FEARED: FPGA-based Encryption for At Rest Embedded Data

Scott Smith, Meghna Mandava, Hsin-De Lee
scottcs2,meghnam4,hsindel2@illinois.edu

Abstract

Physical attacks are one type of attack that can be performed on a computer system. Like the name suggests, these attacks require gaining physical access to a device. Once gained, the attacks often take the form of directly reading or modifying data inside of memory and storage devices. These types of attacks have prompted hardware designers to create sophisticated data protection schemes for data in memory and in storage. However, these protection schemes are typically designed for powerful CPUs used in cloud or data center applications. Data at rest encryption is one commonly used method to protect against physical attacks on storage devices. This paper discusses the design and implementation of a hardware-accelerated data at rest protection scheme for embedded systems that employ SD cards. We hardware accelerate AES-XTS encryption and decryption. The design is evaluated on a small embedded device featuring CPU cores and an embedded FPGA. We demonstrate our design is realistic to employ on moderately powerful embedded devices and can achieve significant speedup over pure software implementations of the algorithm.

1 Introduction

Security is becoming an increasingly important concern in computing. This is true across a myriad of application domains ranging from embedded systems to cloud-scale systems. VPNs, firewalls, and antivirus software are common security approaches that people employ to help secure their systems from outside attack. However, such approaches are not capable of defending against physical attacks. Other techniques are necessary to protect sensitive data once someone gets hold of a device.

Physical attacks require physical access to a device. Physically preventing someone from accessing a machine is the easiest method for preventing such attacks. However, preventing untrustworthy sources from accessing a physical machine is not always possible. This is typically discussed in the context of cloud providers and data centers, where many employees may potentially have access to the hardware and the client using these services has no way of restricting or monitoring who has access.

There is also significant potential for physical attacks on embedded systems and internet of things (IoT) devices. Embedded systems, by their nature, are often running in the field without active supervision and physical protection. Many of these devices gather sensitive user data. Gaining physical access to an unprotected device could enable a hacker to read or modify data or code. Protecting against such attacks on embedded, low-cost, and low-power devices is important.

Without sufficient protection, a hacker can gain a great deal of information by probing the memory bus or by reading data from storage devices. For example, a hacker can access data stored on a DRAM chip if they rapidly cool the DRAM after power is removed from the device [1]. Accessing data held in non-volatile storage is typically even easier. For example, data stored onto an SD card or USB flash drive could simply be stolen by unplugging the device from the embedded system.

The ability to read (and potentially modify) data stored in memory devices by an unwanted party has prompted the development of memory-level encryption. This is also known as protecting data in use. For example, in the last couple of years Intel has released Intel Software Guard Extensions (SGX) which is capable of encrypting memory [2]. This and other systems provide some protection from unwanted eyes.

Storage devices are often easier sources to steal data from due to being nonvolatile. Protecting data in storage devices is commonly called data at rest protection (DARP). Many products have been developed which enable encryption of storage devices. Encryption of the data inside of storage can be performed by an individual application, the operating system, the storage device, or the network. Intel, HP, and more offer storage devices which support encryption with little overhead [3]. Self-encrypting SD cards have also been developed, but don't seem to be widely available or have very good performance. For example, ATP Electronics offers an SD card with built-in encryption support but it has a measly read performance of 10.35 MB/s [4].

The importance of security in many embedded applications clearly implies that supporting encrypted storage is very important. Unfortunately, the the lack of a large pool of high-performance, encryption-supporting SD cards to choose from currently makes adding encrypted storage to an embedded project sub-optimal. Field programmable gate arrays (FPGA) offer an enticing solution to this problem due to their flexibility and processing capabilities. Although FPGAs are flexible and powerful, they are often considered difficult to program. These devices are typically programmed at the register transfer level (RTL). Recently, more promising methods for programming FPGA devices have been developed. For example, Xilinx High Level Synthesis (HLS) is a C-based flow for programming FPGA devices [5]. In the last year, a new programming flow for FPGAs has been proposed called PyLog [6]. PyLog enables FPGAs to be programmed via Python which is a very friendly, high-level programming language. Combining the processing power and flexibility of FPGAs with high-level languages like Python makes developing accelerators simpler and easier. They enable developers to focus on the algorithms and spend less time worrying about low-level hardware optimizations.

In this paper we propose FEARED: an **F**PGA-based **e**ncryption system for **a**t **r**est **e**mbedded **d**ata on SD cards. Our design is optimized for medium-end to low-end embedded systems which include FPGAs. Our low-area implementation makes it perfect for embedded devices which need encrypted storage devices but don't want to sacrifice on performance. The hardware design is optimized to match the SD card performance to minimize wasted area. Our design is open-source and developed with PyLog.

