International Journal of Computer and Electrical Engineering, Vol. 1, No. 4, October, 2009
1793-8163

Real-time FPGA-based Privacy and Authenticated Surveillance System

M.M. Abutaleb, A.M. Allam, A. Hamdy, M.E. Abuelwafa, and E.M. Saad

***Abstract*— The detection of moving objects in a frame sequence is an essential processing component for video surveillance. The main objective of this paper is to create a prototype system that can produce secure streams, ciphertext, for every pixel in the detected moving blobs, plaintext, at a much higher rate to be applicable for real time applications. This approach can be applied in security applications such as anti-theft systems and forensic investigation systems. It could potentially be of benefit to financial investment companies, the military and security forces in order to keep certain information encrypted that no one who does not know exactly how to decrypt will not be able to obtain this information. FPGAs have been used to process larger images at faster speed because of their configuration flexibility and high data processing speed. In this paper, FPGA-based architectures for the proposed privacy and authenticated surveillance system are presented. This design can process 768x576 frame sequence at a very high bit rate that reaches to 40M bit per second in a single FPGA chip, which is adequate for most real-time vision applications.**

***Key words*— Motion detection, cryptography, authentication, FPGA.**

I. INTRODUCTION

This work presents a robust method to secure streams of moving objects appearing in images of the frame sequence that may be sent in open communication channel. A motion detection technique is applied to detect the moving pixels in the image sequence taken from a stationary camera. Then, a cryptographic protocol and an authentication technique are applied only w.r.t. the detected moving pixels. This method is suitable for FPGA implementation that can be used in applications which require real-time.

A. Motion Detection

There are three conventional approaches to motion detection: temporal differencing [1]; optical flow analysis [2]; and background subtraction [3-5]. Temporal differencing is suitable in dynamic environments, but generally does a poor job of extracting all relevant feature-pixels. Optical flow can be used to detect independently moving objects in the presence of camera motion; however, most flow-computation methods are complex and inapplicable in real-time. Motion detection by background subtraction can be divided into adaptive and non-adaptive background methods. Non-adaptive methods need off-line initialization; errors in the background accumulate over time.

A common method of adaptive backgrounding is to average the frames over time [6]. This creates an approximate background. This is effective where objects move continuously and the background is visible for a significant portion of time. It is not robust for scenes with slowly-moving objects. It cannot handle a multimodal backgrounds caused by the repetitive motion of the background. Hence, in this work a fast motion detection algorithm based on a multi-modal distribution, modeling each pixel as a mixture of normal distributions, is used to detect the moving objects with a small number of calculations, as presented in [7], to achieve a high frame rate for real-time requirements. This method deals robustly with slowly-moving objects as well as with repetitive background motions of some scene-elements.

B. Security

An automatic surveillance system sends important information about the movement of detected object. Is this information not important? Is this information allowed to be readable for anybody? Is it true that the received information from this system? The answer of all this questions is no, so we use the cryptography to give our system the privacy of information the user need, and use an authentication technique to be sure about the source of the information.

Diffie-Hellman key exchange [8] was proposed in 1976 during collaboration between Whitfield Diffie and Martin Hellman and was the first practical method for establishing a shared secret over an unprotected communications channel. In 2002 [9], Hellman suggested an algorithm called Diffie-Hellman-Merkle key exchange in recognition of Ralph Merkle's contribution to the invention of public-key cryptography. Encryption techniques are used to prevent any unauthorized individuals from reading or altering data transmitted. In this work, an improved stream cipher technique based on Diffie-Hellman protocol is used; a stream cipher is a symmetric key cipher where plaintext bits are combined with a pseudorandom cipher bit stream (key stream). In a stream cipher the plaintext digits are encrypted one at a time, and the transformation of successive digits varies during the encryption.

Although Diffie-Hellman key agreement is a non-authenticated key-agreement protocol, it provides the basis for a variety of authenticated protocols. Challenge-response authentication is a protocol in which one party presents a question ("challenge") and another party

M.M. Abutaleb is with the Department of Electronics, Communications, and Computer,Faculty of Engineering, Helwan University, Cairo, Egypt
A.M. Allam, is with the Department of Electronics, Communications, and Computer,Faculty of Engineering, Helwan University, Cairo, Egypt
A. Hamdy is with the Department of Electronics, Communications, and Computer,Faculty of Engineering, Helwan University, Cairo, Egypt
E.M. Saad is with the Department of Electronics, Communications, and Computer,Faculty of Engineering, Helwan University, Cairo, Egypt
M.E. Abuelwafa is with the Department of Electronics, Communications, and Computer,Faculty of Engineering, Helwan University, Cairo, Egypt

must provide a valid answer ("response") to be authenticated. In Kerberos authentication protocol [10], the challenge is an encrypted randomly-generated integer, while the response is the encrypted integer plus one, proving that the other end was able to decrypt the integer. This protocol is used in the proposed design to reduce the hardware and speed up the system.

