Journal of Computing and Information Technology - CIT 23, 2015, 4, 329–340
doi:10.2498/cit.1002764

Hardware Accelerator Approach Towards Efficient Biometric Cryptosystems for Network Security

Charles McGuffey, Chen Liu and Stephanie Schuckers

Clarkson University, Potsdam, New York, USA

Protecting data and its communication is a critical part of the modern network. The science of protecting data, known as cryptography, uses secret keys to encrypt data in a format that is not easily decipherable. However, most commonly secure logons for a workstation connected to a network use passwords to perform user authentication. These passwords are a weak link in the security chain, and are a common point of attack on cryptography schemes. One alternative to password usage for network security is to use a person's physical characteristics to verify who the person is and unlock the data correspondingly. This study focuses on the Cambridge biometric cryptosystem, a system for performing user authentication based on a user's iris data. The implementation of this system expanded from a single-core software-only system to a collaborative system consisting of a single core and a hardware accelerator. The experiment takes place on a Xilinx Zynq-7000 All Programmable SoC. Software implementation is performed on one of the embedded ARM A9 cores while hardware implementation makes use of the programmable logic. Our hardware acceleration produced a speedup of 2.2X while reducing energy usage to 47.5 % of its original value for the combined enrolment and verification process. These results are also compared to a many-core acceleration of the same system, providing an analysis of different acceleration methods.

Keywords: hardware acceleration, biometric cryptosystem, iris recognition, Reed-Solomon code, Hadamard code

1. Introduction

In the modern world, data is becoming more prolific, and much of it is digitally accessible [1]. This widespread availability of data over the network leads to sensitive information becoming vulnerable, which means that it must be protected. One way for this protection to happen is through cryptography, the science of encoding or decoding information. Cryptography works by taking data to be protected, known as a "secret", and performing a transformation on that data to make it appear as though the data is a random sequence of bits or characters. This transformation is known as "encryption". Intended users of the data will be able to make sense of the random sequence, or to "decrypt" the data, through a methodology known only to them. Other parties that do not have access to the decryption instructions will not be able to make sense out of the data. If done correctly, converting the random sequence of bits back to the real meaningful data with no knowledge of the decryption key should be computationally complex, meaning that it cannot be achieved with current computing technology within a reasonable amount of time.

Most commonly, secure workstation logons for a workstation connected to a network make use of passwords for verifying user authenticity [15]. However, passwords are not ideal in that they are prone to being lost, stolen, or forgotten. To overcome the weakness of traditional password-based authentication, many research groups have investigated the use of biometrics, which are unique physical or behavioral characteristics of people, to protect cryptographic keys. Several researchers have developed algorithms using biometric measurement for cryptography systems, which can greatly improve the security of network systems. These implementations, called biometric cryptosystems, combine biometrics and cryptography in a manner that allows accurate matching of biometrics in an encrypted, secure domain [2].

In order to take advantage of biometric cryptosystem design, the algorithm needs to be able to run at a speed that is effective for use in commercial products. For most applications, this means calculations and cryptography must occur in real-time. This study attempts to achieve that goal by performing hardware acceleration of a biometric cryptosystem algorithm.

Hardware acceleration allows a biometric cryptosystem algorithm to be executed more quickly by providing a hardware platform that can handle the computation more efficiently. For each accelerated function, software code is replaced with a hardware component dedicated to perform the assigned computations, allowing increased parallelism and improving the speed of the function relative to running on a general-purpose processor. This performance increase often significantly outweighs the extra power consumed by the additional hardware component(s), reducing the total energy consumed by the system for the given task.

The performance improvement presented in this study provides a template for making biometric cryptosystems more viable in an industrial setting, such as commercial network security, where they can provide secure access to data without the use of passwords. In addition to the practical benefits of cryptography, this study provides insights on the strengths and weaknesses of the hardware acceleration process. Analysis of what target function characteristics lead to the greatest efficiency gain provides a framework for determining the most efficient use of hardware acceleration. This will also allow more effective usage of hardware acceleration, including applications beyond the scope of biometric cryptosystems.

The rest of the paper is organized as follows: a brief overview of biometric cryptosystems is provided in Section II; Section III discusses the reference biometric cryptosystem design in detail; hardware acceleration methods and benefits are discussed in Section IV; Section V provides details on the experimental setup and procedure; results and associated discussion are provided in Section VI; final conclusions are drawn in Section VII.

2. Biometric Cryptosystems

Traditional cryptography methods make use of password protection and cryptographic keys to guard valuable information from attackers. The key is used to encrypt or decrypt data as necessary and is usually long and difficult to guess or memorize. This has led to keys being released based on password authentication, which allows users to memorize a relatively short password while providing data encryption that is still strong enough to resist attackers [3]. However, passwords themselves are a relatively vulnerable medium. Passwords are often poorly chosen, allowing them to be compromised through intelligent guessing or brute force attacks [3]. Additionally, passwords are often recorded on unsecure mediums, such as unencrypted files or pieces of paper that are left lying around. If the password is not recorded, it must be memorized, causing the existence of password recovery services, which take additional time and provide additional potential for security flaws. These weaknesses have caused a significant amount of research into alternatives to passwords.

