High-Performance Low-Energy Implementation of Cryptographic Algorithms on a Programmable SoC for IoT Devices

Boyou Zhou, Manuel Egele, Ajay Joshi

ABSTRACT

Due to severe power and timing constraints of the "things" in the Internet of things (IoT), cryptography is expensive for these devices. Custom hardware provides a viable solution. However, implementations of cryptographic algorithms in the devices need to be upgraded frequently compared to the longevity of these "things". Therefore, there is a critical need for reconfigurable, low-power and high-performance cryptography implementations for IoT devices. In this paper, we propose to use an FPGA as the reconfigurable substrate for cryptographic operations. We demonstrate our proposed approach on a Zedboard, which has two ARM cores and a Zynq FPGA. The implemented cryptographic algorithms include symmetric cryptography, asymmetric cryptography, and secure hash functions. We also integrate our cryptographic engines with the OpenSSL library to inherit the library's support for block cipher modes. Our approach shows that the FPGA-based reconfigurable cryptographic components consume between 1.8× and 4033× less energy and run between 1.6× and 2983× faster than the software implementation. At the same time, the FPGA implementation of cryptographic operations is more flexible compared to custom hardware implementations of cryptographic components.

1. INTRODUCTION

The current trend toward connecting a variety of electronic devices, such as thermostats, robots, and smart devices, has led to the dawn of the Internet of Things (IoT) paradigm [1]. IoT devices frequently handle sensitive information, such as command and control signals for smart homes, automobiles, and factory floors. Thus, our society would be best served by leveraging strong cryptography primitives that can guarantee the integrity and confidentiality of IoT data. Strong cryptographic primitives could have helped avoid some of the most recent security breaches in this area (e.g., [2–6]). Furthermore, the growth of data-volume in the IoT space is projected to be unprecedented. For example, Cisco predicts that the amount of data produced and consumed by $4 \sim 5$ billion IoT devices will grow to 15 exabytes (a billion billion bytes) by 2020 [7]. At the same time, protecting the data at that scale will become increasingly challenging [8]. The increasing demand for data has led to the following conflict. On the one hand, capabilities of IoT devices are constrained by a low power budget, which can be as low as 6.45 μw watts [9] in energy harvesting applications. On the other hand, energy-intensive cryptographic operations are required to protect all the data produced and consumed by the IoT devices. The perspective of growth, device longevity, and the observation that cryptographic operations are relatively costly when compared to regular system functionality, call for novel solutions for IoT devices to provide cryptographic primitives that are scalable, flexible and energy-efficient.

Historically, energy-efficient and scalable implementations of cryptographic algorithms have been introduced by chip manufacturers as dedicated crypto-engines, which are essentially Application Specific Integrated Circuit (ASIC) units embedded with the processor [10–12]. These ASIC implementations provide performance and energy efficiency advantages but they are not flexible. The longevity of IoT devices necessitates flexible cryptographic functionality. For example, cryptographic algorithms that are found insecure (e.g., DES) should be replaced with more secure algorithms (e.g., AES). Thus, to achieve high performance, energy efficiency, and flexibility, we propose to implement the cryptographic primitives in a Field Programmable Gate Array (FPGA) that is integrated with a processor in a System-on-Chip (SoC) configuration. In this paper, we design and thoroughly evaluate the performance, energy efficiency, and flexibility of various cryptographic algorithms. To this end, we evaluate each algorithm in a realistic setting where the cryptographic functionality is implemented in an FPGA and exposed to Linux user-space programs through a simple modification of the OpenSSL [13] cryptographic library. Specifically, we perform our evaluations on a Digilent Zedboard, a System on Chip (SoC) platform that features two ARM cores connected to an FPGA. This setup further illustrates that our solution can seamlessly integrate with existing software that depends on the hugely popular OpenSSL library. For example, a minimal installation of the Ubuntu Linux (16.04) operating system distribution contains as many as 113 packages depending on the OpenSSL library. Our implementation seamlessly benefits all of these applications without further modifications to these packages. The detailed contributions are listed below:

- To provide IoT devices with high performance and energy-efficient cryptographic primitives, we propose, a flexible hardware solution based on commodity off-the-shelf FPGAs (Section 3).
- To demonstrate feasibility, we implement cryptographic engines for symmetric and asymmetric cryptographic algorithms, as well as cryptographic hash functions on the Zedboard platform in a ready-to-use system. We develop our functions in an application program for evaluations (Section 4).
- We achieve flexibility in the online reconfiguration of FPGA hardware and block cipher modes in software. As our FPGA-based symmetric encryption engine is integrated into the OpenSSL library, we inherit support for the various block cipher modes implemented by OpenSSL (Section 4).
- We synthesize implementations of AES, Rivest & Shamir & Adleman (RSA), Data Encryption Standard (DES) and Secure Hash Algorithm (SHA) hardware as obtained from Opencores [14] (Section 5).