C. Hardware Implementation

High complexity algorithm that produces the secure streams of moving objects from image sequences has been incorporated into today's video surveillance systems, due to computational cost and lack of real-time capability. This makes the development of such algorithms on the hardware timely. Standard algorithms have been restricted to small frame sizes, low frame rates, and solely implemented in software running on general-purpose computers. Real-time needs of such systems can be improved by a significant amount with the use of hardware.

Hence, this paper presents a hardware implementation of a proposed method, which takes advantage of data parallelism for a further implementation on a field programmable gate array (FPGA), a re-configurable computing platform. Throughput has been drastically increased while; in contrast, the latency has been decreased, with the use of pipelining. This implementation is used to produce the secure streams of moving objects appearing in image sequence at a high bit rate in a single FPGA chip.

This paper is organized as follows. In section 2 and 3, the algorithms are formulated and the suggestions are introduced. In section 4, the hardware implementation design and performance analysis of the proposed system are discussed. Conclusion is given in section 5.

II. Motion Detection based on Multi-modal Distribution

A fast and efficient algorithm, presented in [7], is used here to extract the moving objects in each frame for software and hardware implementation. In this algorithm, each pixel is modeled as mixture of three distributions (k = 3) and each distribution is represented by mean-value (μ) and weight-value (ω) maintained at time t. The mixture is sorted every time in decreasing order of weight values. Here, each pixel is checked against the distributions, until the match is found. The matching condition is achieved if the variation of the pixel Xt is within R% (matching ratio) from its mean value.

For the matched distribution, the current pixel is stored as the processed pixel value XP,t (XP,t = Xt) to be used in the next time t+1. Also, the temporal differencing is applied where the pixel is considered as foreground pixel if:

$$| X_t - X_{P,t-1} | > T \qquad (1)$$

where T is the threshold value. If foreground pixel is detected, the weight of the matched distribution will be updated as in equation (2) and its mean will be kept without any change. While if foreground pixel is not detected, the weight and mean of the matched distribution will be updated as in equation (3) and (4) respectively, where α is the learning factor.

$$\omega_{k,t} = (1 - a)\, \omega_{k,t-1} \qquad (2) \qquad \omega_{k,t}$$

$$= (1 - a)\, \omega_{k,t-1} + a \qquad (3)$$

$$\mu_{k,t} = (1 - a)\, \mu_{k,t-1} + a\, X_t \qquad (4)$$

If the current pixel is not matched with any distribution, this pixel will be classified as foreground pixel. The mean of the third distribution of that pixel will be replaced by its intensity value and its weight will be selected as lower value than other distributions. Also, the processed pixel will be equal to the value of the mean of the first distribution. Real-time performance with high resolution video streams can be achieved by this algorithm.

Figure 1 shows the frame under study. The next step is the motion detection by our proposed method in [7], which is performed to extract the foreground pixels; moving pixels. Figure 2 demonstrates the extraction of the foreground pixels. Subsequently, the post-processing is performed to fill the holes inside the blobs, as it is demonstrated in Fig. 3. Figure 4 shows the intensity information of the detected moving objects.

Fig.1 Single image from the sequence

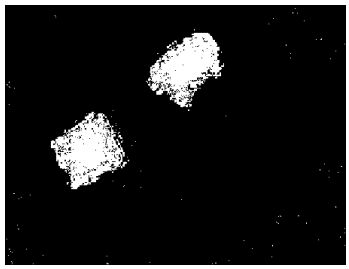

Fig.2 Pre-processing foreground image

Fig.3 Post-processing foreground image

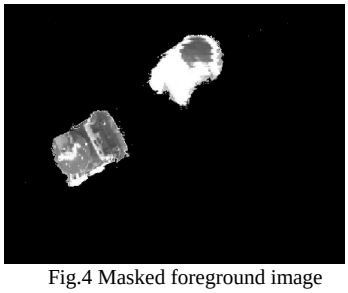

Fig.4 Masked foreground image

III. Privacy and Authentication

A. Encryption/Decryption Algorithm

The simplest, and original, implementation of the Diffie-Hellman key exchange protocol [8] uses the Multiplicative group G of integers modulo n, where n is prime and g is primitive root mod n. Here is a general description of the protocol:

1. Source (eg. A) and destination (eg. B) agree on global parameters:
   - large prime number *n*.
   - finite cyclic group *G* and a generating element *g* in *G*.
2. Source A:
   - chooses a secret key *Xa*.
   - sends a public key $Ya = g^{Xa} \bmod n$ to destination B.
3. Destination B:
   - chooses a secret key *Xb*.
   - sends a public key $Yb = g^{Xb} \bmod n$ to source A.
4. Source A: computes a shared secret key $K = (g^{Xb} \bmod n)^{Xa} \bmod n$.
5. Destination B: computes a shared secret key $K = (g^{Xa} \bmod n)^{Xb} \bmod n$.

Both A and B have arrived at the same value, because gXaXb and gXbXa are equal mod n. Note that only a, b and gXaXb = gXbXa mod p are kept secret. All the other values, n, g, gXa mod n, and gXb mod n, are sent in the clear. Once A and B compute the shared secret they can use it as an encryption key, known only to them, for sending messages across the same open communications channel. Of course, much larger values of Xa, Xb, and n would be needed to make it secure. If n were a prime of at least 160 bits, then even the best algorithms known today may not find Xa given only g, n, and gXa mod n, even using all of mankind's computing power. The problem is known as the discrete logarithm problem.

The Diffie-Hellman problem (DHP) is stated informally as follows: Given an element g and the values of gx and gy, what is the value of gxy? Formally, g is a generator of some group and x and y are randomly chosen integers. In the Diffie-Hellman key exchange, an eavesdropper observes gx and gy exchanged as part of the protocol, and the two parties both compute the shared key gxy. In cryptography, for certain groups, it is assumed that the DHP is hard, and this is often called the Diffie-Hellman assumption. The problem has survived scrutiny for a few decades and no "easy" solution has yet been publicized [11].

In a synchronous stream cipher, a stream of pseudo-random digits is generated independently of the plaintext and ciphertext messages, and then combined with the plaintext (to encrypt) or the ciphertext (to decrypt). In the most common form, binary digits are used (bits), and the keystream is combined with the plaintext using the exclusive-or operation (XOR). In a synchronous stream cipher, the sender and receiver must be exactly in step for decryption to be successful. Stream ciphers can be viewed as approximating the action of a proven unbreakable cipher, the one-time pad (OTP). A one-time pad uses a keystream of completely random digits. Substituting pseudorandom data generated by a cryptographically secure pseudo-random number generator is a common and effective construction for a stream cipher.

B. Shared Key Generator

The encryption/decryption algorithm requires computation of the modular exponentiation to obtain shared secret key. Modular exponentiation is a type of exponentiation performed over a modulus. Doing a "modular exponentiation" means calculating the remainder when dividing by a positive integer m (called the modulus) a positive integer b (called the base) raised to the e-th power (e is called the exponent). In other words, it is needed to calculate c such that:

$$C = b^e \bmod n \tag{5}$$

In the domain of hardware implementation, an intelligent algorithm is needed in order to reach a higher efficiency. Hence, exponentiation is achieved by performing a number of squaring and multiplications. This mathematical operation has involved a few, modular operations; modular multiplication, modular addition, and subtraction operations on large integers [12].

Given the integers b, e, and n, the e has to be changed to binary in order to start the algorithm to compute be. There are two variations which depend on the direction by which the bits of e (e contains h-bits) are scanned: (LR) and Right-to-Left (RL). The LR binary method is more widely known as follows [12]:

Left-to-Right Binary Method
Input: b; e; n
Output: $C := b^e \bmod n$
1. **if** $e_{h-1} = 1$ **then** $C := b$ **else** $C := 1$
2. **for** $i = h - 2$ **downto** 0
2a. $C := C \cdot C \bmod n$
2b. **if** $e_i = 1$ **then** $C := C \cdot b \bmod n$
3. **return** *C*