One alternative method to the usage of passwords is the biometric cryptosystem. A biometric cryptosystem takes data about a particular feature of each user, called a “biometric”, and records that data. Biometrics are defined as behavioral or physiological characteristics of a user. Examples of biometrics include fingerprints, iris scans, facial patterns, hand geometry, signatures, and keystrokes, etc. Biometric systems enroll users by storing information about the users’ biometrics. If a user attempts to gain access to the system under protection, that user’s biometric is checked. If the biometric data provided by the user matches the corresponding entry in the system and has the appropriate privileges, the user is granted access. If the biometric data does not match, or matches an entry without the required privileges, then the user is denied access to the system. This eliminates the need of passwords and the security risks associated with their usage.

There are several issues associated with the use of biometrics prior to their application to cryptography. In order for a biometric to be useful, it must be able to be collected and accurately compared to other data collections. The difficulty of biometric collection is usually due to the requirement of a specific hardware setup, and is therefore irrelevant to our discussion. Accurate comparison, however, is a relevant issue. When biometric collection occurs, there is a high variance in the data collected due to different positions of the subject in the collection, variance in the machine, noise, and other issues. Additionally, biometric data can change due to human growth, injury, habit changes, and other real-world events [2]. A biometric system must be able to handle these data variances in order to be effective. Complicating matters, the analysis system must also be able to distinguish between the biometric data of different people. Biometric systems must seek to minimize the false acceptance rate (FAR), where two different people are identified as the same person, and the false rejection rate (FRR), where one person is not identified as himself/herself on separate data collections [3]. This dual constraint poses a significant challenge to biometric systems, even prior to their application in cryptography.

There are also problems that arise specifically when biometrics are applied to cryptography. The first of these issues is privacy. If biometric templates are stored unencrypted in the system, attackers could potentially steal the biometric data of users from the system [4]. This has privacy implications that cause the system losing its functionality from both ethical and economic standpoints. In addition to these issues, the scenario of a specific account being compromised must be taken into consideration as well. It is important that this scenario would not result in any other system that uses the same biometric being compromised. Revocability, or the ability to generate multiple secure identities for a user, is a requirement used to prevent a security breach from permanently compromising a user's access to a system. This prevents the use of raw biometric data, and instead necessitates using a transformation that is not based exclusively on the biometric data. However, working with biometric template post-transformation provides significant challenges in coming up with alignment processes necessary to deal with the variances and tolerances that must be accounted for [5], as discussed above.

A large amount of research has been performed attempting to solve these problems [5]. One of the promising methods is an algorithm using iris biometrics through error correcting code proposed by Anderson et al. from a research group in Cambridge University [6]. This algorithm is referred to herein as the "Cambridge biometric cryptosystem", which we will discuss in detail next.

3. The Cambridge Biometric Cryptosystem

The Cambridge biometric cryptosystem is an algorithm that confirms user authenticity through the use of an iris template. Each user in the system is enrolled by providing a 256-byte iris template and receiving a randomly generated 140-bit key. These two inputs are used to generate two variables that are stored on a physical token that the user receives. The first of the generated variables is a hash of the original 140-bit key. This hash function is a mapping function used to obscure the original 140-bit key. The second variable, called a locked template, is the result of performing an exclusive-or (XOR) function between the enrollment template and the result of putting the randomly generated key through Reed-Solomon and Hadamard encoding sequentially [6, 7]. When a user attempts to gain access to the system, the user provides an iris sample and the physical token. The locked template is XORed with the user's sample template, producing the encoded key with errors introduced by the differences between the enrollment template and the sample template. This result is then put through Hadamard decoding, followed by Reed-Solomon decoding. If the person attempting to access the system is a valid user with the correct token, the result of the decoding will be the original key. If someone is trying to access the system using someone else's token, the result will be different. The decoded key is hashed after calculation using the same hash method as in the enrollment process, and then compared to the hashed result stored on the token. If they are the same, the user is deemed valid and granted access. If the results are different, then the user is treated as an imposter and is not given access to the system. A block diagram of this cryptosystem is shown in Figure 1.

The Cambridge biometric cryptosystem makes use of Reed-Solomon error-correction code to handle burst errors in the iris template. These are large errors that affect many contiguous bits of an iris template. Burst errors are often caused by eyelashes blocking views of the iris or other devices that cause significant error in the picture. Reed-Solomon code handles these errors by dividing the input into several blocks. The blocks are interpreted as coefficients to a polynomial using the domain of a Galois field. Additional parity blocks are added to create a resulting polynomial that is evenly divisible by the defining polynomial of the Galois field. During the decoding process, the division is performed on the input blocks, and the remainder is used to locate blocks that are in error. These error blocks are then entirely recalculated based on the results. This means the number of errors in a particular block does not matter, only the number of blocks that contain errors. For the Cambridge biometric cryptosystem, the 140-bit input is divided into 20 blocks of 7 bits, and 12 parity blocks are added to create a result of 32 blocks of 7 bits. This allows the system to correct up to 6 block errors for any given encoding and decoding process.

