Real Time Video Encryption Final Project Report

Wesley Folz, Radik Nasibulin
Department of Electrical and Computer Engineering, University of Arizona, Tucson AZ
wesfolz@email.arizona.edu, radikraisovich@email.arizona.edu

Abstract – The growth of the low powered and low cost embedded systems brought a large number of surveillance systems with real time remote monitoring to the market. A large portion of these systems lack proper security measures and have exposed user's data. In this paper we investigate possible implementations of secure video capturing and transmission between two systems on a network. The goal is to find an optimal implementation to transmit high quality secured video in real time. Our simulated system is tested with encryption of 1080p video at 30fps in real time using AES and Elliptic Curve encryptions.

I. Introduction

Real-time video encryption can be useful in a portable surveillance system, where data must be streamed securely to remote viewing locations. A specific example of this could be unmanned aerial surveillance drones which need to send high quality encrypted video frames to some sort of command center. To be practical, these surveillance drones must operate on low power while still meeting the performance constraints of real-time video transmission and encryption.

Many approaches to achieve real-time video encryption focus on developing clever algorithms to efficiently encrypt image data. Our approach differs from this in that we are looking at well-established encryption algorithms such as Advanced Encryption Standard (AES) and Elliptic Curve Encryption (ECC) that are known to be cryptographically secure to do the real-time video encryption. We investigate the performance of these algorithms by simulating realistic embedded systems using SystemC and examine the performance increases in offloading the cryptography to dedicated hardware accelerators.

This paper will explore the computational complexity of full data encryption of a video stream. ECC and AES encryption techniques will be explored in software and hardware co-processors to meat real time deadlines. We also attempt to optimize the power consumption of our embedded system.

II. Related Work

There have been proposed a number of approaches to secure real time video streams. In majority of cases proposed designs scrambling data or partially encrypt part of the data. It is generally believed that full video encryption is too computationally intensive to perform in real time. Most of the proposed designs have been tested using software implementation of the encryption, and therefore are too slow.

Work done by Joshi and Dalal have shown some promises of the real time application. They were able to show real time performance using purely software encryption. However, their design is limited to very low resolution for modern applications, 240x320 at 30 FPS [7].

Bergeron, among others, has proposed partial encryption of a stream. In this scenario the structure of the stream is kept intact including the header information. This means that the stream is playable, but it is modified to a pint where it will be impossible to reconstruct. This is done to decrease the amount of computation to be performed [8].

Research done in [5] demonstrates that an FPGA based AES encryption coprocessor is capable of handling the performance constraints of real-time video encryption. Their design of a 128-bit AES encryption with 10 rounds achieved a throughput of 4.28 Gbits/sec and consumed only 4.96mW of power, making it suitable for low-power embedded systems.

III. System Level Overview

A. Dataflow

To understand the system, consider first the application. The system is a low powered embedded system that captures images from a camera with 1920x1080 resolution, at 30 frames/second. Every captured image is stored in a camera buffer. These images are then encoded to MPEG format and then processed by a cryptographic coprocessor that encrypts the data and stores it in a memory block. The network interface can then send the encrypted data to another embedded system via WiFi Direct, where the data can be decrypted. Each video frame is put through the dataflow model shown in Figure 1.

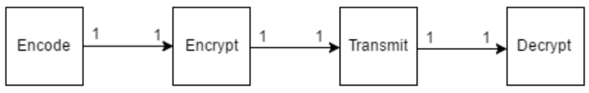

Fig 1. Dataflow Model

B. *Pipeline Processing*

Our system employs a pipelining scheme to process each video frame that comes in from the camera buffer. Our pipeline has four stages: encode, encrypt, transmit and decrypt. Given a frame rate of 30 frames/sec, each stage in the pipeline has 33ms to complete its task. Once a task finishes an operation the result must be saved until the following task is executed, after this the result of the task can be overwritten. For example, in stage 1 of the pipeline, the encoder encodes frame 1 and stores it in memory, in stage 2 the encoder encodes frame 2 while the cryptographic coprocessor encrypts frame 1. In stage 3 the encrypted frame 1 is being transmitted across the network, while the encoded frame 2 is being encrypted and frame 3 is being encoded, this means that the encoded frame 1 is no longer needed and can be overwritten by the encoded frame 3. To implement this, each component in the pipeline has two dedicated blocks in memory that they alternate reading from and writing to.