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

- We thoroughly evaluate these implementations along the dimensions of performance, energy, and power. To assess improvements along these dimensions, we also compare the FPGA-based implementations with their respective software-only counterparts (Section 5).

2. RELATED WORK

Since FPGA enables building flexible hardware, various implementations of the cryptographic algorithms on FPGA substrates have been proposed. An FPGA implementation of AES was first shown to have benefits in terms of performance improvements and energy savings in 2004 [15]. Here, the stream reached a speed of 204Mbps on average for AES encryption. Since then, FPGAs were explored as a potential solution to cryptographic applications for low power devices. Good et al. proposed a minimal data path design of AES, which is used only on 8-bit data path in all the operations in AES [16]. This design made the entire AES fit in small programmable areas. Hamalainen et al. presented an upgraded version with the cost of using 3.1k gates [17] compared to [16] using $\sim$4.7k gates.

With the increasing area of FPGAs, more and more resources could be used for cryptographic designs. Bulens et al. first put one full round of SubBytes operations in AES on FPGA in 2008 [18]. With pipelined designs, the plain text can be continuously streamed into the cryptographic units for encryption/decryption to achieve high throughput. Good et al. showed the designs from the fastest to the smallest, which can provide different design points on different hardware platforms, in order to achieve a trade-off between speed and area [19]. Our design achieved a speed of 397Mbps compared with their design, which had a speed of 358Mbps encryption/decryption speed. Modern FPGAs have enough programmable logic that can fit the expansion of 10 to 14 rounds of SubBytes operations. Hoang et al. showed a fast implementation of AES with the key size of 128 bits, with a look-up table of SubBytes [20]. There are also partial and dynamic reconfigurable implementations for more flexible solutions on FPGA [21]. As partial configurations are designed for high-end FPGA applications, low-energy FPGA has not yet widely adopted the partial configurations for their applications.

Side channel attacks use power, time or other physical properties to break cryptographic algorithms. Once discovered, it is possible to prevent a given side channel attack by modifying the implementation of cryptographic algorithms. Robust designs, such as [22, 23], are resistant to Differential Power Analysis (DPA) leaking AES keys in their implementation. However, robustness and DPA resistant designs come at a higher cost in performance or energy consumption. Other cryptographic algorithms, such as Hummingbird [24] and Whirlpool [25], have also been implemented on FPGA. Recent works have shown that FPGA can be an ideal solution for modern cryptographic computation. Saarinen et al. implemented Authenticated Encryption with Associated Data (AEAD), AES-GCM on Zedboard [26]. Saarinen et al. also showed one variant implementation of Whirlpool [27]. These works contain detailed explanations of implementation and performance analysis, but a lack of power and energy analysis and multi-crypto-engines implementation. Our work includes performance, power comparisons measurements on FPGA and software cryptographic algorithm implementations. Due to the particular importance of energy budget in IoT designs, we also include the energy cost comparison between measurements of FPGA and software implementations.

3. BACKGROUND

Performance, energy consumption, and security are three major considerations for IoT devices. Using software-based cryptographic operations on IoT devices enables us to upgrade the IoT device security. Nevertheless, these software-based cryptographic operations are expensive in terms of energy for IoT devices. Hence, the custom hardware implementation of the algorithm is a viable option, as it can boost performance and have lower power consumption compared to its software implementation. Yet cryptography implementations are frequently changed many times in the lifetime of IoT devices. The IoT devices can be employed for 5-10 years [1]; on the contrast, the OpenSSL library can be updated once a month on average over last 10 years [13]. These custom hardware implementations cannot be changed once they are manufactured. Using FPGAs solves this problem by allowing the hardware to receive updates.

FPGAs provide a good middle ground as they provide flexibility. The FPGA clock speed is much slower than the clock on a general purpose processor. However, the FPGA can complete the computation intensive functions much faster than the software implementation. For example, a software implementation of AES can take up to 600,000 cycles on an ARM machine, while a hardware implementation only needs 20 cycles [14]. Even though FPGA clock speed can be $10\times$ slower than General Purpose Processor (GPP), the FPGA implementation can still be up to $3{,}000\times$ faster.

In addition to the performance boost, FPGA implementation also consumes much lower power to complete the same amount of computation. Compared to GPP, FPGAs do not have extra operations such as instruction fetching or instruction decoding. Data paths on the FPGA are customized to the algorithm in order to achieve its maximum computation efficiency. Without operations to loading instructions in GPP and with customized data path, FPGAs have much higher energy efficiency in computationally intensive applications.