Hadamard coding is used by the Cambridge biometric cryptosystem to handle random errors in the iris template. These are small errors that may be caused by transmission errors, or may be related to minor changes in the image of the iris taken. Hadamard coding is based on the concept of the Hadamard matrix, a matrix containing positive and negative values with a magnitude of one, where the values are arranged in a particular order. The initial Hadamard matrix, of order one, is shown in Equation 1.

$$H_1 = \begin{bmatrix} + & + \\ + & - \end{bmatrix} \qquad (1)$$

Subsequent Hadamard matrices are computed as shown in Equation 2.

$$H_k = \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix} \qquad (2)$$

A Hadamard code uses a matrix that differs from a Hadamard matrix in two ways. The first is that the additive inverse of the Hadamard matrix is stacked below the original matrix as shown in Equation 3. The second is that all negative ones in the Hadamard matrix are replaced by zeros.

$$H_c = \begin{pmatrix} H \\ -H \end{pmatrix} \qquad (3)$$

Hadamard encoding interprets its input as a series of blocks. Each block is treated as a row index. The output result of the block is the entirety of the row corresponding to that index. During the decoding process, each block is compared to every row of the Hadamard matrix, and the row index corresponding to the row with the least bit differences from the input block is the output. This allows the system to return the correct output if the number of bit errors is less than half the minimum distance between rows of the matrix. For the Cambridge biometric cryptosystem, the input consists of 32 7-bit blocks. A block size of seven means that there are 128 rows in the matrix. Since the matrix used is constructed from two square matrices stacked on top of each other, each row consists of 64 bits. Thus, the resulting output is 32 64-bit blocks. This choice allows each block to contain up to 15-bit errors before becoming a burst error.

The Cambridge biometric cryptosystem appears to successfully fulfill the requirements of a strong and safe biometric cryptosystem. According to the tests performed by Anderson et al., the system performed with a 99.5 % correct

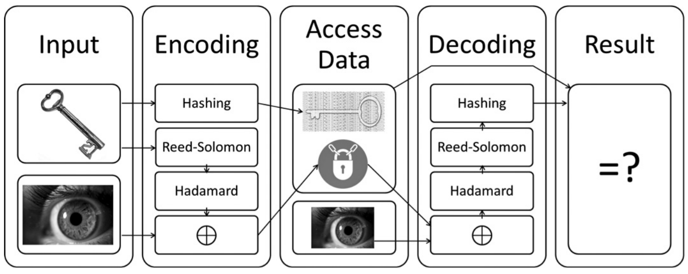

Figure 1. The Cambridge Biometric Cryptosystem. The Process of enrollment includes all items in the encoding block. The verification process encompasses the steps illustrated in the decoding block.

match rate [6]. The algorithm answers security concerns by storing only modified versions of the key and iris template, protecting these from being accessed by intruders. In the event that a user's access data is compromised, a new random key can be generated for that user, resetting the system without loss of functionality. This combination of accuracy and security makes the Cambridge biometric cryptosystem a strong choice for our subsequent software implementation and hardware acceleration.

4. Hardware Acceleration

Since the development of the computer, people have been trying to make computation faster. This was initially achieved through increasing the speed of Central Processing Units (CPUs) that handle a single stream of instructions. But recently, CPUs have started to reach a level where their performance cannot be increased without generating more heat than can be dissipated in a reasonable amount of time and space [8]. This has caused the growth of research into new areas of computational acceleration.

New methods of computation focus on two aspects: running multiple instruction sequences (known as "threads") simultaneously, and decreasing the execution time of certain instructions. These goals are usually accomplished through the use of multiple CPU cores running in parallel, a graphical processing unit (GPU), or custom hardware. Using multiple CPU cores allows the execution of multiple threads simultaneously on one CPU. This is effective in scenarios where programs have many sequences of instructions that are not dependent on one another. However, the fact that the hardware in the CPU is not being modified, just made more plentiful, means that instructions are not executed any faster. Programs without opportunity for parallel computation do not benefit from multi-core hardware architecture. Similarly, making use of a GPU allows significantly increased parallelism by performing multiple calculations simultaneously. In addition, GPUs are optimized to perform certain types of calculations commonly used in graphics processing, allowing faster execution time for these instructions. The disadvantage of GPUs is that they are optimized for particular types of calculations, reducing the benefits they provide for computation of a more general type. Alternatively, custom hardware allows particular hardware units to be defined to handle certain computations [9]. Systems can be tailored to optimize for the most common form of computation in the target application. For custom hardware, the amount of parallelism in computation is defined by the amount of hardware space allocated by the designer for additional hardware for each particular functionality. The disadvantage of this method is that custom hardware must be designed specifically for the system it is intended for, as opposed to off-the-shelf CPUs or GPUs. This takes significant amount of design time, and also requires the use of customizable hardware resources, whether a Field Programmable Gate Array (FPGA) or a Complex Programmable Logic Device (CPLD). These hardware devices consist of a variety of hardware resources that can be configured in varying manners to produce the custom hardware.

Hardware acceleration also changes energy efficiency in addition to processing speed. A system using hardware acceleration has more hardware to power than the same system without hardware acceleration, causing an increase in the power consumed. However, the energy usage also depends on the time, which typically decreases during the process of hardware acceleration [10]. Depending on the rates of change in time and power consumption, it is possible to improve the total energy usage with hardware acceleration.