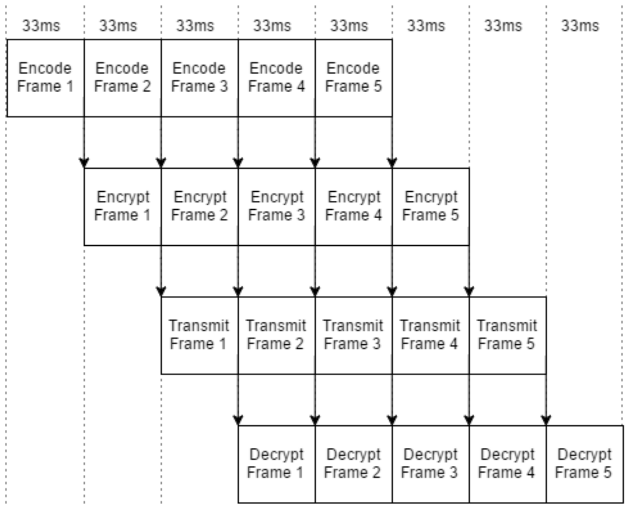

Fig 2. Video Frame Pipeline

C. *System Architecture*

As shown in Figure 2, the main embedded system consists of a memory block slave, a microprocessor master and a crypto block, MPEG encoder and network interface that all act as a slave and master. All the system blocks are connected using AMBA ARM bus. The new data arrives into the system from the Camera Buffer which sends it directly to the encoder. The camera data is then encoded by the MPEG encoder, processed by the crypto block which encrypts and stores the data in the Memory block. The encrypted data is then sent to the other embedded system via WiFi Direct. The second embedded system consists of a network interface to receive encrypted camera data, a crypto block to decrypt data, a microprocessor and a memory block to store the decrypted data.

1) *MPEG Encoder:* The MPEG encoder block is modelled as a Broadcom VideoCore IV GPU. The GPU can acquire a video stream directly from a camera buffer without interrupting the performance of other components. It is modeled as a data source that generates data at a rate of 30 frames per second. The encoder was tested for the data generation rate. It was setup to encode video at 1280x720 at 30 frames per second.

2) *Cryptographic Coprocessor:* Our system assumes that the cryptographic key exchange was performed via Diffie-Hellman before any camera data arrives. For this reason, we don't simulate the key exchange as it acts as initial overhead and does not impact the ability of our system to perform real-time video encryption. This means that at the start of the simulation, both embedded systems already possess the necessary encryption and decryption keys.

The system was initially tested and benchmarked using a software only implementation for the cryptographic computations. Both AES and ECC were implemented in software and ran on the microprocessor. After this, the software implementation was replaced with a hardware accelerator that performs the cryptographic processing. The hardware implementation works as follows: First the CPU makes a master bus request sending the source address of the data to be encrypted or decrypted, the destination address in memory where the encrypted or decrypted data is written, a flag telling the coprocessor to either encrypt or decrypt the data and the length of the data to be encrypted or decrypted. At this point the cryptographic coprocessor makes a master bus request asking to read the data to be encrypted or decrypted directly from the main memory. The data is then encrypted or decrypted using either AES or ECC. Once the cryptographic computation is done, the coprocessor makes another master bus request and write the resulting data back to memory.

a) *AES:* The AES cryptographic coprocessor used in this design is a GRLIB IP core which uses a 128-bit encryption key with 10 number of rounds. When synthesized on a Xilinx Vertex-2 XC2V6000-4 device, the core has a maximum frequency of 125MHz [3].

b) *ECC:* The ECC cryptographic coprocessor used in this design is a GRLIB IP core which implements a 233-bit elliptic curve to generate an encryption key. When synthesized on a Xilinx Vertex-2 XC2V6000-4 device, the core has a maximum frequency of 93MHz [3].

