Securing the Future of 5G Smart Dust: Optimizing Cryptographic Algorithms for Ultra-Low SWaP Energy-Harvesting Devices

Zeezoo Ryu

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Application

Matthew Hicks, Chair
Wenjing Lou
Peng Gao

May 8, 2023
Blacksburg, Virginia

Keywords: Embedded Cryptography, AES, SHA, ECC, CMAC, HMAC, RSA, Energy Harvesting, Execution Time, Binary Size

Securing the Future of 5G Smart Dust: Optimizing Cryptographic Algorithms for Ultra-Low SWaP Energy-Harvesting Devices

Zeezoo Ryu

(ABSTRACT)

While 5G energy harvesting makes 5G smart dust possible, stretching computation across power cycles affects cryptographic algorithms. This effect may lead to new security issues that make the system vulnerable to adversary attacks. Therefore, security measures are needed to protect data at rest and in transit across the network. In this paper, we identify the security requirements of existing 5G networks and the best-of-breed cryptographic algorithms for ultra-low SWaP devices in an energy harvesting context. To do this, we quantify the performance vs. energy tradespace, investigate the device features that impact the tradespace the most, and assess the security impact when the attacker has access to intermediate results. Our open-source energy-harvesting-tolerant versions of the cryptographic algorithms provide algorithm and device recommendations and ultra-low SWaP energy-harvesting-device-optimized versions of the cryptographic algorithms.

Securing the Future of 5G Smart Dust: Optimizing Cryptographic Algorithms for Ultra-Low SWaP Energy-Harvesting Devices

Zeezoo Ryu

(GENERAL AUDIENCE ABSTRACT)

Smart dust is a network of tiny and energy-efficient devices that can gather data from the environment using various sensors, such as temperature, pressure, and humidity sensors. These devices are extremely small, often as small as a grain of sand or smaller, and have numerous applications, including environmental monitoring, structural health monitoring, and military surveillance. One of the main challenges of smart dust is its small size and limited energy resources, making it challenging to power and process the collected data. However, advancements in energy harvesting and low-power computing are being developed to overcome these challenges. In the case of 5G, energy harvesting technologies can be used to power small sensors and devices that are part of the 5G network, such as the Internet of Things (IoT) devices. Examples of IoT devices are wearable fitness trackers, smart thermostats, security cameras, home automation systems, and industrial sensors. Since 5G energy harvesting impacts the daily lives of people using the relevant devices, our research seeks to find out what kind of measures are necessary to guarantee their security.

Dedication

I dedicate this to my parents, sister, and my husband, who have always been there for me when I needed them.

Acknowledgments

I would like to express my sincere gratitude to my advisor, Dr. Matthew Hicks, for his guidance, support, and patience throughout my research. His insightful feedback and constructive criticism have helped me to grow as a researcher and to develop a deeper understanding of the field. I am also grateful to my lab mates of the FoRTE (Fostering Research for Targeted Excellence) Research Group for their support and feedback throughout my time at Virginia Tech. Finally, I would like to thank my family and friends for their unwavering support and for believing in me throughout my academic journey.

Contents

List of Figures

List of Tables

List of Abbreviations

AES Advanced Encryption Standard

CMAC Cipher Block Chaining - Message Authentication Code

ECC Elliptic-curve Cryptography

ECDH Elliptic-curve Diffie-Hellman

ECDSA Elliptic Curve Digital Signature Algorithm

ECIES Elliptic Curve Integrated Encryption Scheme

HMAC Hash-Based Message Authentication Code

RSA Rivest-Shamir-Adleman

SHA Secure Hash Algorithm

Chapter 1

Introduction

Smart dust is a type of wireless sensor network that consists of tiny, self-powered devices that can be used to collect data about the environment. Smart dust devices are typically no larger than a grain of sand, and they can be used to monitor a wide variety of environmental factors, such as temperature, humidity, pressure, and light. This technology requires advancements in miniaturization, integration, and energy management. Designers can use microelectromechanical systems (MEMS) to build sensors that are only a few micrometers in size. MEMS can also be used to build optical communication components and power supplies, while microelectronics provide increased functionality in smaller areas with lower energy consumption. Smart dust has the potential to be used in a variety of applications, including environmental monitoring, disaster response, and military surveillance [1].

Batteries make 5G smart dust impossible because it is large, heavy, expensive, high maintenance, and flammable. In this case, since 5G provides energy and communication, we can find an opportunity to harvest energy from the 5G spectrum by converting energy to electrical power for use in the 5G network. With the ever-increasing demand for faster mobile communication and utilization of increased data, 5G mobile networks emerged. While 5G networks provide a higher data transfer rate and better coverage than their predecessors, the traffic volume in 5G networks may lead to an energy crunch causing economic and environmental problems [2]. Since energy derived from external sources are abundantly available in the environment, energy harvesting is the most promising technique to fulfill the energy

requirements of wireless networks [3].