This project makes use of an FPGA for the implementation and testing of the biometric cryptosystem, in order to be able to tailor the system as exactly as possible for the maximum potential benefit. An FPGA was chosen as the method of customizable hardware due to its increasing use in commercial product releases. An FPGA is an integrated circuit consisting of logic blocks and Random Access Memory (RAM) blocks designed to be configured by a customer or a designer after manufacturing. The hardware used for the project is a Xilinx Zedboard [11], a system-on-a-chip (SoC) that incorporates the functionality of an ARM® dual-core Cortex$^{\text{TM}}$ -A9 MPCore$^{\text{TM}}$ Processing System (PS) and a Zynq-7000 All-programmable FPGA [11]. This system uses the processor as its main computational unit, while being able to connect to the

FPGA portion of the chip for hardware acceleration of various functions. This makes the platform an ideal setup for testing custom hardware acceleration.

In addition to the hardware package, the Xilinx Zedboard also comes with software to program the system. Xilinx Software Development Kit (SDK) and Vivado software packages are used for programming the software and hardware components, respectively [12]. The Xilinx SDK software version 2013.4 used in this work allows a user to create, download, and run C code on the Zedboard, while the Xilinx Vivado Design Suite version 2013.4 used in this work allows the user to create custom hardware setups. These setups can be converted into binary configuration files used by SDK to setup the Zedboard. This setup allows the algorithm to be run both with and without hardware acceleration. Different software functions in the algorithm are implemented through the hardware acceleration process and the results in terms of performance and efficiency are compared.

5. Methodology

The process used to generate a hardware-accelerated version of the Cambridge biometric cryptosystem consisted of several steps. The first component of the project was identifying the target function for acceleration. Once this was completed, a hardware accelerator was designed for the target function. Upon completion of the hardware accelerator, the software was modified to make use of the new component. The resulting system was then tested and analyzed to determine its overall effectiveness. The process flow diagram for this methodology is shown below in Figure 2.

The system setup for this project involved a Zedboard, a computer with Xilinx SDK and

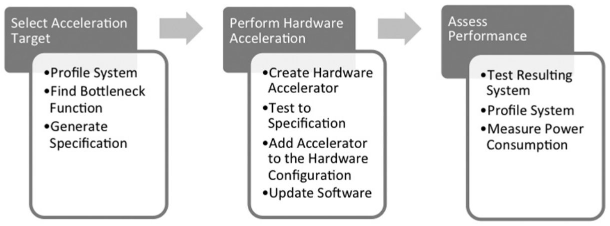
Select Acceleration Target
•Profile System
•Find Bottleneck Function
•Generate Specification
Perform Hardware Acceleration
•Create Hardware Accelerator
•Test to Specification
•Add Accelerator to the Hardware Configuration
•Update Software
Assess Performance
•Test Resulting System
•Profile System
•Measure Power Consumption

Figure 2. Methodology flowchart.

Xilinx Vivado software and at least two available USB ports, and two USB-microUSB cables. The Zedboard was plugged in and set to cascaded JTAG mode by setting bits MIO[6 : 2] to ground. One USB-microUSB cable was connected between the UART port of the board (J14) and the computer, and the other was connected between the JTAG (J17) port and the computer. The board was then powered on. In Xilinx SDK, the port used to program the FPGA was set to the port that was connected to the JTAG pin.

This research made use of a C implementation of the Cambridge biometric Cryptosystem developed by the authors for a past research project [13]. In order to find a target function for hardware acceleration, the system was profiled using the profiling option built into the SDK software. This functionality uses a timer to generate hardware interrupts periodically. At each hardware interrupt, the system stores its current location in the program flow. The number of data points can be used to determine total system runtime while the locations of the data points can be used to determine the amount of time spent on each function, and the number of times each function was called during execution. The snoop control unit (SCU) timer on the Zedboard was used to generate the interrupts.

The profiling results provided a breakdown of the total time required for computations in the system, and how that time was spent. Using this data, the function that required the most computation time was selected as the target for hardware acceleration.

The next phase of the research was to design the hardware accelerators. By analyzing the target software function, specifications for input and output were created. Based on these input and output specifications, a control flow for the hardware accelerator was created. Using Xilinx Vivado Design Suite and the VHDL hardware description language, these hardware accelerators were implemented. Each accelerator consisted of two components: a computational component and an interface component. The computational component received input and control signals as chosen in the specification and performed the computations the system was designed to complete. The interface component handled the communication between the computational component and the Advanced

Extensible Interface (AXI) between the programmable logic (PL) and the PS. This involved converting between data registers and input or output signals and converting control registers to individual control signals for the computational component. Each component was individually tested to ensure that it met specifications prior to combining the pieces to generate the finished hardware accelerator. This accelerator was then added to the hardware configuration.

Once the hardware setup was updated with the new accelerator, the next step was to edit the software project to use the hardware accelerator instead of the original software function. This was done using the SDK software. A *C* function was written with the same input and output arguments as the original function. Rather than performing the computation, this function would send the appropriate data to the memory-mapped registers used for input and control. The function would then wait for a signal from the hardware accelerator indicating that the computation was finished. Upon reception of this signal, the function would read the output register and return the appropriate results. Each instance of the original computation function in the project was then replaced by a call to the new function, completing the software changes. After each change, the software was tested to ensure that its functionality was unaffected.