FPGA provides a lot of benefits in performance and power, but it also has some drawbacks. Since the manufacturing technologies are different for FPGAs and GPPs, manufacturers only provide GPP and FPGA connected in the same package, while on different dies (off-die). In some platforms, they are even in different packages (off-chip). One of the key drawbacks is that in order to access discrete FPGA, GPPs are required to use off-die or off-chip channels, which have higher latency compared to on-chip channels. SoC solutions like Zedboard, where GPP and FPGA are integrated on the same chip, can communicate through on-chip buses, like the Advanced eXtensible Interface (AXI) bus, which largely reduces the communication latency. AXI ports are also used for off-die, high-speed communications, such as CPU communicating with main memory. These Zedboard platforms have lower communication cost and are off-the-shelf solutions. Thus we have used Zedboard as our target system.

Numerous cryptographic operations have been implemented on FPGA (see §2). Nevertheless, any FPGA has its own limitations on area usage and available resource to implement all the cryptographic algorithms. This demands that there should be a re-configurable scheme, where hardware can be configured with available crypto-engines. We propose to use FPGAs for crypto operations as FPGAs can be reconfigured and at the same time, they are more energy efficient than software implementations. Considering the limitations on the FPGAs, we implement our system on Zedboard and evaluate the performance and energy cost of our proposed scheme.

4. EXPERIMENTAL SETUP

To evaluate the use of FPGAs as an energy-efficient re-configurable substrate for crypto-engines, we use AES, RSA, and SHA as examples of symmetric crypto, asymmetric crypto, and hash functions, respectively. We demonstrate our approach on Zedboard platform by *Digilent*. The Zedboard platform uses the *Zynq 7000* SoC chip containing two *ARM9* cores and one *Zynq 7000* FPGA. The FPGA substrate interfaces directly with the system main memory through AXI ports. The persistent storage