A modular multiplication problem is defined as the computation of P = A x B mod n. The modulus multiplication operation is needed after the separation of exponentiation into a number of squaring and multiplication. Separating the multiplication operation into a number of modular addition operations will be used. The method of computing P is as follows [12]:

Modular Multiplication Operation
Input: A; B; n
Output: $P := A \times B \bmod n$
1. $P := 0,\ B' := 0$
2. **while** $B' < B$ **loop**
2a. $B' := B' + 1$
2b. $P := (P + A) \bmod n$
3. **return** *P*

There are basically four general approaches for computing the product P [12-15]: Multiply and then divide, Interleaving multiplication and reduction, Brickell's method and Montgomery's method. All approaches above have a common disadvantage that it doubles up the number of bits for each multiplication and hence a large register is needed to store this result.

In this design, each modular multiplication is realized by a

series of additions and subtractions to overcome this problem. The modular addition problem is defined as the computation of S = (P + A) mod n. The proper method of computing S is as follows:

Modular Addition Operation
Input: A; P; n
Output: S := (*P* + *A*) mod *n*
1. *S* := *P* + *A*
2. **while** *S* ≥ *n* **loop**
2a. *S* := *S* - *n*
3. **return** *S*

Note that modular addition involves subtraction operation in step 2a. Based on the algorithms described above, a proposed substantial-speed and memory-efficient method is developed as follows:

Complete Proposed Method
Input: b; e; n
Output: C := b^e mod n
1. **if** e_{h-1} = 1 **then** *C* := *b* **else** *C* := 1
2. **for** *i* = *h* - 2 **downto** 0
2a. *C* := (*C* mod *n*)^2 mod *n*
2b. **if** e_i = 1 **then** *C* := ((*C* mod *n*) . *b*) mod *n*
3. **return** *C*

where each multiplication is realized by a series of additions, shift arithmetic, and the modulus (*mod n*) is realized as a number of subtractions as discussed previously.

C. Secret Key Generator

Linear feedback shift registers (LFSRs) have long been used as pseudo-random number generators. An LFSR is a shift register whose input bit is a linear function of its previous state. The only linear functions of single bits are xor and inverse- XOR; thus it is a shift register whose input bit is driven by the exclusive-or (XOR) of some bits of the overall shift register value. The bit positions that affect the next state are called the taps. In the diagram shown in Fig.5, the taps are (16,14,13,11,1) , the feedback polynomial is x16 + x14 + x13 + x11 + 1. An h-bit LFSR (exponent contains h-bits) is used in the proposed design based on the standard LFSR in [16] to generate the secret number in source (Xa) and destination (Xb) each time randomly.

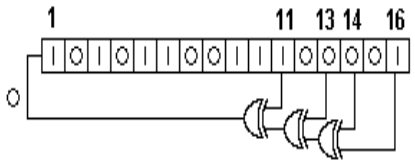

Fig.5 A 16-bit LFSR

D. Challenge-Response Authentication

In Kerberos protocol [10] as shown in Fig.6, the challenge is an encrypted integer m, while the response is the encrypted integer m + 1, proving that the other end was able to decrypt the integer m. An LFSR is used also here to generate integer m each time randomly.

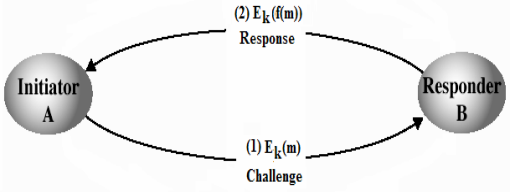

Fig.6 Challenge-response authentication

IV. Hardware Descriptions

A. Data Flow

The data flow of the proposed algorithm is divided into three modules as shown in Fig.6. In the Motion_Detection module, images of the frame sequence are read from memory into the FPGA and aligned to extract the binary masks of foreground images. This module is previously implemented in [7]. Then in the Intensity_and_Location module, each image in the frame sequence is masked by the binary masks to generate serial streams of the intensity component for each moving pixel with its index. Finally, these components are fed into the Privacy_and_Authentication to process the input/output encrypted streams according to the privacy and authentication protocols which are discussed previously. Of importance, the computation process can be fully pipelined and partially paralleled in hardware to improve its processing throughput and thereby enable real-time use.

In the block diagram of Fig. 7, the connections between modules consist of only unidirectional data and a corresponding data valid signal. Once a set of data is generated in a module, it is registered into the downstream module for processing. At the end of the pipeline, the serial streams are transmitted or received according to the usage communication protocol.