Performance evaluation was the final phase of the project. After each hardware accelerator was added to the system, the resulting system was profiled as described above. This provided data about the runtimes of the overall system and the specific function accelerated. This data was used to calculate total runtimes and system speedups for the entire system and the target functions. Additionally, the system power usage report generated by Vivado was used to obtain the power usage of each system. Multiplying the power usage by the total time per computation provided an estimation of the amount of energy consumed by that calculation. These time and energy statistics allow us to compare the different systems and evaluate the effectiveness of the acceleration process.

6. Results and Discussion

6.1. Initial Software Profiling

In order to select a target function for hardware acceleration, the initial software implementation of the Cambridge biometric cryptosystem was profiled as described above. Due to the separation of the enrollment and verification processes, three distinct runtime profiles were generated: one for each process run individually, and one for enrollment and verification run in sequence. These runtime profiles are shown in Figure 3, Figure 4, and Figure 5, respectively. For this data, time spent on profiling functions has been removed in order to focus on the algorithm runtime. Functions that took less than 50 microseconds were combined into the “others” section to simplify the information. According to the verification and sequential profiles, the functions with the most computational cost and thus the greatest potential for hardware acceleration are the mult_polys function, the hadamard_decode function, and

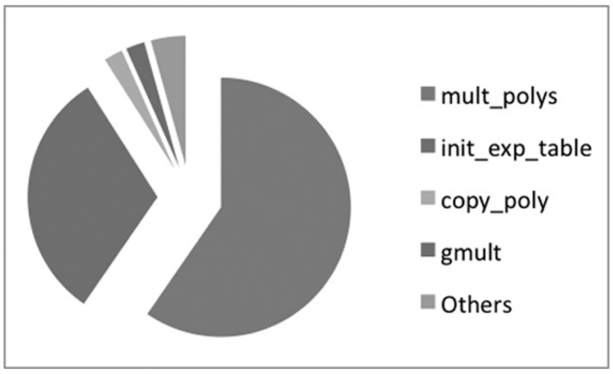

Figure 3. Profiling results for the initial software implementation of enrollment.

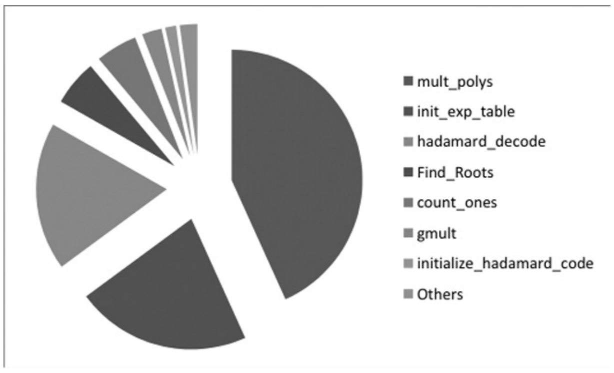

Figure 4. Profiling results for the initial software implementation of verification.

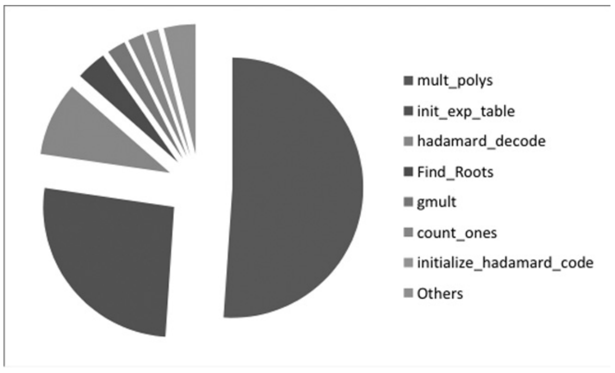

Figure 5. Profiling results for the initial software implementation of enrollment followed by verification.

the init_ exp_table function. The profile for enrollment does not have the same cost for the hadamard_decode function since this function is only used during verification. Thus hardware acceleration began on the mult_polys function.

6.2. Hardware Accelerator Setup

The hardware acceleration process involves several different phases, as described in Section V. The results of each of these phases will be discussed individually below.

The first step in the acceleration process was creating an input/output specification for the mult_polys function. This function takes two input arrays and returns one output array. In the general case, the input arrays can be of any size, and the size of the output array is equal to the sum of the sizes of the two input arrays. For this particular program, both input arrays have the size equal to twice the number of parity bytes used in the Reed-Solomon Codes: 24 bytes. The mapping from input to output is very similar to that of regular polynomial multiplication. Each array element is treated as a coefficient to a polynomial where the index of the array corresponds to the power of the term. When doing regular polynomial multiplication, each term in the first array would be multiplied with each term in the second array. The result from each multiplication is added to the output term with power (index) equal to the sum of the powers of the two inputs. In the mult_polys function, this process is changed slightly. Multiplication across a Galois Field is used rather than multiplication for combining terms. The addition used for combining the output terms is replaced with the XOR operation. This process, combined with the knowledge of the input and output sizes, completes the specification of the mult_polys function.