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

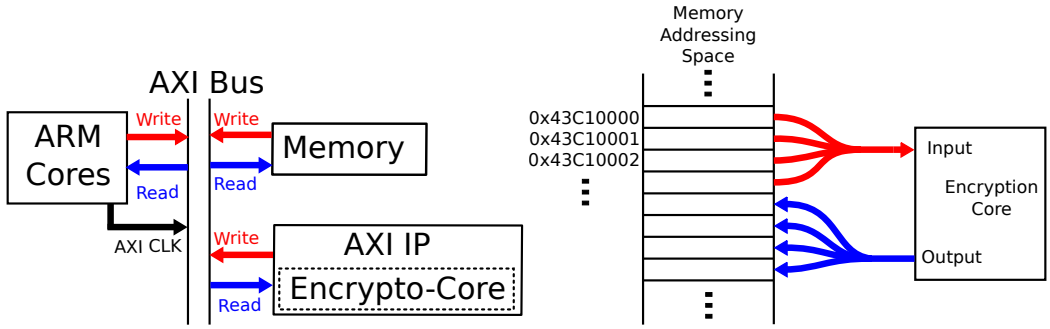

Figure 1: System Architecture for AES Encryption Engine

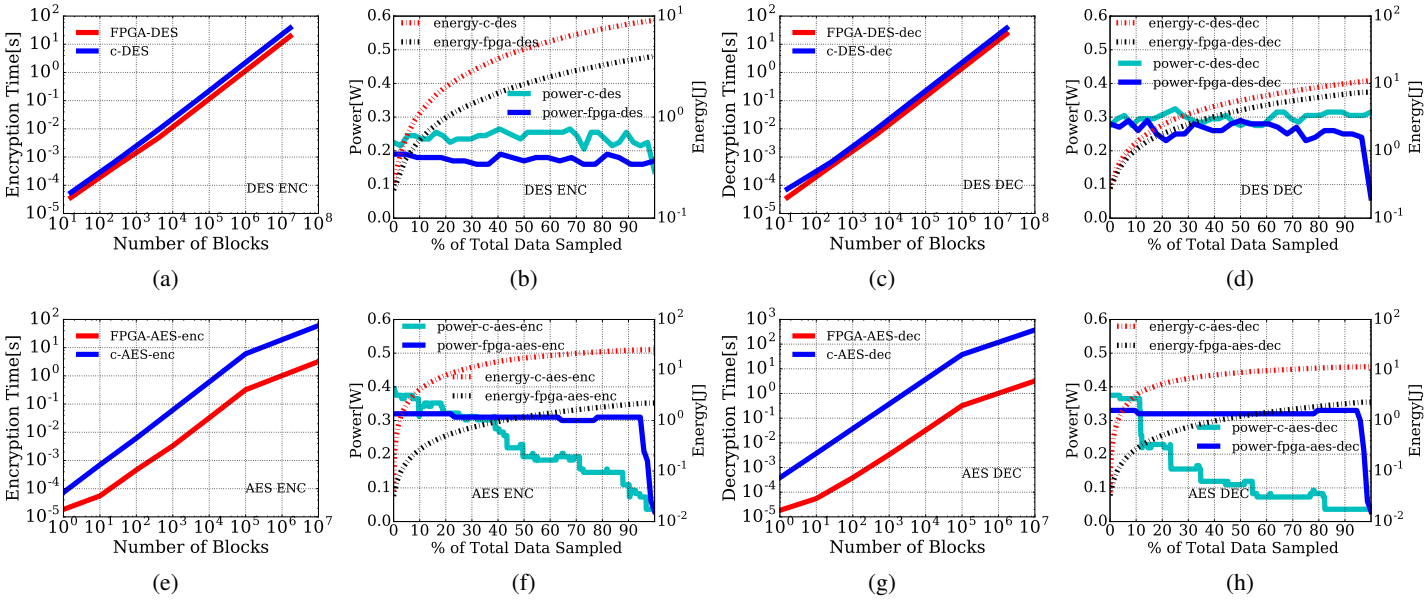

Figure 2: Comparisons between C implementation (software), and FPGA implementation (hardware) of DES, AES in encryption (ENC) and decryption (DEC). We encrypt and decrypt data ranging from 16 blocks to 16^6 blocks in DES experiments, and from 1 block to 10^7 in AES experiments. (a) and (c) show time comparisons in DES. Encryption is 1.9× faster and decryption is 1.6× faster in DES. (b) and (d) show power consumption and energy comparisons. Encryption has 3.9× energy reduction and decryption has 1.9× energy reduction. (e) and (g) show time comparisons in AES. Encryption is 18.8× faster and decryption is 116.6× faster on FPGA. (f) and (h) show power consumption and energy comparisons in AES. Encryption has 13.9× energy reduction and decryption has 6.0× energy reduction.

on the Zedboard has two partitions, one is the boot partition for loading FPGA bitstreams and starting the Linux Kernel. The other partition contains an installation of the Linaro Linux distribution [28]. The boot partition contains three files: one file for mapping bitstreams to FPGA, one file for Linux Kernel, and the device tree file defines all the devices that are present in the system. To minimize unwanted energy consumption, we select minimum device tree, which boots up the Linux system and configures the *Zynq 7000* FPGA. We build a system with the device tree we created, which consists of no other functionalities but Linux running on the ARM core, AXI peripheral blocks, and encryption IP cores as it is shown in Fig 1. *Vivado* provides a custom AXI IP core wrapper, which integrates well with CPU communication channels on the one hand, and wraps the custom design inside the IP wrapper on the other hand. The ARM core provides at maximum 512 32-bit channels as communication channels for communications. The interfaces of these channels on the FPGA side can be synthesized into registers on the programmable logic. On the ARM core side, these channels are memory mapped I/Os, which can be accessed by memory operations.

We downloaded the hardware implementation of AES, DES, RSA, and SHA for evaluations, from Opencores [14], synthesized them in the custom AXI wrapper and mapped them on the Zedboard. The input, output, and configuration bits connect to communication channel interface registers, and the AXI wrapper along with our custom designs are mapped to the FPGA. These registers can be addressed with memory offset provided by *Vivado* after the synthesis of the wrapper. With the calculated memory locations of these registers, offloading encryption operations consists of the writing phase and reading phase. During the writing phase, plain text and keys are written to the corresponding memory locations. In the reading phase, encrypted data are copied out to the output of encryption function. With FPGA acceleration, the cryptographic functions only consist of mapping pointers to the memory locations and memory copying.