While 5G energy harvesting makes 5G smart dust possible, stretching computation across power cycles affects cryptographic algorithms. This effect may lead to new security issues that make the system vulnerable to adversary attacks. Therefore, security measures are needed to protect data at rest and in transit across the network. Understanding the security requirements of existing 5G networks is crucial for this research. Data must be protected at rest and in transit across the network, considering confidentiality, integrity, and authentication. Some challenges we expect include the fact that ultra-low SWaP devices are wimpy, and security must tolerate the disruption of power cycles even when the attacker has physical access.

We want to focus on identifying the security requirements of existing 5G networks and the best-of-breed cryptographic algorithms for ultra-low SWaP devices in an energy harvesting context. To do this, we quantify the performance vs. energy tradespace, investigate the device features that impact the tradespace the most, and assess the security impact when the attacker has access to intermediate results. Our open-source energy-harvesting-tolerant versions of the cryptographic algorithms provide algorithm and device recommendations and ultra-low SWaP energy-harvesting-device-optimized versions of the cryptographic algorithms. Based on the results, we can look into the security measures on intermittent computation devices during their cycle to harvest energy in a batteryless setting.

Chapter 2

Background

2.1 Advanced Encryption Standard

Advanced Encryption Standard (AES) is a symmetric key encryption standard adopted by the U.S. government. The standard comprises three block ciphers, AES-128, AES-192, and AES-256, adopted from a larger collection originally published as Rijndael. Each AES cipher has a 128-bit block size, with key sizes of 128, 192, and 256 bits, respectively [4]. The AES ciphers have been analyzed extensively and are now used worldwide, as was the case with its predecessor, the Data Encryption Standard (DES). AES was announced by the National Institute of Standards and Technology (NIST) as U.S. FIPS PUB 197 (FIPS 197) on November 26, 2001, after a 5-year standardization process in which fifteen competing designs were presented and evaluated before Rijndael was selected as the most suitable [5].

2.2 Secure Hash Algorithm

SHA256 is a cryptographic hash function that was adopted as a FIPS (Federal Information Processing Standards) in 2002 [6]. It is a member of the SHA-2 family of hash functions, including SHA-384 and SHA-512. SHA256 is a 64-bit hash function, producing a 256-bit (32-byte) hash value. This hash function was designed to be secure against collision and preimage attacks. A collision attack occurs when an adversary tries to find two different

messages that produce the same hash value. For a preimage attack, the adversary tries to find the exact message that produces a given hash value.

2.3 Elliptic-curve Cryptography

Elliptic-curve public key cryptosystems were independently developed by Victor Miller and Neal Koblitz in 1985 [7][8]. It is a public key encryption technique based on elliptic curve theory that can be used to create faster, smaller, and more efficient cryptographic keys. ECC is considered a natural modern successor of the RSA cryptosystem because ECC uses smaller keys and signatures than RSA, which makes it faster to generate and use, providing faster key agreement and signature verification. Many algorithms use ECC, including Elliptic Curve Digital Signature Algorithm (ECDSA), Elliptic Curve Diffie-Hellman (ECDH), and Elliptic Curve Integrated Encryption Scheme (ECIES). ECDSA is digital signature algorithm using a private key to sign a message and then uses a public key to verify the signature. The signature is created by performing a mathematical operation on the message or document using the private key. The public key is used to verify the signature by performing the same mathematical operation. If the results of the two operations match, then the signature is valid. ECDH is A key agreement protocol established by the National Institute of Standards and Technology (NIST) in 2018 [9]. It allows two parties, each having an elliptic-curve public–private key pair, to establish a shared secret over an insecure channel, thereby ensuring confidentiality of data. ECIES is a public-key authenticated encryption scheme allowing two parties to encrypt a message or document so that only the intended recipient can decrypt it. When a shared key is generated, it is used to encrypt a message, and the encrypted message can only be decrypted by the party that has the private key corresponding to the public key that was used to encrypt the message.

2.4 Hash-Based Message Authentication Code

HMAC is a cryptographic algorithm that is used to verify the authenticity and integrity of a message. It works by combining a hash function, such as SHA-256 or MD5, with a secret key that is shared between the sender and receiver of the message. The result is a fixed-size, unique authentication code or tag that can be used to verify that the message has not been tampered with or altered during transmission [10].

2.5 Cipher Block Chaining - Message Authentication Code

CMAC is a cryptographic technique that provides message authentication using a symmetric key block cipher, such as the Advanced Encryption Standard (AES). The algorithm depends on the choice of an underlying symmetric key block cipher. CMAC is an adaptation of a keyed-hash message authentication code (HMAC) to block ciphers, allowing them to serve a similar purpose. CMAC enables the detection of message tampering, forgery, or alterations by producing a fixed-length, unique authentication tag for a given message and a secret key [11].