The hardware accelerator was designed to perform the same computations as the software function, but in a manner that allowed multiple computations to be executed in parallel. The first step was designing a component that would perform multiplication across a Galois field. In the software version of the system, Galois multiplication was performed by taking the log of both inputs across a Galois field, and then taking the Galois exponential of the sum of the results. This method works for all non-zero inputs. Any zero inputs were handled by a check prior to computation, and simply returning zero if a zero input existed. The Galois multiplication hardware was designed using the same algorithm. Memory blocks were created for the Galois logarithm and Galois exponentiation functions. Once these memory blocks were completed, combining them to create Galois multiplication became a relatively simple task.

After the Galois multiplication component was designed and tested, the mult_polys accelerator could be constructed. The component design diagram for this component can be found in Figure 6. This hardware component makes use of a number of Galois multiplication components equal to the size of the first input array: 24 in this case. For each component, the corresponding term of the first input array is the first factor.

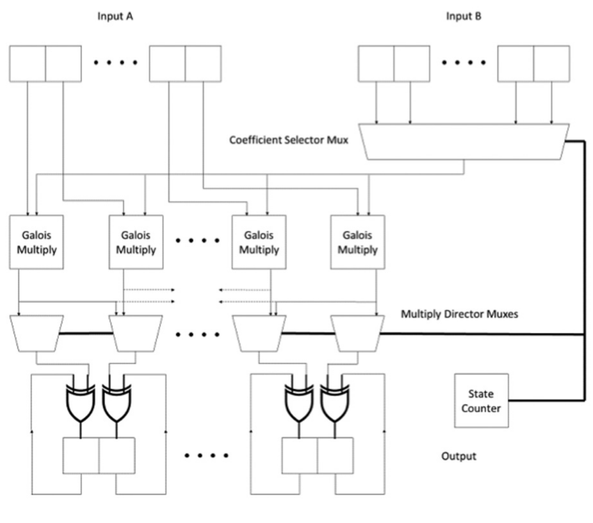

Figure 6. Mult_polys hardware accelerator schematic.

The second factor is time dependent, but component independent. That is, the second factor changes over time, but is the same for any component at the same time. At the beginning of the computation, the second factor is the first term of the second array. Each computation cycle, the next lowest index term is used as the second factor for the components. These computations combine to create every combination of terms between the two input arrays.

The output of the computation is stored in a larger output vector. When computation starts, this vector is initialized to all 0's. Upon completion of a computation, the result of the multiplication component is XORed with the appropriate output term, and then stored as the new output term. Because the XOR operation is commutative and associative, the order of these operations does not affect the computation. The appropriate output term is found by adding the index of the second array used as a factor to the multiplication component number. This results in the lower order terms being processed before the higher order terms, with the same number of computations taking place during each computation cycle. After all computation cycles have been completed, the output vector is complete, and can be sent back to the processor.

Creating the AXI Interface to the processor involved developing input and output buffers, and a control buffer that the processor could use to control the system by setting flags. The control flags are simple: one bit for the processor to tell the accelerator to begin computation and one bit for the accelerator to tell the processor when computation has finished. For this component, the input and output buffers were the same. Upon receiving a start signal, the accelerator reads the input buffer into its internal memory, then clears the control buffer. When the computation completes, the system overwrites the result to the input/output buffer and writes a 1 to the completion bit and a 0 to the start bit. This interface completes the input and output requirements for this accelerator.

The hardware components were tested using the simulation software in Vivado. The inputs to the system were forced to particular values, and the output signals were observed. This allowed the designers to validate the correctness of the system.

After the simulation phase, adding the hardware accelerator to the hardware system took place in the Vivado environment. An instance of the accelerator was created, and connected to the processor using an AXI interface. This results in the processor treating the accelerator as a memory-mapped peripheral. This change also required setting one of the PL clocks to control the accelerator. For this research, a clock frequency of 200 MHz was chosen since it was the highest frequency supported. This new hardware configuration was then imported into SDK in order to update the software.

Updating the software involved replacing calls to the mult_ polys function with calls to the hardware accelerator. A replacement function with the same signature was created that could be called by other sections of the code that required this computation. This function, called mult_polys_HW, made use of the memory-mapped accelerator. Using the memory locations specified in the configuration files, it would load the inputs into the appropriate registers, then write a 1 to the start flag. The function would then periodically check the completion flag. Upon the flag becoming a 1, the function would read the data from the registers and return them to its caller. Once this function was completed and tested, calls to mult_polys in the project were replaced with calls to mult_polys_HW. This completed the acceleration cycle.

6.3. Hardware Accelerator Profiling

Upon the completion of the hardware accelerator for the mult_ polys function, the system was again profiled to determine its effectiveness. The results of comparing the runtime profiles before and after hardware acceleration are shown in Figure 7. As this graph shows, the runtime of the mult_polys function is much lower in the accelerated version, with a relative speedup of approximately 25X. This has resulted in the overall system speedups shown in Table 1. The speedup is the greatest for the enrollment process since the mult_polys function occupies a larger fraction of the total runtime. The verification process has a lesser speedup. The speedup for the sequential run is a balancing of the individual speedups, with weight assigned to the