In the experiments for evaluation, we use our own application

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

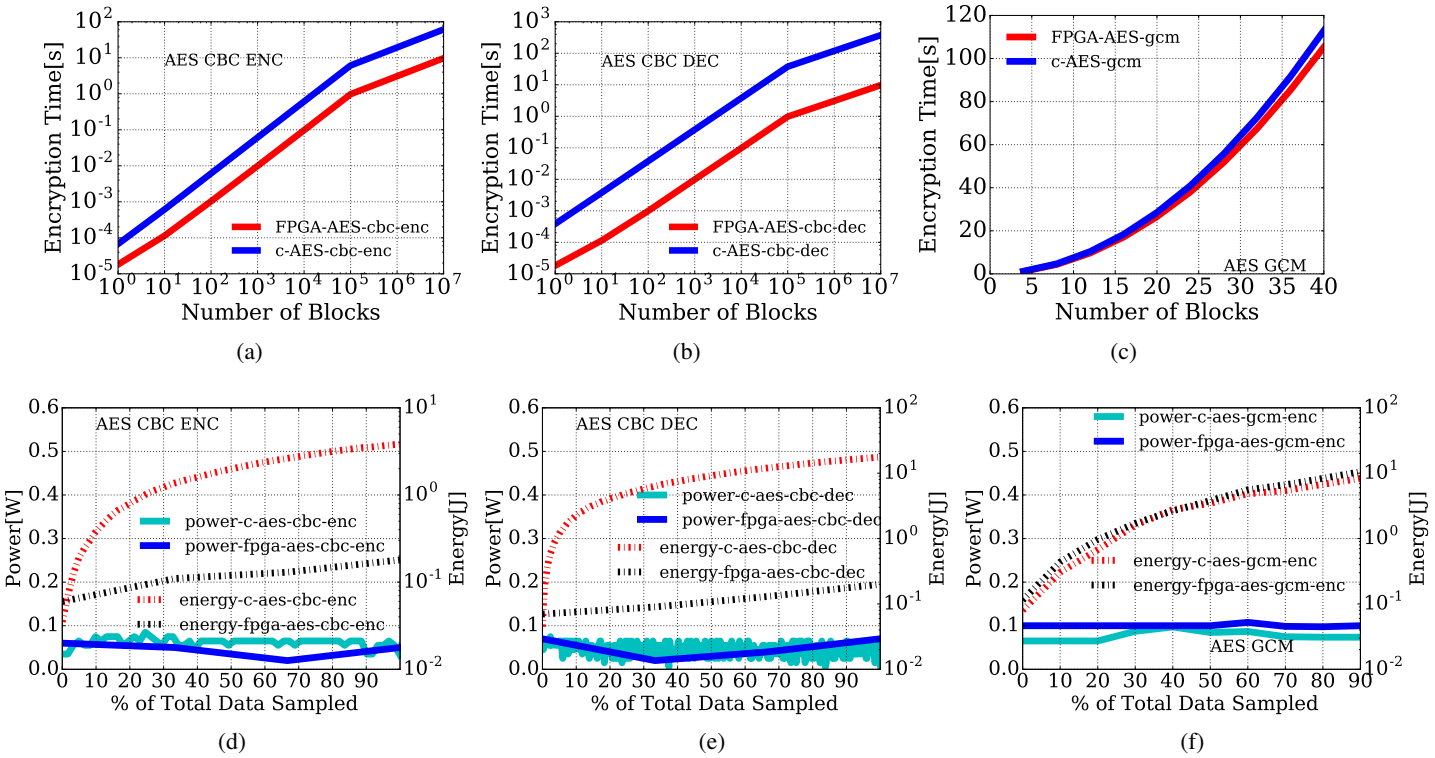

Figure 3: Comparisons between C implementation (software), and FPGA implementation (hardware) of AES block cipher modes, for both CBC and GCM. In all the hardware implementations, AES core engine operations use FPGA, while block cipher modes use C implementation. Thus, we still can see a significant performance boost and energy savings in CBC modes, while much fewer benefits in GCM modes. In CBC mode, we encrypt and decrypt data ranging from 1 block to 10^7 blocks. In GCM mode, we encrypt data ranging from 4 to 40 blocks. (a), (b) and (c) show time comparisons. CBC encryption is 6.3× faster, CBC decryption is 38.6× and GCM is 1.1× faster on FPGA. (d), (e) and (f) show power consumption and energy comparisons. CBC encryption has 33.2× energy reduction, CBC decryption has 154.8× energy reduction and GCM has 1.2× energy reduction.

programs to load data into memory and to encrypt/decrypt /hash for repeated measurements. We measure AES operations in Electronic Code Book mode (ECB) and cipher block chaining (CBC). We use the same crypto-engine core for both modes in cryptography. In cipher block chaining, current block operations depend on previous block results, known as the read-after-write dependency between two operations. As a result, the AES crypto-engine can not be implemented in a streaming fashion, where plain text data are continuously fed into crypto-engines and encrypted data are produced at the same time. We integrate our implementation of the AES crypto function into OpenSSL to enable other block cipher modes. We use AES block cipher as a building block to construct a symmetric encryption scheme where the hardware accelerated block cipher is used in combination with a software implementation of the block cipher modes. In order to achieve system reconfiguration, we synthesize all the algorithms and write them to bitstream on the memory storage (SD card on Zedboard). Our hardware implementation embeds well into existing libraries so that users can easily utilize our implementations for their designs. All that is needed to upgrade the cryptographic libraries is to replace the bitstream file with the new bitstream file through secured channel on the network. Whenever a re-configuration is requested, reconfiguration of the FPGA can be achieved by sending the bitstream to the configuration device port. After the generated crypto-engine bitstream are copied to FPGA, the Zedboard system takes less than a second to reconfigure. In this way, the system can get updated bitstream and map them to FPGA accordingly.

For the power and energy analysis, we first measure the static power consumption of the entire Zedboard platform and then record the power consumption for each individual algorithm. We make sure that fluctuations have been averaged out through a large number of experiments. In each algorithm measurement, we measure a certain amount of data cryptographic operations that are larger than average web page size (2MB) [29]. We record the power consumption of 16^6 256-bit blocks for SHA, 16^6 64-bit blocks for DES, 10^5 1K-bit blocks for RSA and 10^7 128-bit blocks for AES for 10 different runs at a sample rate of 1 sample per second. After the sampling, we average each data point among 10 runs of experiments to get the average power for each sampling interval. We deduct the static system power consumption of both ARM, FPGA, and peripheral devices to get the dynamic power consumption for the measurements with FPGA acceleration. For the pure software implementation measurements, in addition to the deduction of overall static power consumption, we also deduct the static power consumption of FPGA. Since the software implementation takes much longer time to finish, we use percentages of processed data to normalize the total time for energy comparisons.