2 Related Work

As discussed in the previous section, the importance of data-at-rest encryption has been stressed in recent years. The Advanced Encryption Standard (AES) is a block cipher which has been widely adopted for the encryption of digital data. AES-XTS is a mode of AES which extends the XOR–Encrypt–XOR (XEX) block cipher with a method called ciphertext stealing (XTS). Ciphertext stealing allows for encrypting plaintext which is not divisible by the block size without changing the length of the message. This is preferred when encrypting storage devices [7]. This is because AES-XTS is a tweakable cipher, which makes it allows each block of input data to be encrypted differently in an efficient manner. This can be done using the block number, without the need for an initialization vector. AES-XTS is also more difficult for an attacker to tamper with then a regular block cipher like AES-CBC (cipher-block chaining). Both modes are vulnerable to an attacker corrupting a block, but AES-XTS prevents attacks which tamper with a specific bit within the block [8]. As such, the XTS mode is regarded as best method for disk encryption currently available. AES-XTS is approved by the National Institute of Standards and Technology (NIST) to be used for data-at-rest encryption because of its security properties [9].

AES-XTS is a block cipher mode. This means that it uses a block cipher, specifically AES-CBC, as a component in its intermediate steps in addition to the operations it performs to achieve the above properties. There are many implementations in literature of AES-CBC which show its potential to be optimized for both latency and area on various devices [10, 11]. However, there are fewer implementations of AES-XTS and even fewer that are done on chips that are appropriate for embedded systems. Since AES-XTS has a high-degree of parallelism, it can be accelerated on an FPGA [12].

Past works have explored FPGA implementations of the algorithm. Table 1 shows the boards they were implemented on, the throughput, resource utilization on the implemented board, the predicted resource utilization on the PYNQ board, and the amount that the throughput overshoots the desired throughput for an SD card with 100 MB/s reads (10 MB/s write). Most of these implementations were done on the Xilinx Virtex-5, which is a very powerful FPGA with a significant amount of on-chip memory and digital signal processing (DSP) blocks. The implementations used around 6k-7k logic slices. The Virtex-7 board, which is the latest in the series, costs around $5000 [13, 14]. Clearly, these implementations have not been evaluated with embedded application uses in mind. Another implementation on an Altera Cyclone II achieved a throughput of 20 Gbps using 12k logic elements [15]. The Cyclone II is perhaps more suited to embedded applications, but is still quite large with 68,416 logic elements. The latest board in the Cyclone series can cost upwards of $1000 [16]. A smaller FPGA board like the PYNQ-Z2 has only 13k logic slices and can be purchased for $200. This is much closer to the sort of chip that would be used in embedded applications [17]. Past implementations would take up a significant portion of the PYNQ-Z2 resources. The estimated resource demands range from anywhere between 20%-50% of the board. Most embedded applications are responsible for handling a myriad of tasks. In many cases, an FPGA would need to fit more than just hardware accelerators for encryption and decryption of stored data. Because data-at-rest encryption is typically focused on data-center drives, AES-XTS implementations on smaller and lower-power devices have not been generally explored. Our implementation uses significantly less area making it more applicable for small embedded devices.

Table 1 also highlights the performance overshoot of the prior implementations. These accelerators were optimized for systems with much higher throughput needs than an SD card. As a result, naively implementing these accelerators directly on the PYNQ board would result in severe resource underutilization. Our design matches the throughput of a 100 MB/s read, 10 MB/s SD card and therefore is able to significantly reduce the amount of resources needed by the accelerators.

AES-XTS is also used in many industry products for on-disk storage for cloud and other large-scale data [8, 21, 22]. However, these products are not intended for use in embedded applications. Most full-disk encryption schemes require extensive setup and do not match the constraints embedded devices face. As previously discussed, embedded devices are prone to physical attacks because of the nature of their use. Thus, there is a need for efficient, self-encrypting storage for small, embedded devices. Spyrus and Google have both announced SD cards with AES encryption, but these are not

Implementation	Technique	Board	Throughput (Gbps)	Overshoot	Board Resource Utilization	Estimated PYNQ Resource Utilization
[18]	Encryption / Decryption	Virtex 5	28.7	3587x	13%	51%
[19]	Encryption	Virtex 5	35.8	44750x	8%	32%
[19]	Decryption	Virtex 5	30	3750	10%	38%
[20]	Encryption / Decryption	Virtex 5	5.7	712x	6%	24%
[15]	Encryption	Virtex 5	38.077	47500x	8%	30%
[15]	Encryption	Cyclone II	19.56	24450x	36%	-

Table 1. Previous FPGA-based hardware accelerators for AES-XTS

currently available for purchase and seem to be marketed toward PC use [23, 24]. ATP recently released a self-encrypting SD card but its low read/write speeds make it unideal for embedded applications [4]. Due to the technology, area, and power constraints on SD cards, we believe that accelerating encryption on an FPGA rather than on the SD card itself could result in better performance. Since the goal is to sustain the bandwidth of the SD card, the previous works which accelerate AES-XTS are excessive with respect to throughput. Instead, our encryption and decryption implementation aims to be more efficient than the previous works, so that it is suitable for use with embedded applications that have strict power and area constraints.