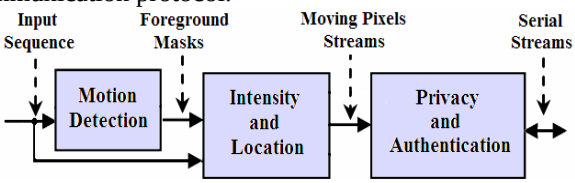

Fig.7 Data flow of the design

B. Architectures

For Intensity_and_Location module, it consists of Pixel_Ext block and Int_Loc_Ser block, as shown in Fig. 8. A control signal rd_in is active when mask signal (foreground pixel) from Motion Detection module is ready in the input. Every clock clk signal, Pixel _Ext block generates J signal as the intensity component F of the current processing pixel concatenated with its corresponding location idx if and only if mask signal equal one. This operation is used to speed-up the system where the data of moving pixels accumulated with their locations are much smaller than the whole frame data to be encrypted/ decrypted. Also it achieves real-time requirement where it is able to send/receive 25-30 fps (frames per second), video rate. A control signal ld is used to load the concatenated data of the detected pixel to Int_Loc_Ser block which outputs it on Ps as serial streams according to synchronous line Cs and enable line En of the whole system controls.

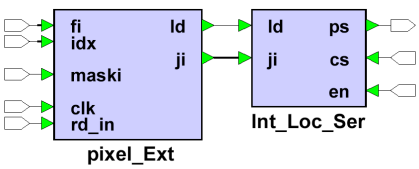

Fig.8 The structure of Intensity_and_Location module

For Privacy_and_Authentication module, it represents the security part in the proposed system which is designed in the form of pipeline and parallel processing as shown in Fig.9 to achieve maximum speed up. A control signal rst initializes

the parameters of the system. Fr_En is active when the new frame is ready to process as a new cycle. Signals g and n represent selective prime and primitive root that are required to compute the shared secret key.

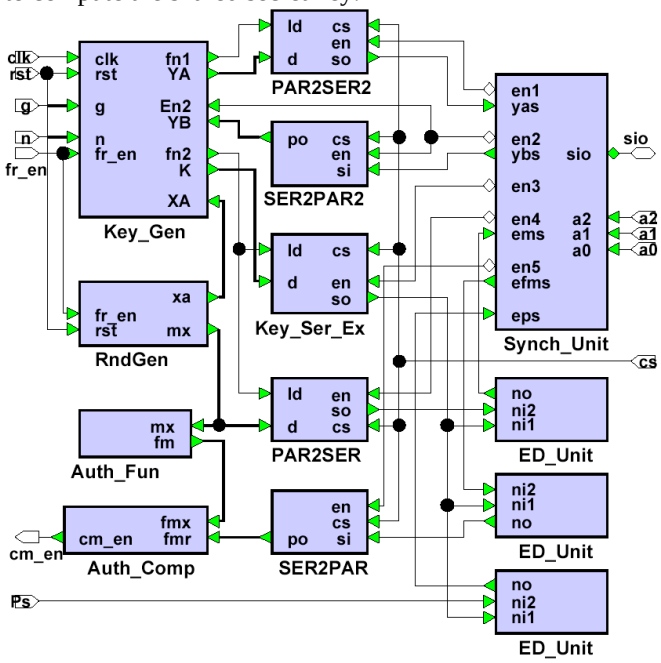

Fig.9 The structure of Privacy_and_Authentication module

Key_Gen block is used to compute the shared secret key based on Diffie-Hellman key exchange protocol which is discussed in section 3.1. It consists of two Mod_Unit blocks, as shown in Fig. 10. Mod_Unit block is used to compute Co3, the modular exponentiation C= be mod n, based on the proposed algorithm which is discussed in section 3.2. First Mod_Unit block is used to compute Ya = gXa mod n, public key of source A, then activates the finish control signal Fn1. The second Mod_Unit block is used to compute K = YbXa mod n, the shared secret key of source A and destination B, when enable line En2 is active then activates the finish control signal Fn2. Note that, Xa and Ya in the source A are only replaced by Xb and Yb in the destination B to compute the shared secret key.