5. EVALUATION

In this section, we show the performance and energy based comparison between software implementation and hardware FPGA implementation of AES, RSA, DES, and SHA256. We perform the analysis for both encryption and decryption for all the symmetric and asymmetric cryptographic algorithm. We

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

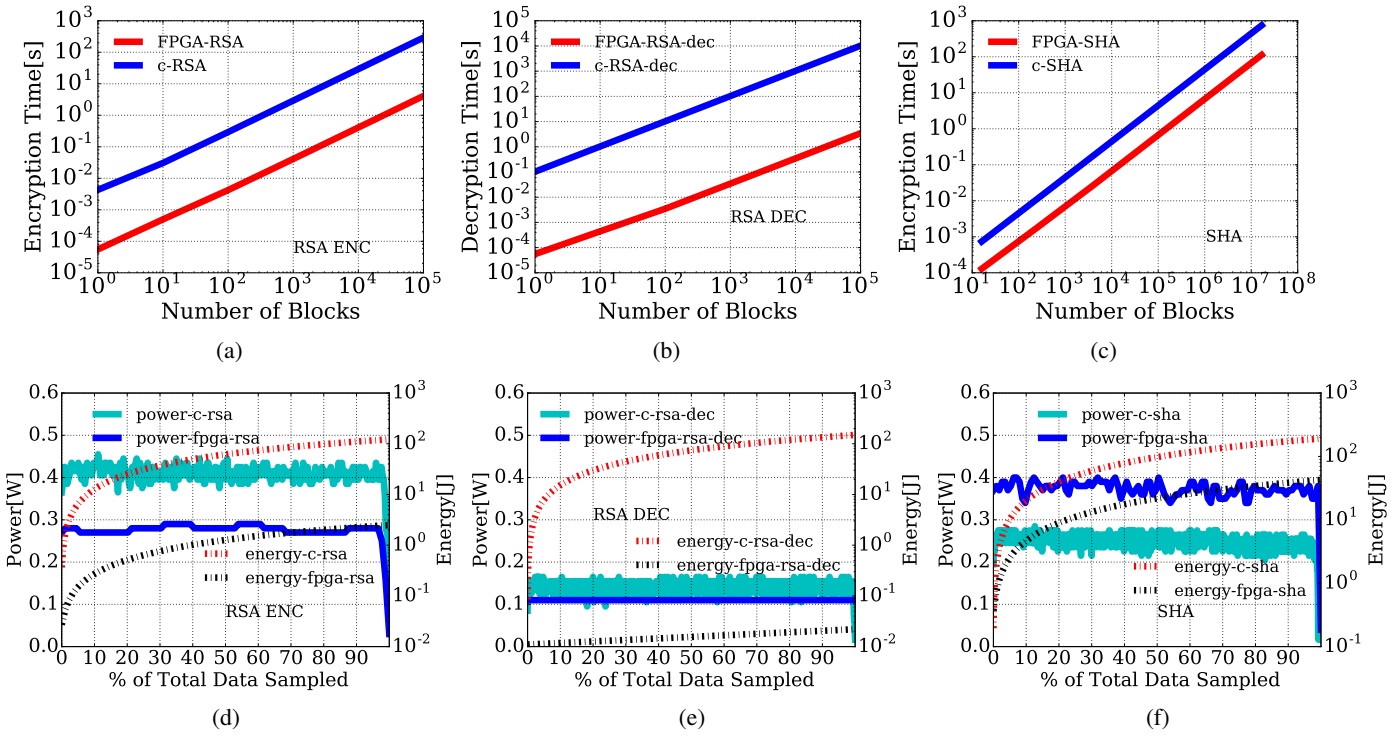

Figure 4: Comparisons between C implementation (software), and FPGA implementation (hardware) of RSA in encryption (ENC) and decryption (DEC) and SHA. We encrypt and decrypt data ranging from 1 block to 10^5 blocks in RSA. (a) and (b) show time comparisons. Encryption is 71.2× faster and decryption is 2983.1× faster. (d) and (e) show power consumption and energy comparisons. Encryption has 6.5× energy reduction and decryption has 4033.0× energy reduction. We do hash ranging from 16 blocks to 16^6 blocks in SHA. (c) shows time comparisons. (f) shows power consumption and energy comparisons. We can see a performance boost, energy savings of hardware implementation in SHA.It shows 6.6× faster in time, 4.6× reduction in energy,

present measurements results of AES Cipher Block Chaining (CBC) and AES Galois/Counter Mode (GCM) mode as the examples for block cipher modes. AES CBC uses the bitwise exclusive-or between the previous encrypted block and current plain text block as the input for the next block in order to increase pseudo-randomness in the cipher text. AES GCM, in addition, has the authentication of data along with encryption in block cipher modes.