2.6 Rivest-Shamir-Adleman

RSA is a public-key cryptosystem that is widely used for secure data transmission and digital signatures. It is an asymmetric cryptosystem, which means that it uses a pair of keys – one public and one private. The public key can be freely shared, while the private key must remain secret. RSA relies on the mathematical properties of large prime numbers and

modular arithmetic to provide security [12].

Chapter 3

Motivation

3.1 Identify 5G Ultra-Low SWaP Device Security Requirements

We want to explore whether the crypto algorithms, including AES, SHA, RSA, and ECDH will meet the security requirements. These algorithms are used in various settings to meet the security requirements of using a trusted platform. Unlike obsolete crypto algorithms such as DES, these crypto algorithms are still mainly used in different devices. Once we investigate the aforementioned crypto algorithms, we can be confident that most of the crypto algorithms widely implemented in our systems have been investigated. Any other usage of crypto algorithms will be highly unlikely because those areas have not yet been proven.

3.2 Identify and Implement Hardware Support for Crypto

The four requirements include the following. First, true random-number generation (TRNG) with an ephemeral key is required. Second, a unique, unclonable device identifier (PUF) with a static key is required. Third, the hardware must be robust against physical and environmental attacks. Fourth, the system must be simple, low-power, and all-digital. Insecure

options include Ring Oscillators (ROs) and Static Random-Access Memory (SRAM). We will use a RingRAM approach combining ROs and SRAM to eliminate their individual weaknesses.

3.3 Identify Representative Ultra-Low SWaP Platforms

The ultra-low SWaP devices we will experiment with include MSP432P401R, Adafruit Metro M0 Express, MSP430FR5994, MSP430FL5529LP, SAML11 XPLAINED PRO, Apollo3 Blue, and HiFive1 Rev B01. These devices were chosen on the basis of popular usage among researchers and manufacturers of digital devices. These boards cover most of the commonly used devices for architecture and memory usage. All of them will have a software development kit in which the programmer is able to maneuver a way to implement code into the existing devices. Most have a dedicated Integrated Development Environment (IDE) that is expected to be compatible with popular operating systems, including Windows, MAC, and Linux. System-on-a-Chip (SoC) features are crypto accelerators and Trusted Execution Environments. Word widths are 16-, 32-, and 64-bit in thumb mode. Moreover, the grounding requirement is Zephyr OS compatibility and development board availability.

3.4 Define Energy Recording and Replay System

The energy will be recorded by getting the execution time and the current consumption for each device. The datasheet will provide the average current consumption, and this number will be multiplied by the execution time in order to gain energy consumption.

3.5 Identify, analyze, and optimize crypto algorithms

Crypto algorithms will be optimized using the GDB debugger option. We will investigate the impact of different optimization levels on execution time, binary size, and energy consumption. To do this, three different options for optimization are O0 which stands for no optimization; Osize, which stands for optimizing for size; and Ofast, which is an optimization for speed.

3.6 Deal with the ramifications of intermittent computation

One of the main obstructions that deter batteryless sensing from existing in the mainstream of sensing is the lack of a standardized, flexible hardware platform [13]. One apparent reason is that there is room for errors that make the system prone to security vulnerabilities leading to adversary attacks. We need enough energy to be harvested for intermittent computing to turn on the processor. The continuation of program execution depends on the amount of stored energy. Once the power is exhausted, the device abruptly fails and turns back on. The point of turning off is what security researchers are primarily interested in. This is the point where security breaches may occur, and adversaries have a chance to penetrate the system.

Chapter 4

Implementation

An overview of the development process is shown in Figure 4.1. We aimed to find three different C implementations of crypto algorithms to investigate the energy vs. performance tradespace for popular crypto algorithms on seven ultra-low SWaP energy harvesting devices shown in Table 4.1. The crypto algorithm implementations were chosen based on whether or not the algorithm produced the correct output. This was tested by having a result-checking function to verify whether the implementation of the algorithm yields the expected output. Table 4.2 shows the different crypto algorithms found through open-source libraries. Next, we created an open-source testbed that unifies all the C implementations of crypto algorithms. We make use of these implementations by cleaning the code and incorporating it into a single project.

Device	ISA	Memory	Voltage
MSP432P401R	ARMv7m	256 KB Flash, 64 KB RAM	1.82 V
Adafruit Metro M0 Express	ARM m0+	256 KB Flash, 32 KB RAM	3.3 V
MSP430FR5994	MSP430 (FRAM)	256 KB FRAM, 8 KB SRAM	1.8 V-3.6 V
MSP430FL5529LP	MSP430 (Flash)	128 KB Flash, 8 KB SRAM	3.0 V
SAML 11 Xplained Pro	ARM m23	64 KB Flash, 16 KB SRAM	3.0 V-3.6 V
Apollo 3 Blue	ARMm4Sub	1 MB Flash, 384 KB RAM	3.3 V
RISC-V HiFive1 Rev B	RISC-V	4 MB Flash, 16 KB SRAM	1.8 V

Table 4.1: List of Ultra-Low SWaP Devices Tested.