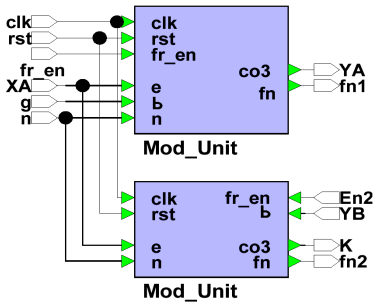

Fig.10 Key_Gen block

Rnd_Gen block is pseudo-random number generator which is discussed in section 3.2. Each frame, it is used to generate a secret key Xa in the source (or Xb in the destination) and randomly-generated information mx, the challenge of the entity authentication, only in the source. Auth_Fun block is used to compute a predetermined function fmx of the originally-offered information in the source (or the received information in the destination). Auth_Comp block is used to compare fmx with the received decrypted information fmr, the response of the entity authentication, in only in the source. A signal Cm_En represents the decision of the authentication process to start the encryption/decryption algorithm of plaintext (moving pixels streams) in ED_Unit block. PAR2SER and SER2PAR blocks are used to convert parallel to serial and serial to parallel data inside the system.

Based on three address lines A0,A1,A2 and synchronous line Cs (bit rate), Synch_Unit block is used: (1) to control the operation of Privacy_and_Authentication module with the whole system, (2) to connect the in/out serial stream of signal sio to one of five serial signals Yas (Ya as serial), Ybs (Yb as serial), Ems (encryption of mx as serial), Efms (encryption of fmr as serial), and Eps (encryption of Ps, moving pixels streams), and (3) to activate PAR2SER, SER2PAR, and Key_Ser_Ex blocks according to the operation by using five enable lines En1, En2, En3, En4, and En5. Key_Ser_Ex block is used to generate shared secret key streams and change it by doing XOR for each two successive bits in key after the end of the current key. This operation is used to reduce the system overhead and make it one-time pad through current frame but to keep the security level, the key is recomputed each frame completely using Key_Gen block.

C. *Performance*

For enhanced hardware performance, several hardware optimizations were made:

- *Pipeline structure*. A heavily pipelined hardware structure was used to maximize throughput. Once the pipeline is full, the hardware can produce a result on every clock cycle.
- *Optimizations of Motion_Detection module.* Memory bandwidth reduction is achieved by utilizing distribution similarities in succeeding neighboring pixels and data flow reduction is also achieved by processing only one distribution at time through the hardware, as presented in [17].
- *Fast memory access.* Due to the pipelined hardware architecture used, the major system bottleneck turned out to be memory access. A high speed multi-port memory controller is used to increase the data throughput of the design.
- *Efficient Modular Exponentiation*. Modular exponentiation is performed as a number of additions and subtractions inside the loops and uses the same register size to store the result of each modular multiplication.

D. *Simulation and Synthesis*

A simulation is performed to test the logic function of the hardware design and it is presented to verify the correctness of the algorithms implemented by the proposed modules. A simulation of the Privacy_and_Authentication module is performed with 25 ns simulation clock period (40 MHz). Assumed that n = 253, g = 3, Xa = 97. The public key of source A is computed as Ya = 397 mod 253 = 40. Assumed that Yb = 248, the received public key of destination B. Finally, the shared secret key of source A and destination B is computed as K = 24897 mod 253 = 160. The simulation result of the key generator (Key_Gen) process is shown in Fig. 11.

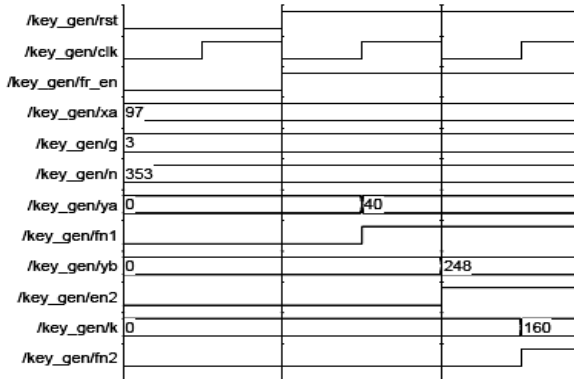

Fig.11 Simulation waveforms of the key generator

Each new frame, Rnd_Gen block is used to generate a secret key and authentication information mx randomly. The simulation result of the random generator (Rnd_Gen) process is shown in Fig. 12.