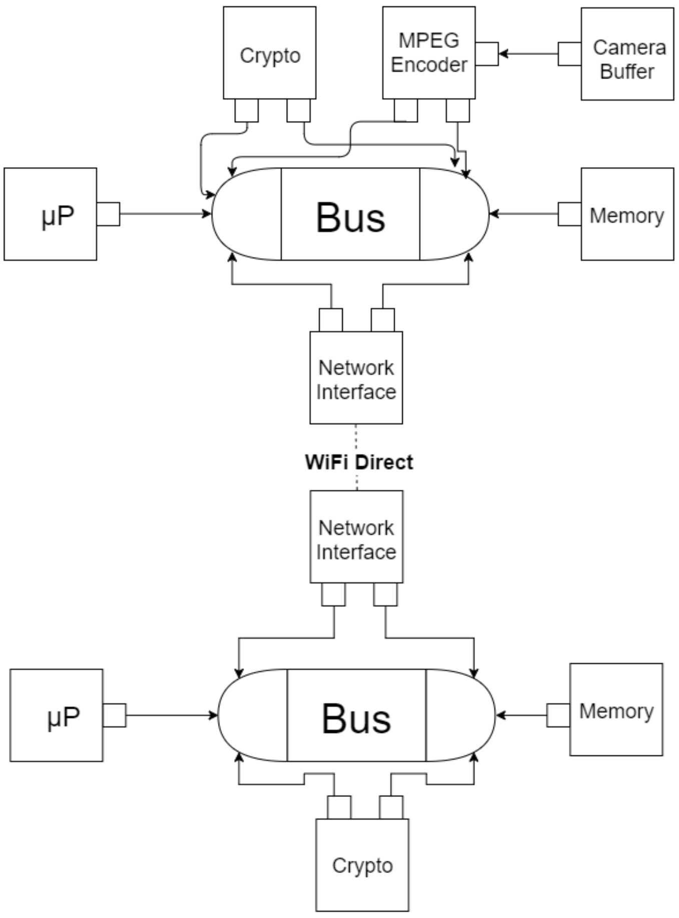

Fig. 3. System Level Architecture

3) Network Interface Card: The network interface card (NIC) transmits and receives data similarly to the description provided in [1]. First the CPU makes a master bus request sending a buffer descriptor containing the memory address and size of the Ethernet packet to be transferred. The buffer descriptor is written to the NIC's local memory array. At this point the NIC makes a master bus request asking to read the Ethernet packet data directly from the main memory. Once the NIC reads the data from memory, it sends it across the network via WiFi Direct.

When the NIC receives a packet from the network, it stores it in its local receive buffer. It then makes a bus request to transfer the packet data to main memory. After the packet data is transferred to memory, the NIC triggers a global interrupt notifying the CPU that a new packet was received.

a) WiFi Direct: Our system assumes that the WiFi direct connection has already been established, so we don't simulate the P2P group formation. Instead we just use throughput, transmit and receive power specifications of the Atheros AR4100(P) chipset found in [4] to determine the total delay and power required to transfer data across the network.

The specification of each component in our system is summarized in Table I.

TABLE I. COMPONENT SPECIFICATIONS

Component	Frequency (MHz)	Throughput (Mbits/s)	Power (mW)
GRAES	125	159	36
GRECC	93	2.4	160
Mali 200 GPU	275	6600	490
ARM11 CPU	700		160
Atheros AR4100(P) NIC		54	Tx: 498 Rx: 153

IV. RESULTS

A. High Resolution Dynamic Video

To evaluate the performance of all the proposed systems, 4 models were build. Two models utilized FPGA based encryptions, the AES and ECC. Two additional models used software implementation of the encryptors. To find a design with optimal performance all the systems are initially set to full performance. The AES hardware is set to 125MHz, and ECC to 93MHz.