3 Proposed Idea

Due to the rapid pace of technological development, people need powerful platforms to execute a huge amount of computation to meet the requirement of applications. FPGA is a good platform for doing hardware acceleration. However, FPGA design is time-consuming compared to software design.

To deal with this problem, Xilinx provides a High-Level Synthesis (HLS) compiler that enables C, C++, and SystemC programs to be directly targeted into Xilinx devices without the need to manually write RTL. Vivado HLS is widely reviewed to increase developer productivity. Vivado HLS supports C++ classes, templates, functions and operator overloading. Nevertheless, users need to manually optimize their code with HLS pragmas (compiler directives) to give the HLS compiler hints on parallelism and desirable synthesis approaches. The performance highly depends on the designer's experience in adding pragmas in the optimal places. If the application is complicated, it will lead to long source code which is hard to maintain and read.

In order to overcome the shortfalls of HLS, we will use PyLog to implement our design. It allows us to write the code in Python language which is a widely-used programming language for software engineers. PyLog automatically optimizes the code and generates the C code for the Vivado HLS tool. PyLog doesn't require (but does allow) hardware optimization pragmas to be placed manually. Thus, we can focus on implementing the algorithm to optimize the design and spend less time considering specific optimization strategies. Additionally, compared to C, PyLog reduces the number of lines of code that need to be written. This makes it easier to rapidly optimize and iterate the code.

As discussed in the previous sections, we want to accelerate the process of encryption and decryption for embedded devices which use SD cards. There are many embedded systems that must meet various timing and other constraints that are imposed on them by the real-time behavior of the external world to which it is interfaced. For example, vehicle systems for automobiles, satellite communications, multimedia systems that provide text, graphic, audio, and video interfaces, and so on and so forth. PyLog is a powerful tool that can allow us to accelerate the AES-XTS algorithm to meet application constraints.

We will implement the AES-XTS encryption and decryption algorithm accelerators on a PYNQ board. The accelerator will be developed in PyLog. In order to demonstrate the algorithm's correctness and performance, we will create a trivial example which utilizes both encryption and decryption. First, we will start with some plaintext bitmap images on an SD card. We will then transfer the data to DRAM whereupon the AES-XTS encryption accelerator will encrypt the data. After finishing the algorithm, the encrypted data will be stored back on the SD card. Once all the data is encrypted, we will also read the data back from the SD card and then decrypt it. A timer will be used to calculate the time for both encryption and decryption. We will compare the latency and system throughput to the pure software implementation.

Our goal is to optimize the design to the point that the reads and writes from the SD card do not see any throughput loss, while keeping the area of the design small and minimizing the latency overhead.

4 Expected Results

We intend to implement an AES-XTS encryption and decryption scheme on the PYNQ-Z2 board that protects SD card data used by an embedded application. We predict that our implementation will see better performance than self-encrypting SD cards which are currently on the the market. The ATP self-encrypting SD card has 10 MB/s reads and 5 MB/s writes. Modern SD cards typically see at least 10 MB/s writes and much faster reads. We believe that encryption and decryption are not well-suited to be done on the SD card itself. We expect to be able to achieve line-rate encryption and decryption of SD card data with little latency overhead and significantly less area than prior implementations of AES-XTS. Our implementation will be less powerful than the Vertex and Cyclone implementations, but will better match the performance requirements of an SD card.

5 Timeline

- **Week 1 (11/14-11/20):**
  - Solidify the project idea and perform a literature review.
  - Create a draft for the project proposal.
  - Discuss the project idea with Professor Chen.
  - Finalize the project proposal and submit it.
- **Week 2 (11/21-11/27):**
  - Write a simple accelerator in PyLog and use it on the Pynq board.
  - Learn the AES-XTS encryption and decryption algorithms and implement them in software.
- **Week 3 (11/28-12/4):**
  - Implement the AES-XTS encryption and decryption algorithms in PyLog.
  - Determine how to interface the PyLog accelerator with the SD card.
- **Week 4 (12/5-12/11):**
  - Perform final optimizations to the algorithms to maximize performance under our design constraints.
  - Begin gathering data for use in our final presentation and report.
- **Week 5 (12/12-12/20):**
  - Create a final presentation and deliver the presentation to the class.
  - Create a final report and submit it.