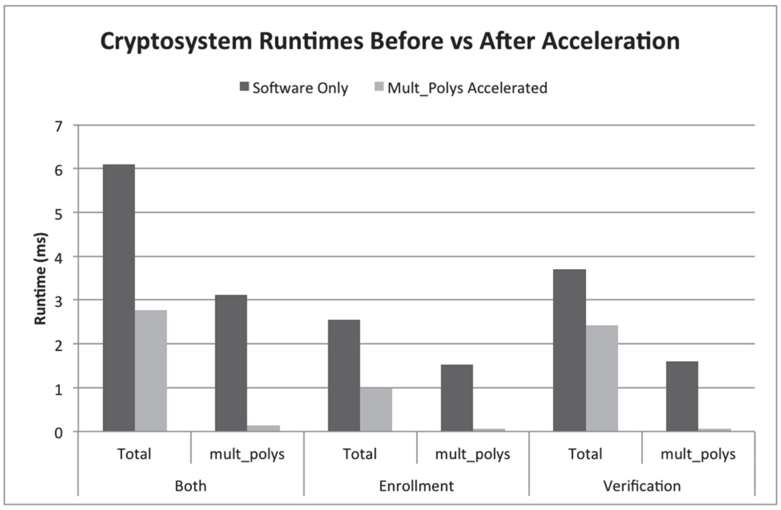

Figure 7. Cryptosystem runtimes before and after hardware acceleration.

individual processes based on their relative runtime.

Function	Runtime before Acceleration (ms)	Runtime after Acceleration (ms)	Speedup
Enrollment	2.55	0.97	2.63
Verification	3.7	2.43	1.52
Both	6.09	2.77	2.20

Table 1. Hardware acceleration speedups.

The hardware acceleration also affected the total energy usage of the system. The additional hardware component caused the power usage to change from 1.477 W to 1.704 W, representing a 15.4 % increase. However, the amount of energy used for any given computation was reduced due to the improved runtime. Table 2 shows the energy consumption of the resulting systems. Since the power usage is constant between different computation sets, the energy usage follows the same patterns as the runtime data.

Function	Energy before Acceleration (mJ)	Energy after Acceleration (mJ)	Percentage of Reduction
Enrollment	3.77	1.65	56.1
Verification	5.46	4.14	24.2
Both	8.99	4.72	47.5

Table 2. System energy consumption.

These results show that hardware acceleration is an effective choice for improving the overall efficiency of biometric cryptosystems. The hardware acceleration caused significant decreases in the runtime of the original system. These runtime decreases help outweigh the additional power that they require, resulting in overall energy cost reductions per computation.

6.4. Comparison with Multi-Core Acceleration

Comparing the results of custom hardware acceleration methods to alternative methods allows researchers to analyze the similarities and differences of methods and their relative levels of effectiveness. One such comparison can be drawn between this work and the many-core acceleration methods applied to the Cambridge biometric cryptosystem [14], in which case the authors used the 48-core Intel Single-Chip Cloud Computer (SCC) platform.

In this work, custom hardware acceleration was used to achieve a speedup of approximately 2.20X. This was achieved by accelerating the mult_polys function. By contrast, the many-core acceleration attempted to improve the hadamard_decode function. This is because the mult_polys function is of sequential computation type, which is suitable for hardware accelerator design; while the hadamard_decode function is of parallel computation type, which is suitable for being distributed over many cores for parallel processing. The acceleration of hadamard_decode resulted in a speedup of up to 1.18X [14]. The limited effectiveness of this approach can be explained with a couple of reasons. The first reason is that the hadamard_decode function takes a lower percentage of the computation runtime than that of the mult_polys function, as shown in Figures 3-4. Another reason is that in the many-core implementation, the MPI-like (message passing interface) programming model contributed significant communication overhead.

This shows that custom hardware acceleration can have greater potential impact on system runtimes than many-core acceleration for this application. However, due to the fact that many-core systems are more prevalent and versatile than custom hardware systems, which must be designed for a particular operation, a hybrid approach would provide the benefits from both approaches.

7. Conclusions

Biometric cryptosystems represent a future direction that provides user security without the drawbacks of traditional password-based approaches. By combining biometrics with cryptography, biometric cryptosystems allow accurate matching of biometrics in an encrypted and secure domain. This is an important step towards creating a network where data can be protected efficiently and securely. Henceforth, improving the efficiency of biometric cryptography would greatly enhance its potential to be readily deployed towards modern network security.

In this work, a concrete software implementation of the biometric cryptosystem developed by Anderson, Daugman, and Hao at the University of Cambridge has been optimized using hardware acceleration. The result is a system that can perform user enrollment in 0.97 ms and user verification in 2.43 ms on a Xilinx Zedboard System-on-a-Chip (SoC) platform, representing a 2.63X and 1.52X speedup comparing with the software counterpart. This system requires only 1.704 W of power, resulting in energy consumption of 1.65 and 4.14 mJ for enrollment and verification, respectively. This represents a 56.1 % and 24.2 % reduction comparing with the software counterpart.

This work also compares the custom hardware acceleration with a many-core acceleration implementation of the same system. This comparison illustrates the greater acceleration potential provided by custom hardware acceleration compared to parallel computing on many-core platform, and the corresponding increase in designer overhead. Understanding these tradeoffs allows system designers to select the best hardware system for their application.