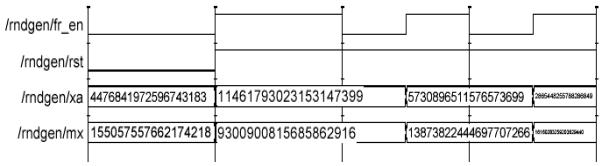

Fig.12 Simulation waveforms of the random generator

Through current frame, Key_Ser_Ex block is used to change a shared secret key and output the current key as serial stream. The simulation result of the key serial exchange (Key_Ser_Ex) process is shown in Fig. 13.

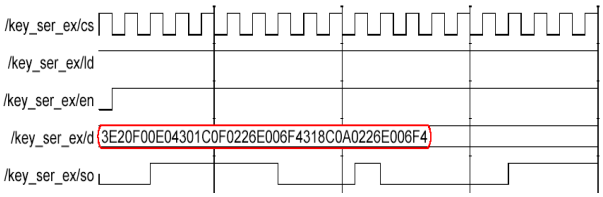

Fig.13 Simulation waveforms of the key serial exchanger

Each moving pixel is represented in our system by 28-bits which accumulate intensity and location bits with a single flag bit. This flag bit, the last left bit, is constant by 0 or 1 through the current frame and must be inverted with a new frame in the source to be used as indicator for each new frame received in the destination. A key size of 168-bits is used to encrypt/decrypt 28 moving pixel each time. The simulation result of the encryption and encryption process (ED_Unit) for test frame serially with a key stream is shown in Fig. 14 and Fig. 15. Fig.14 depicts part of the encryption process, while Fig. 15 shows the corresponding decryption process. All simulation results are identical for the corresponding theoretical results.

In regard to the designated hardware realization, The VHDL code is synthesized by considering VERTEX-II Xilinx chip 2V2000fg676 which has 10752 CLB slices and 22872 D-flip flops or Latches. The VERTEX-II family provides the density, speed, and features to integrate entire systems, including multiple buses into a single chip. Throughout the synthesis results, there are a few points worth to be discussed. Motion_Detection module as presented in [17] uses 418 CLB slices with 1.94% utilization and the clock frequency report showed the critical frequency is 80.7MHz.

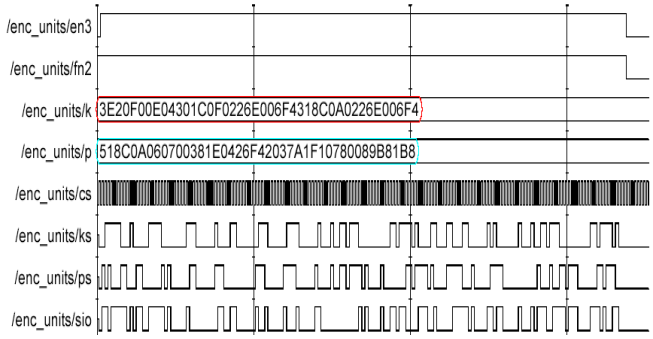

Fig.14 Simulation waveforms for part of the encryption process

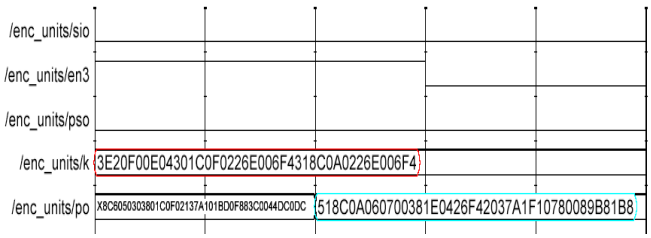

Fig.15 Simulation waveforms for part of the decryption process

Intensity_and_Location module uses 29 CLB slices with 0.27% utilization and the clock frequency report showed the critical frequency is 269.4MHz. While *Privacy_and_Authentication* module uses 5120 CLB slices with 47.62% utilization and 6844 DFFs with 29.92% utilization. The clock frequency report showed the critical frequency is 42.6MHz for key generator. This frequency represents the degree of the system overhead but not limits the speed of the whole system where it is used here only before encryption/decryption process. However, the critical frequency can possibly be increased further by optimizing the circuit through place & route the internal probes. The serial to parallel and parallel to serial converters could achieve 406.8MHz and 207.2MHz respectively. The frequency achieved of the whole system is 40MHz which can encrypt/decrypt the image sequence of size 768x576 with bit rate of 40Mbs (bits/Second) that achieves desired real time rate, 25-30 fps.