The test is performed on a sample video with resolution of 1920x1080. First five frames of the video is used for the simulation. The first 2 frames are I and P frames respectively. These frames are the largest in size, where P frame occurs more frequently and on average have a size of 116kBytes. The size of the P frames do not exceed 290kBytes. On the other hand the last three I frames take significantly less space and occur more often. These five frames are a good estimate of the overall encryption load of the system. Compare that to the average bitrate from [6]. Generally an expected bitrate for high quality video of 1920x1080 resolution is 5.76Mbits/s. Making it an average per frame (30 FPS), 5.76x0.033 = 0.190Mbits/frame or 23kBytes/frame.

The results of the simulations are summarized in Tables II and III. The timing results include bus read/writes and task executions. The energy consumption results summarizes the amount of energy consumed by a component during the whole execution.

The design with AES coprocessor has achieved encryption and decryption rates that are well within the constrain of 33ms. The encryption and decryption time for the largest frame, P, is 19 ms. The bottleneck of this design is the wireless transmission, where it could not transmit the P frame within 33ms. It caused the decryptor to miss 2 following frames. The energy consumed by encryption and decryption is much less than the energy consumed by the encoder and the network hardware.

The other systems were unable to encrypt or decrypt the first I frame before the deadline. Table III show the execution time to compress the first frame I.

TABLE II. EXPERIMENTAL RESULTS OF THE SYSTEM WITH AES CO-PROCESSOR

		Encode	Encrypt	Transmit	Decrypt
Time ms	frame 1 126 (kB)	9.01	8.38	18.83	8.38
	frame 2 289 (kB)	10.30	19.05	42.84	19.06
	frame 3 49 (kB)	8.39	3.27	18.83	0.00
	frame 4 11 (kB)	8.09	0.73	1.96	0.00
	frame 5 1.1 (kB)	8.08	0.67	1.50	0.67
Energy (mJ)		39.60	0.89	30.58	0.80

TABLE III. EXPERIMENTAL RESULTS OF THE SYSTEMS WITH AES IN SOFTWARE, AND ECC IN SOFTWARE AND A CO-PROCESSOR

	Encryption		Decryption	
	Time (ms)	Energy (mJ)	Time (ms)	Energy (mJ)
AES µP	98.14	15.58	98.15	15.58
ECC HW	423.88	67.45	423.92	67.45
ECC µP	66.22	10.47	24.35	3.77

B. *High Resolution Static Video*

To further investigate the possibility of the proposed systems another video sample was used. This video sample consists of an 8-frame scene with very little motion. The resolution of the video is reduced to 1920 x 720. The total size of the video is 18 kBytes with the last frame P taking up about 12 kBytes.

Tables IV and V show the results of the components when they were processing the last frame. Only the design with ECC co-processor failed to process this frame in time. It missed the deadline by 9.7ms. All other design showed good performance with some room to relax the encyption and decryption time. Incorporating the AES coprocessor would make the best choice as it has the lowest computational overhead, and consumes the least energy.

TABLE IV. EXPERIMENTAL TIMING (MS) RESULTS OF ALL FOUR SYSTEMS

	Encoder	Encrypt	Transmit	Decrypt
AES HW	3.58	0.84	1.90	0.85
AES µP	3.58	9.90	1.84	9.91
ECC HW	3.58	42.71	1.92	42.75
ECC µP	3.58	6.67	1.83	2.45

TABLE V. EXPERIMENTAL ENERGY (MJ) RESULTS OF ALL FOUR SYSTEMS

	Encoder	Encrypt	Transmit	Decrypt
AES HW	51.480	0.035	1.380	0.035
AES µP	51.480	2.360	1.380	2.380
ECC HW	51.480	10.180	1.380	10.220
ECC µP	51.480	1.580	1.370	0.570

C. *Design Space Exploration*

The AES co-processor model can further be enhanced for energy consumption by frequency scaling. Table VI summarizes the performance of the system for both video files after frequency scaling for their respective longest frames. The frequency of the AES can be scaled down to as low as 72MHz to save energy for the video of 1920x1080. This only saves about 0.09 mJ for encryption and 0.04 mJ for decryption. The amount of energy saved will grow in a real application when hundreds of frames are encrypted.