In addition to providing a concrete implementation of a biometric cryptosystem, this project provides a process that can be used as a guideline for future applications of hardware acceleration to biometric cryptosystems for network security. Making use of this methodology will allow for the development of systems that take significantly less runtime and less energy per computation, significantly increasing their useful potential.

Acknowledgment

The Zedboard SoC platform used for FPGA implementation is provided through Xilinx University Program. The Intel Single Chip Cloud Computer Platform is provided by Intel Labs. This work is supported by the National Science Foundation under Grant Numbers IIP-1332046, IIP-1068055 and ECCS-1301953. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] T. Hey, S. Tansley, K. Tolle, The fourth paradigm, data-intensive scientific discovery. Microsoft Research, Redmond, WA.

[2] A. Jain, A. Ross, U. Uludag, Biometric template security: challenges and solutions. *European Signal Processing Conference*, (2005).

[3] U. Uludag, S. Pankanti, S. Prabhakar, A.K.Jain, Biometric cryptosystems: issues and challenges. *Proceedings of the IEEE*, **92**(6), (2004) pp. 948–960.

[4] A. ROSS, A. OTHMAN, Visual cryptography for biometric privacy. *IEEE Transactions on Information Forensics and Security*, **6**(1), (2011) pp. 70–81.

[5] C. RATHGEB, A. UHL, A survey on biometric cryptosystems and cancelable biometrics. *Journal on Information Security*, **2011**, (2011) pp. 3.

[6] R. ANDERSON, J. DAUGMAN, F. HAO, Combining cryptography with biometrics effectively. Technical Report 640, University of Cambridge, 2005.

[7] C. CLARK, Reed-Solomon error correction. British Broadcasting Company Research and Development White Paper, WHP 031, July 2012.

[8] C. LIU, R. DUARTE, O. GRANADOS, J. TANG, S. LIU, J. ANDRIAN, Critical path based hardware acceleration for cryptosystems. *International Journal of Advancements in Computing Technology*, **4**(1), (2012) pp. 438–452.

[9] D. LAU J. BLACKBURN C. JENKINS, Using c-to-hardware acceleration in FPGAs for waveform baseband processing. *Software Defined Radio Technical Conf. Product Exposition*, (2006).

[10] C. LIU, O. GRANADOS, R. DUARTE, J. ANDRIAN, Energy efficient architecture using hardware acceleration for software defined radio components. *Journal of Information Processing Systems*, **8**(1), (2012) pp. 133–144.

[11] GETTING STARTED WITH ZEDBOARD. Version 7, AVnet Electronics Marketing.

[12] ZYNQ ALL PROGRAMMABLE SOC LINUX-FREERTOS AMP GUIDE. Version 2013.10, Xilinx, November 25, 2013.

[13] C. MCGUFFEY, C. LIU, S. SCHUCKERS, Implementation and optimization of a biometric cryptosystem using iris recognition. Biometric and Surveillance Technology for Human and Activity Identification XII, SPIE DSS 2015, Baltimore, Maryland, USA, April 20-24, 2015.

[14] C. MCGUFFEY, C. LIU, Multi-core Approach Towards Efficient Biometric Cryptosystems. *The 2015 International Workshop on Embedded Multicore Systems (ICPP–EMS 2015), in conjunction with ICPP 2015*, Beijing, China, September 1-4, 2015.

[15] P. REID, *Biometrics and Network Security*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

Received: August, 2015
Revised: October, 2015
Accepted: October, 2015

Contact addresses:

Charles McGuffey
Clarkson University
8 Clarkson Avenue
Potsdam
NY 13699
USA
e-mail: `mcguffcj@clarkson.edu`

Chen Liu
Clarkson University
8 Clarkson Avenue
Potsdam
NY 13699
USA
e-mail: `cliu@clarkson.edu`

Stephanie Schuckers
Clarkson University
8 Clarkson Avenue
Potsdam
NY 13699
USA
e-mail: `sschucke@clarkson.edu`

MR. CHARLES MCGUFFEY obtained his B.S. Degree with Honors in Computer Engineering and Computer Science in 2015 from Clarkson University, Potsdam, New York, USA. He is currently a Computer Science Ph.D. candidate at Carnegie Mellon University. His research interests include hardware acceleration, many-core computing and algorithm design.

DR. CHEN LIU received the B.E. Degree in Electronics and Information Engineering from the University of Science and Technology of China in 2000, the M.S. Degree in Electrical Engineering from the University of California, Riverside in 2002, and the Ph.D. Degree in Electrical and Computer Engineering from the University of California, Irvine in 2008, respectively. Currently he is an assistant professor in the Department of Electrical and Computer Engineering at Clarkson University, Potsdam, New York, USA. His research interests are in the areas of multi-core multi-threading architecture, hardware acceleration for scientific computing and the interaction between system software and micro-architecture. He is a member of IEEE, ACM, and ASEE.

DR. STEPHANIE SCHUCKERS is the Paynter-Krigman Endowed Professor in Engineering Science in the Department of Electrical and Computer Engineering at Clarkson University and serves as the Director of the Center of Identification Technology Research (CITeR), a National Science Foundation Industry/University Cooperative Research Center. She received her doctoral degree in Electrical Engineering from The University of Michigan. Professor Schuckers's research focuses on processing and interpreting signals which arise from the human body.