For each of the ultra-low SWaP devices, there exist different development environments. Thus, for each of the devices, we identified and implemented board-specific environments. If

Algorithm	Implementations	License
AES	Tiny	Unlicense
	Gladman	Copyright
	MbedTLS	Apache License
SHA	Saddi	Copyright
	Gladman	Copyright
	MbedTLS	Apache License
ECDH	Micro	BSD 2-clause
	Tiny	Unlicense
	MbedTLS	Apache License
RSA	BearSSL	Copyright
	Tiny	Unlicense
	LibTomCrypt	Unlicense

Table 4.2: Crypto Algorithm and C Implementations of the Algorithm.

an example code ran into errors while compiling and executing, we debugged or found another board-specific environment. After configuring the environment to create a new project, three different implementations of each algorithm were imported into it. For instance, if we were to test the three implementations of AES, all the folders and files containing the three implementations would be imported into the project. Then, a main file was created to initialize the chosen board, start the timer on the board, initialize the algorithm, call the function to test the operation and get the elapsed time in clock cycles. To check the correctness of the operation, there is an optional function to check the result, which is commented out when getting the testing metrics. An overall flow of the testing interface is shown in Figure 4.2.

The open-source testbed provides a step-by-step manual for compiling and running based on the choice of device, algorithm, and implementation. For each device, there exists a separate library customized for a specific development board. In the customized library, we have created different folders containing each crypto algorithm. Inside the folder of a crypto algorithm, there will be libraries for different algorithm implementations. In addition,

there will be a file containing the device-specific main function to run the crypto algorithm operation. A simplified version of the directory tree for a single device-customized library is shown in Figure 4.3.