V. Conclusion

In the proposed surveillance system, a fast and efficient motion detection algorithm has been used to extract the moving objects in each image of frame sequence then an improved stream cipher and authentication techniques have been applied only w.r.t. the detected moving pixels. So, secure streams of detected moving objects in the frame sequence can be transmitted or received across the open communications channel. Real-time performance and efficient hardware are achieved by this method.

A hardware implementation for the proposed method has been presented in the form of pipeline and parallel processing to achieve maximum speeding up. Several hardware optimizations were also made to enhance hardware performance. This design could encrypt/decrypt the streams of moving objects appearing in 768x576 image sequence taken from a stationary camera at a very high bit rate of 40Mbs in a single FPGA chip and achieve real time requirement.

REFERENCES

[1] C. Anderson, P. Burt, and G. van der Wal, "Change detection and tracking using pyramid transformation techniques", in Proceedings of SPIE. Intelligent Robots and Computer Vision, vol. 579, pp. 72-78, 1985.

[2] J. Barron, D. Fleet, and S. Beauchemin, "Performance of optical flow techniques", International Journal of Computer Vision, vol. 12, pp. 42-77, 1994.

[3] A. Kasinski and A. Hamdy, "Efficient Separation of mobile objects on the scene from the sequence taken with an overhead camera". Proc. Int. Conf. on Computer Vision and Graphics, Zakopane, vol. 1, pp. 425-430, Sept. 2002.

[4] C. Ridder, O. Munkelt, and H. Kirchner, "Adaptive Background Estimation and Foreground Detection Using Kalman-Filtering". Proc. Int. l Conf. Recent Advances in Mechatronics, ICRAM .95, pp. 193-199, 1995.

[5] Y. Ivanov, A. Bobick, and J. Liu, "Fast Lighting Independent Background Subtraction". Technical Report no. 437, MIT Media Laboratory, 1997.

[6] G. Halevy and D. Weinshall, "Motion of disturbances: detection and tracking of multi-body non-rigid motion", Machine Vision and Applications, vol. 11, Issue 3, pp. 122-137, 1999.

[7] E.M. Saad, A. Hamdy, and M.M. Abutaleb, "FPGA-based Implementation of a Low Cost and Area Real-time Motion Detection", 15th IEEE Conference in Mixed Design of Integrated Circuits and Systems, Poznan, Poland, pp. 249-254, June 19-21, 2008.

[8] W. Diffie and M. E. Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976, pp: 644-654.

[9] Martin E. Hellman, "An Overview of Public Key Cryptography", IEEE Communications Magazine, pp: 42-49, May 2002.

[10] B. Clifford Neuman and Theodore Ts'o, "Kerberos: An Authentication Service for Computer Networks", IEEE Communications, 32(9) pp33–38. Sept. 1994.

[11] D. Cash, E. Kiltz, V. Shoup "The Twin Diffie-Hellman Problem and Applications." Advances in Cryptology – EUROCRYPT 2008, pp: 127-145, April 2008.

[12] C. K. Koc., "RSA Hardware Implementation. Technical Report TR 801", RSA Laboratories, pp. 1-24, 1996.

[13] M.K. Hani, T.S. Lin, N. Shaikh-Husin, "FPGA Implementation of RSA Public-Key Cryptographic Coprocessor", in Proceedings of TENCON, vol. 3, pp. 6-11, Kuala Lumpur, Malaysia, 2000.

[14] Y.S. Kim, W.S. Kang, J.R. Choi, "Implementation of 1024-bit Modular Processor for RSA Cryptosystem", in Proceedings of Asia-Pasific Conference on ASIC, pp. 187-190, Cheju Island, Korea, 2000.

[15] M. Shand and J. Vuillemin, "Fast Implementation of RSA Cryptography", in Proceedings of 11th IEEE Symposium on Computer Arithmetic, pp. 252-259, Windsor, Ontario, 1993.

[16] Maria George and Peter Alfke, "Linear Feedback Shift Registers in Virtex Devices", XAPP210 (v1.3), April 2007, http://www.xilinx.com/bvdocs/appnotes/xapp210.pdf

[17] M.M. Abutaleb, A. Hamdy, and E.M. Saad, "FPGA-Based Real-Time Video-Object Segmentation with Optimization Schemes", International Journal of Circuits, Systems, and Signal Processing, vol. 2, issue 2, pp. 78-86, 2008.