The frequency of the AES co-processor can be scaled as low as 4 MHz for the 1280x720 video. However there is no energy saving in this model.

	Encryption		Decryption	
	Time ms	Energy mJ	Time ms	Energy mJ
1920x1080	33.0	0.80	33.0	0.76
1280x720	26	0.035	26	0.035

V. CONCLUSION

Utilizing hardware/software co-design methods and transaction-level modelling in SystemC, we were able to simulate our embedded system design and evaluate its capabilities of encrypting and transmitting video in real-time. After optimizing the power consumption of our system we found that it was capable of encrypting 1080p video at 30fps in real time using a 128-bit AES coprocessor operating at 72MHz. The system had a total power consumption of 281mW. Unfortunately, our WiFi direct bandwidth of 54Mbps was too slow to handle some of the larger frames in 1080p video. Utilizing a faster network bandwidth or improved encoding scheme should correct this problem. To process 720p video the AES coprocessor could be scaled all the way down to 4MHz and consumed 174mW**.**

Future work that would improve our design could explore a larger design space exploration and look to further optimize each component in our system. More efficient encoding and encrypting schemes could be looked at and a network interface card with a higher bandwidth could be also added to the system. Additionally, different encryption parameters could be varied such as the number of rounds or they key sizes to compare encryption strength with performance and power consumption.

References

[1] P. Willmann, Hyong-youb Kim, S. Rixner and V. S. Pai, "An efficient programmable 10 gigabit Ethernet network interface card," *11th International Symposium on High-Performance Computer Architecture*, 2005, pp. 96-107.

[2] D. Camps-Mur, A. Garcia-Saavedra and P. Serrano, "Device-to-device communications with Wi-Fi Direct: overview and experimentation," in *IEEE Wireless Communications*, vol. 20, no. 3, pp. 96-104, June 2013.

[3] Cobham Gaisler, "GRLIB IP core user’s manual," version 1.5.0 datasheet, pp. 374-375, 438, Jan 2016.

[4] Qualcomm Atheros, "AR4100(P) system in package 802.11n: general availability," 80-Y2310-1 Rev. B datasheet, March 2013.

[5] S. KOTEL, M. ZEGHID, A. BAGANNE, T. SAIDANI, Y. I. DARADKEH, and T. RACHED. FPGA-Based Real-Time Implementation of AES Algorithm for Video Encryption . Recent Advances in Telecommunications, Informatics and Educational Technologies, pages 27–36, 2014.

[6] "Video Encoding Settings for H.264 Excellence", *Lighterra.com*, 2016.[Online].Available:http://www.lighterra.com/papers/videoencodingh264/. [Accessed: 01- Apr- 2016].

[7] J. Joshi and U. Dalal, "Enhancing Selective ISMACryp Video Encryption for Real Time Applications in Handheld Devices", *Advanced Materials Research*, pp. 403-408, 2011.

[8] B. Boyadjis, C. Bergeron, B. Pesquet-Popescu and F. Dufaux, "Extended Selective Encryption of H.264/AVC (CABAC) and HEVC encoded video streams", *IEEE Trans. Circuits Syst. Video Technol.*, pp. 1-1, 2016.

[9] L. Levision, https://github.com/Yalir/Synthese4A/cryptron, May 2013.

[10] "kokke/tiny-AES128-C", *GitHub*, 2016. [Online]. Available: https://github.com/kokke/tiny-AES128-C. [Accessed: 05- May- 2016].