Table 1, shows a summary of Look-Up Tables (LUTs) utilizations after crypto-engines are mapped to the FPGA. LUTs are gate level units that encode any boolean expression by modeling such functions in truth tables. The number of LUTs tells us the amount of resources that need to be used for logic on FPGA.

	AES	RSA	SHA	DES
Number of Gates	41,427	18,687	29,650	1,275
LUTs Utilization	77.87%	35.11%	55.71%	2.4%

Table 1: This table shows the LUTs utilizations in FPGA of different crypto-engines.

In the measurement tests, we process data ranging from 16 bytes to $268,435,456 (16^7)$ bytes for SHA256 and DES blocks, from 1 byte to 1,600,000 bytes for RSA testbench, and from 1 byte to 160,000,000 bytes for AES. The Zedboard with the Linaro OS running the system consumes 4.0 watts on average, which is the static power consumption (power consumption of the system with running no application program but operating system). The programmable components consume less than 0.035 watts [30] static power. In the following subsections, we discuss how much the FPGA implementations have improved on performance and energy in symmetric cryptography (§ 5.1), asymmetric cryptography (§ 5.2) and hash functions (§ 5.3), compared to software.

5.1 Symmetric Cryptography

In DES implementation, the hardware implementation is 1.9× in encryption and 1.6× in decryption faster than software (see Figure 2). Though the hardware implementation does not show a significant performance improvement, the FPGA consumes much less power. The energy consumption, is 3.9× in encryption and 1.9× in decryption lower for the hardware implementation (see Figure 2).

In the AES implementation, the VHDL code breaks the AES into 10 rounds of permutation and substitution of SubBytes and inverse SubBytes operations. We expand rounds of operations to achieve maximum throughput. As we can see from Figure 2, the AES encryption has 18.8× faster and AES decryption has 116.3× faster. Power consumption in both cases is almost the same (see Figure 2). Since FPGA implementation is much faster than the software implementation, the energy consumption of FPGA is 13.9× less in encryption and 6.0× less in decryption less than the software implementation. We also evaluate the block cipher modes in a similar fashion (see Figure 3). Block cipher modes are all implemented in software, while the crypto-engines are implemented in hardware. AES CBC mode shows 6.2× faster for encryption and 38.6× faster for decryption. It also shows a 33.1× energy reduction for encryption and 154.1× energy-reduction for decryption. GCM mode exhibit 1.1× performance and 1.2× energy savings. Compared to CBC mode,

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

the additional computation in software makes GCM have lower performance boost and energy savings.

5.2 Asymmetric Cryptography

The software implementation of RSA is much slower than its hardware implementation, since RSA requires a large number of multiplications that can be performed in parallel in hardware. With a key length of 1024 bit, the FPGA shows a performance improvement of 71.2× in encryption and 2983.1× in decryption compared to the software implementation (see Figure 4). Figure 4 shows that the power consumption for RSA is less than the software implementation. The energy cost in FPGA is 6.5× lower in encryption and 4033× lower in decryption compared to software implementation.

5.3 Hash Function

In the case of SHA256, the FPGA has a higher power consumption (see Figure 4) compared to software implementation, however, the performance is better by a factor of 6.6× (see Figure 4). As a result, the energy consumption is 4.6× less in the FPGA implementation.

5.4 Evaluation Summary

The evaluation results from Section 5.1, 5.2, and 5.3 show improvements of implementing cryptographic algorithms on FPGA substrate in terms of performance boost, and energy savings. Although in some of the cryptographic algorithms, the FPGA implementations present higher power consumption than software implementations from time to time, due to its high performance in computation, all the FPGA implementations in our evaluation section have much lower the energy cost compared to software implementation.

6. CONCLUSION

Cryptographic operations are one of the most power-hungry computations in today's systems. Our paper evaluates the mapping of crypto-engines to the FPGA of Zedboard, as the FPGA can enable more energy-efficient crypto operations than in software as well as the option to upgrade the cryptographic algorithms through re-configurations. Performance boost, and energy savings in FPGA implementations compared to software implementations range from 1.5× to 2983×, and from 1.8× to 4033×, respectively across a variety of cryptographic algorithms.

7. FUTURE WORK

We will work on performance and energy cost comparisons on other off-the-shelf SoC platforms. In addition, we will have comparisons with anti-DPA implementations of the algorithms presented in this paper, and comparisons of ECC DH (Elliptic Curve Diffie Hellman) with and without a reduction algorithm.

8. REFERENCES

[1] "Internet of everything." `https://systemx.stanford.edu/sites/default/files/uploads/Archive/2014/Nov/11/Lee.pdf`. Accessed: 2016-02-10.

[2] B. Miller and D. Rowe, "A survey scada of and critical infrastructure incidents," in *Annual Conference on Research in Information Technology*, 2012.

[3] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," in *BlackHat USA*, 2014.

