronic Engineering, University of Strathclyde, Strathclyde, UK, from 1976 to 1979, and lecturer in the Department of Electrical and Electronic Engineering, University of Newcastle upon Tyne, UK, from 1979 to 1985. Since 1985, he has been an associate professor and full professor in the Department of Electrical and Computer Engineering, University of Patras, Patras, Greece. His recent research interests focus on VLSI circuit design, low-power VLSI design, systems design, analysis and design of systems for signal processing, telecommunications, memory management, and reconfigurable computing. He has been awarded a large number of research contracts from ESPRIT, RACE, and National Programs. Professor Goutis has authored or co-authored more than 200 scientific articles, book chapters, tutorials and reports, in the areas of his research.