6 Division of Labor

- **Scott Smith:** Scott will focus on learning how to use PyLog and implementing the encryption/decryption accelerators in PyLog. He will also spend time optimizing the code for our particular use case and gathering performance numbers.
- **Meghna Mandava:** Meghna will focus on understanding the AES-XTS encryption/decryption algorithms and then will implement them in software. She will also help translate the code into a format amenable to acceleration using PyLog.
- **Hsin-De Lee:** Hsin-De will focus on creating a simple application to demonstrate the effectiveness of our encryption/decryption accelerators. He will also help integrate the PyLog-based accelerator onto the PYNQ board. He will figure out the interfaces between the SD card, processor cores, and accelerators and write the host software.

References

[1] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: Cold-boot attacks on encryption keys," *Commun. ACM*, vol. 52, p. 91–98, May 2009.

[2] S. Gueron, "A memory encryption engine suitable for general purpose processors." Cryptology ePrint Archive, Report 2016/204, 2016. https://ia.cr/2016/204.

[3] "Ssif solutions guide for data-at-rest," 2009.

[4] "Securstor aes encryption microsd cards."

[5] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for fpgas: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[6] S. Huang, K. Wu, H. Jeong, C. Wang, D. Chen, and W.-m. Hwu, "Pylog: An algorithm-centric python-based fpga programming and synthesis flow," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '21, (New York, NY, USA), p. 227–228, Association for Computing Machinery, 2021.

[7] "Ieee standard for cryptographic protection of data on block-oriented storage devices," *IEEE Std 1619-2007*, pp. 1–40, 2008.

[8] O. Choudary, F. Gröbert, and J. Metz, "Infiltrate the vault: Security analysis and decryption of lion full disk encryption," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 374, 2012.

[9] U. D. of Commerce and M. J. Dworkin, *Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices.* National Institute of Standards and Technology, January 2010.

[10] T. Good and M. Benaissa, "Aes on fpga from the fastest to the smallest," in *Cryptographic Hardware and Embedded Systems – CHES 2005* (J. R. Rao and B. Sunar, eds.), (Berlin, Heidelberg), pp. 427–440, Springer Berlin Heidelberg, 2005.

[11] K. Gaj and P. Chodowiec, *FPGA and ASIC Implementations of AES*, pp. 235–294. Boston, MA: Springer US, 2009.

[12] M. A. Alomari, K. Samsudin, and A. R. Ramli, "A parallel xts encryption mode of operation," in *2009 IEEE Student Conference on Research and Development (SCOReD)*, pp. 172–175, 2009.

[13] Xilinx, *Virtex-5 Family Overview*, 2015. v5.1.

[14] Xilinx, *VC707 Evaluation Board for the Virtex-7 FPGA*, 2019. v1.8.

[15] M. Elmoghany, M. Diab, M. Kassem, M. Khairallah, O. El Shahat, and W. Sharkasy, "Fpga implementation of high speed xts-aes for data storage devices," in *2011 International Conference for Internet Technology and Secured Transactions*, pp. 25–28, 2011.

[16] Altera, *Cyclone II Device Handbook, Volume 1*, 2008.

[17] TUL, *TUL PYNQ™-Z2 board*, 2020.

[18] Y. Wang, A. Kumar, and Y. Ha, "Fpga-based high throughput xts-aes encryption/decryption for storage area network," in *2014 International Conference on Field-Programmable Technology (FPT)*, pp. 268–271, 2014.

[19] A. Shakil and M. Naseem, "Efficient aes-xts pipelined implementation on fpga," *Sir Syed Research Journal of Engineering & Technology*, vol. 1, p. 6, 12 2014.

[20] S. Ahmed, K. Samsudin, A. R. Ramli, and F. Z. Rokhani, "Advanced encryption standard-xts implementation in field programmable gate array hardware," *Security and Communication Networks*, vol. 8, no. 3, pp. 516–522, 2015.

[21] Greg-Lindsay, "What's new in windows 10, versions 1507 and 1511 for it pros," Mar 2021.

[22] "Aes-xts block cipher mode is used in kingston's best encrypted usb flash drives," Dec 2018.

[23] "Rosetta® microsdhctm card."

[24] D. Etherington, "Google's project vault is a secure computing environment on a micro sd card, for any platform," May 2015.