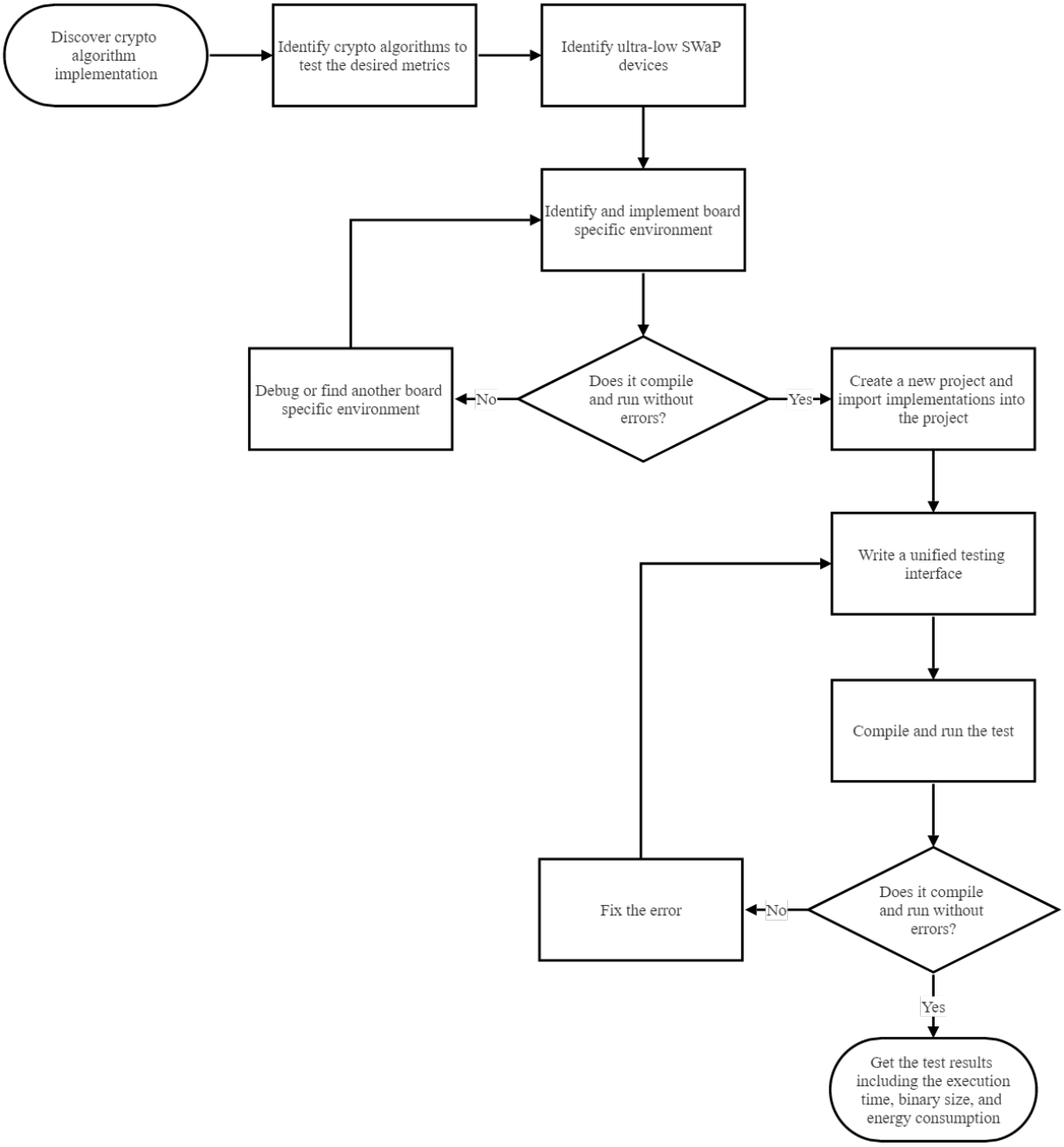


Figure 4.1: Overall flow of the development process to create the testbed.

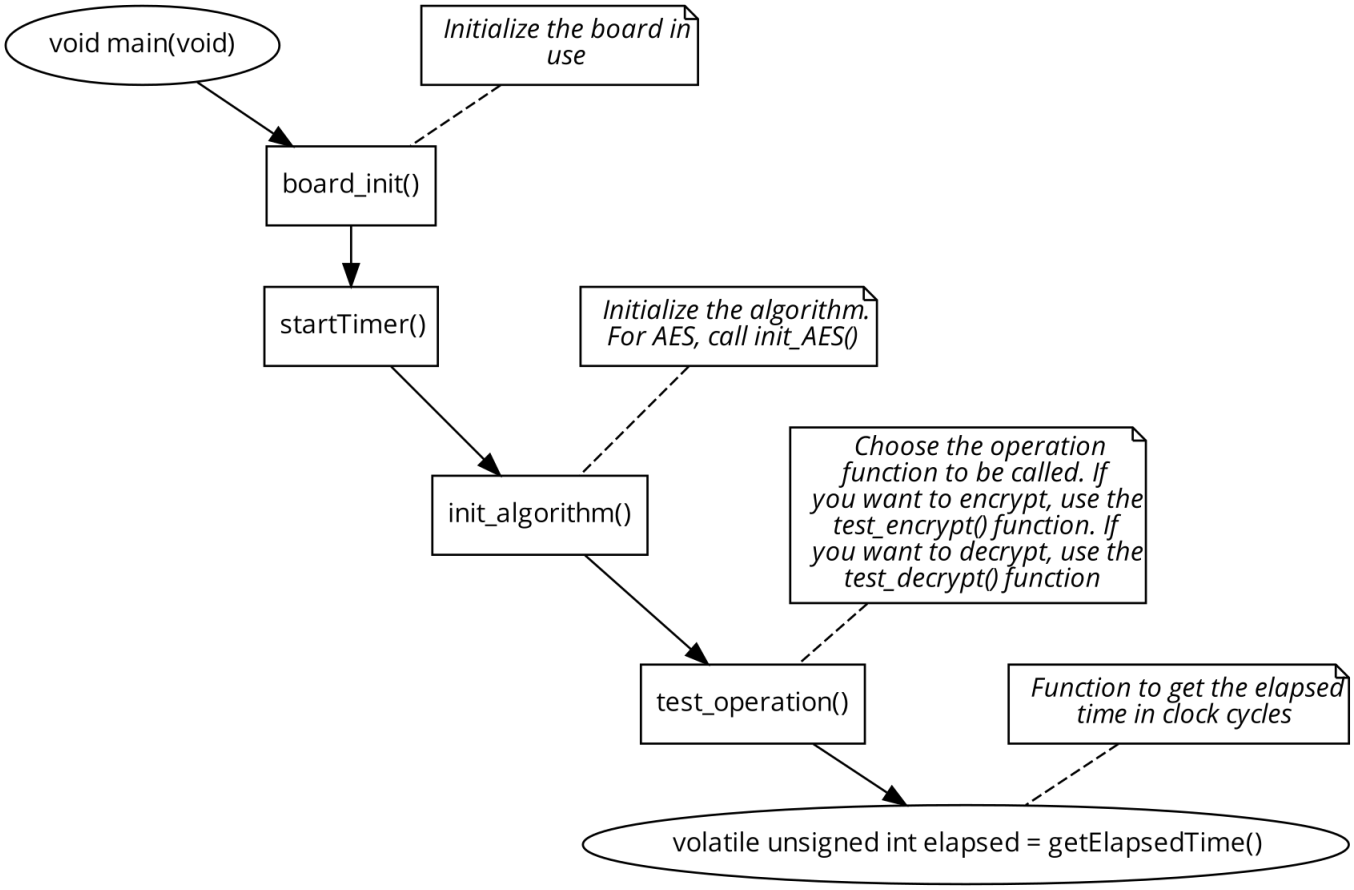


Figure 4.2: Overview of the testing interface.