[4] A. Illera and J. Vidal, "Lights off! the darkness of the smart meters," in *BlackHat Europe*, 2014.

[5] G. Hernandez, O. Arias, D. Buentello, and Y. Jin, "Smart nest thermostat: A smart spy in your home," in *Black Hat USA*, 2014.

[6] S. Zonouz, J. Rrushi, and S. McLaughlin, "Detecting industrial control malware using automated plc code analytics," in *IEEE Symposium on Security and Privacy*, 2014.

[7] "Cisco visual networking index: Global mobile data traffic forecast update, 2015-2020 white paper - cisco." `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html`. Accessed: 07-24-2016.

[8] C. P. Mayer, "Security and privacy challenges in the internet of things," in *Electronic Communications of the European Association of Software Science and Technology*, 2009.

[9] A. Klinefelter, N. E. Roberts, Y. Shakhsheer, P. Gonzalez, A. Shrivastava, A. Roy, K. Craig, M. Faisal, J. Boley, S. Oh, *et al.*, "21.3 a 6.45μw self-powered iot soc with integrated energy-harvesting power management and ulp asymmetric radios," in *IEEE International Solid-State Circuits Conference*, 2015.

[10] "Armv8 technology preview." `http://www.arm.com/files/downloads/ARMv8_Architecture.pdf`. Accessed: 2016-02-09.

[11] "Intel advanced encryption standard instructions (aes-ni)." `https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni/`. Accessed: 2016-02-09.

[12] "Armv8 technology preview." `http://www.atmel.com/Images/doc8106.pdf`. Accessed: 2016-02-09.

[13] "Openssl." `https://www.openssl.org/`. Accessed: 07-26-2016.

[14] "opencores." `http://opencores.org/projects`. Accessed: 2016-02-10.

[15] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat, "Compact and efficient encryption/decryption module for fpga implementation of the aes rijndael very well suited for small embedded applications," in *International Conference on Information Technology: Coding and Computing*, 2004.

[16] T. Good and M. Benaissa, "Very small fpga application-specific instruction processor for aes," in *IEEE Transactions on Circuits and Systems*, 2006.

[17] P. Hamalainen, T. Alho, M. Hannikainen, and T. D. Hamalainen, "Design and implementation of low-area and low-power aes encryption hardware core," in *Euromicro Conference on Digital System Design*, 2006.

[18] P. Bulens, F.-X. Standaert, J.-J. Quisquater, P. Pellegrin, and G. Rouvroy, "Implementation of the aes-128 on virtex-5 fpgas," in *International Conference on Cryptology in Africa*, 2008.

[19] T. Good and M. Benaissa, "Aes on fpga from the fastest to the smallest," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 2005.

[20] S.-S. Wang and W.-S. Ni, "An efficient fpga implementation of the advanced encryption standard algorithm," in *IEEE International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future*, 2012.

[21] J. M. Granado-Criado, M. A. Vega-Rodríguez, J. M. Sánchez-Pérez, and J. A. Gómez-Pulido, "A new methodology to implement the aes algorithm using partial and dynamic reconfiguration," in *Integration, the VLSI journal*, 2010.

[22] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen, "A side-channel analysis resistant description of the aes s-box," in *International Workshop on Fast Software Encryption*, 2005.

[23] S. Shah, R. Velegalati, J.-P. Kaps, and D. Hwang, "Investigation of dpa resistance of block rams in cryptographic implementations on fpgas," in *International Conference on Reconfigurable Computing and Field-Programmable Gate Arrays*, 2010.

[24] X. Fan, G. Gong, K. Lauffenburger, and T. Hicks, "Fpga implementations of the hummingbird cryptographic algorithm," in *IEEE International Symposium on Hardware-Oriented Security and Trust*, 2010.

[25] N. Pramstaller, C. Rechberger, and V. Rijmen, "A compact fpga implementation of the hash function whirlpool," in *International Symposium on Field Programmable Gate Arrays*, 2006.

[26] M.-J. O. Saarinen, "Simple aead hardware interface (sæhi) in a soc: Implementing an on-chip keyak/whirlbob coprocessor," in *International Workshop on Trustworthy Embedded Devices*, 2014.

[27] M.-J. O. Saarinen and B. B. Brumley, "Lighter, faster, and constant-time: Whirlbob, the whirlpool variant of stribob.," in *International Association for Cryptologic Research Cryptology ePrint Archive*, 2014.

[28] "Linaro." `http://www.linaro.org/`. Accessed: 08-04-2016.

[29] "Page bloat update: The average web page is more than 2 mb in size." `https://www.soasta.com/blog/page-bloat-average-web-page-2-mb/`. Accessed on 04-07-2017.

[30] M. Hosseinabady and J. L. Nunez-Yanez, "Run-time power gating in hybrid arm-fpga devices," in *International Conference on Field Programmable Logic and Applications*, 2014.

978-1-5386-3472-1/17/$31.00 ©2017 IEEE