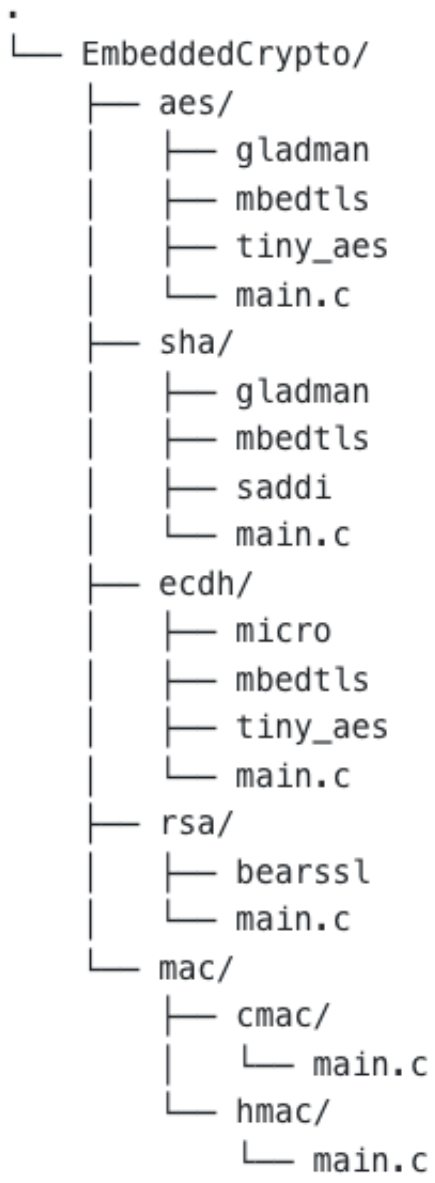


Figure 4.3: Simplified directory tree showing a customized library for a single ultra-low SWaP device.

Chapter 5

Evaluation

5.1 Evaluation Method

Our goal is to provide recommendations for practitioners to decide on the crypto algorithms, algorithm implementations, optimization level, and device combinations that meet individual practitioners' needs. To evaluate our approach, we create evaluation funnels for execution time, energy consumption, and binary size. For each of the evaluation funnels, we answer four questions to achieve our evaluation goal.

1. What is the best implementation for each algorithm?
2. What is the best compiler setting?
3. What is the best algorithm for each task (confidentiality and integrity)?
4. What is the best board for each task?

The evaluation of the performance of crypto algorithms focuses on three metrics. We measured the execution time in clock cycles, energy consumption, and average binary size of the executable when running the crypto algorithm implementation on different ultra-low SWaP devices. To generate the executable for testing, the methodology in Figure 4.1 was followed. Then, the executable was loaded on the specific device being tested. In the development environment outlined in Table 5.1, a compatible compiler must compile and run the program.

Each crypto algorithm test was run on three different GCC compiler optimization levels, which are O0, Ofast, and Osize shown in Table 5.2. The binary size was recorded as the properties of the executable. Then, the execution time was calculated from the cycle count output at the end of the program execution. Next, the energy consumption was calculated by multiplying the power of each device by the execution time retrieved from testing.

Device	Recommended Development Environment
MSP432P401R	Code Composer Studio
Adafruit Metro M0 Express	Arduino IDE
MSP430FR5994	Code Composer Studio
MSP430FL5529LP	Code Composer Studio
SAML 11 Xplained Pro	Microchip Studio
Apollo 3 Blue	Eclipse IDE for Embedded C/C++
RISC-V HiFive1 Rev B	Freedom Studio

Table 5.1: Ultra-Low SWaP Devices and Recommended Development Environments.

Optimization Level	Description
O0	GCC does not perform any optimization and compiles the source code in the most straightforward way possible. Each command in the source code is converted directly to the corresponding instructions in the executable file without rearrangement. This is the best option to use when debugging a program and is the default if no optimization level option is specified.
Ofast	Turns on all optimizations that O3 offers, along with some other optimizations, some of which might not be standards-compliant. For instance, it turns on the fast-math optimization, which assumes floating-point arithmetic is associative (among other things). May or may not make your code faster.
Osize	This option selects optimizations that reduce the size of an executable. The goal of Osize is to produce the smallest possible executable for systems constrained by memory or disk space. In some cases, a smaller executable will also run faster due to better cache usage.

Table 5.2: Three different GCC compiler optimization levels tested.

5.2 Performance

5.2.1 Execution Time

The execution time of each algorithm is an important factor determining the performance of crypto algorithms on ultra-low SWaP devices. Figure 5.1 shows an overall view of the average execution time across crypto algorithms, including AES, SHA, ECDH, HMAC, CMAC, and RSA. It is indicated that ECDH takes the longest time to run among all crypto algorithms regardless of the optimization level. The fastest performance was with the AES256 algorithm at Ofast compiler optimization. This was expected as symmetric key algorithms require less computational power than asymmetric ones and, thus are faster. It is also notable that SHA256 at O0 compiler optimization level is 14x faster than at Osize compiler optimization level.

To answer our first two questions for evaluation, we can take a look at the average execution time across different implementations of AES256, SHA256, ECDH164+, RSA1024, HMAC, and CMAC. From Figure 5.2a, we see that the fastest AES256 implementation is MbedTLS at Ofast optimization level. Compared to Osize of the same implementation, Ofast is 6% faster. Here, Ofast optimization level did generate the fastest result even though this is not a consistent pattern due to its instability. MbedTLS AES is 1.3x faster than Gladman AES and 2.7x faster than Tiny AES.

The fastest implementation of SHA256 shown in Figure 5.2b is MbedTLS at O0 compiler optimization level. This choice of compiler optimization is 7% faster than at Osize. This indicates that Osize is not always faster than O0 even with its small cache size. Comparing the implementations of SHA256, MbedTLS SHA is 1.4x faster than Gladman and 1.4x faster than Saddi.

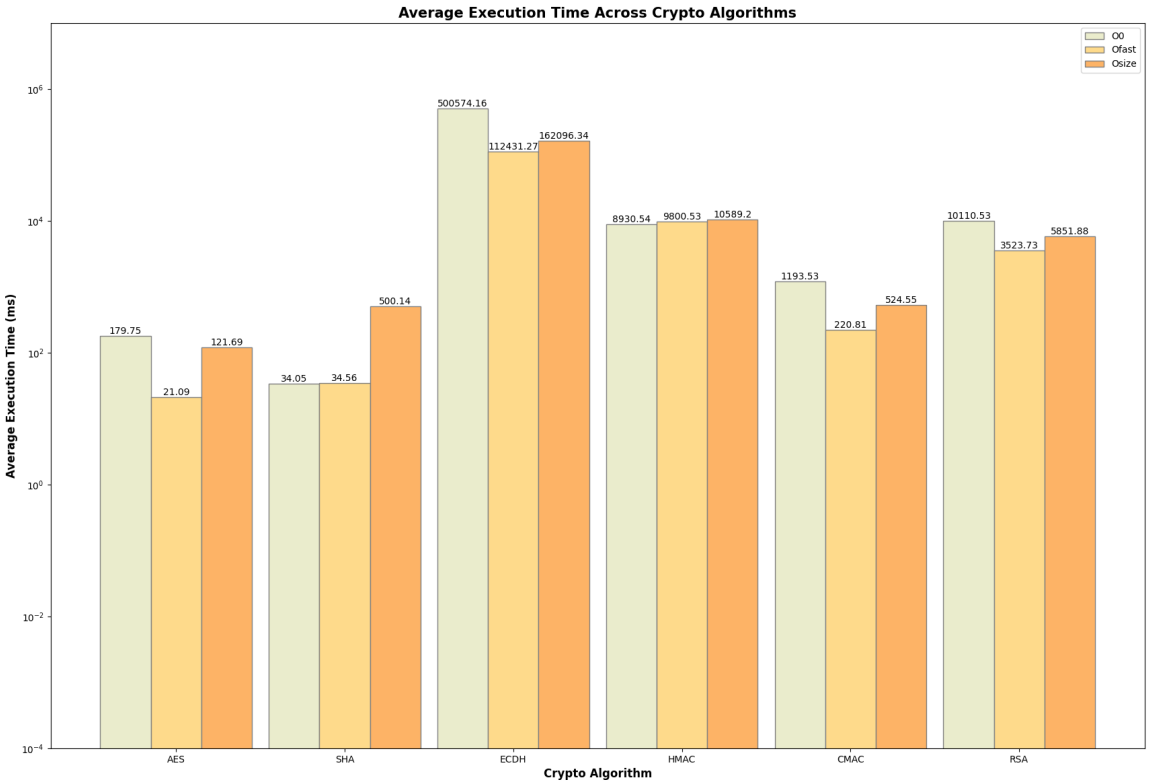


Figure 5.1: Average Execution Time Across Crypto Algorithms

For ECDH164+, the fastest implementation is Micro at Osize compiler optimization level as seen in Figure 5.2c. This is 1.4x faster than at Ofast, which is understandable from the fact that Ofast is an unstable optimization level and optimization at Osize is more likely to produce a smaller executable. Among different implementations, Micro ECDH is 3.7x faster than MbedTLS ECDH and 10.7x faster than Tiny ECDH.

In the case of RSA shown in Figure 5.2d, only one implementation was successfully incorporated in the testbed, which is BearSSL RSA. The fastest compiler optimization level is at Ofast, and this is 1.66x faster than Osize compiler optimization.

For HMAC, the fastest implementation and compiler optimization level is MbedTLS at O0 optimization. CMAC's fastest implementation and compiler optimization level pair is MbedTLS at Ofast optimization. These values are shown in Figure 5.2e and Figure 5.2f, respectively.

To answer our third question, we compare the three algorithms to ensure confidentiality which are AES, ECDH, and RSA. For shared keys, AES is the fastest whereas for public keys, ECDH is the fastest as shown in Figure 5.3. This is expected because ECDH is an improvement to RSA and has shorter key lengths offering better performance.

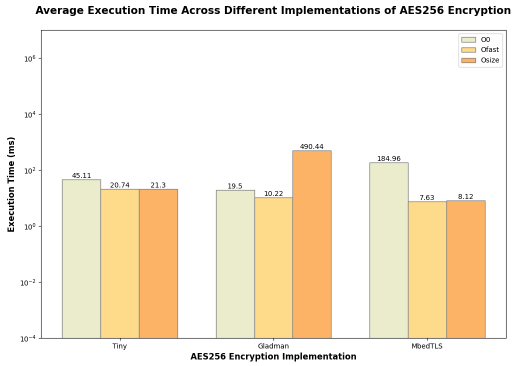
Continuing on with our third question, we compare the three algorithms to ensure integrity which are SHA, CMAC, and HMAC. The best-performing hash function in terms of execution time is CMAC as shown in Figure 5.4.

Lastly, question four is to answer the fastest board for each task; confidentiality and integrity. The best board to ensure confidentiality with shared keys and public keys is Adafruit Metro M0 Express. The best board to have the lowest average execution time and to ensure integrity is MSP432P401R. The overview of the board comparison is shown in Figure 5.5

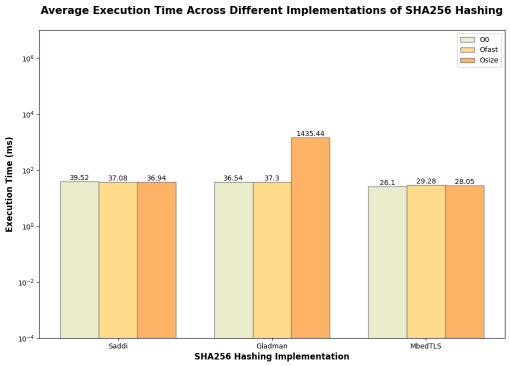
5.2.2 Energy Consumption

When comparing crypto algorithms, the most energy-efficient algorithm and optimization level pair are AES with Ofast optimization level as shown in Figure 5.6. The least energy-efficient algorithm and optimization level pair are ECDH without optimization.

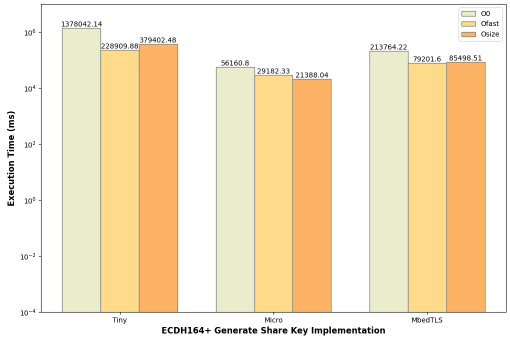
To answer our first two questions for evaluation, we can take a look at the average energy consumption across different implementations of AES256, SHA256, ECDH164+, RSA1024, HMAC, and CMAC. From Figure 5.7a, we see that the most energy-efficient AES256 implementation is Tiny at Osize optimization level. Compared to Ofast of the same implementa-



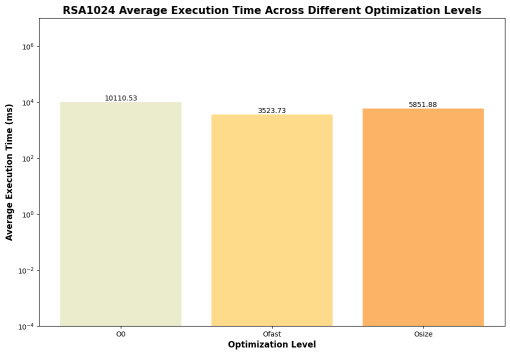
(a) AES256



(b) SHA256



(c) ECDH164+



(d) RSA1024

(e) HMAC

(f) CMAC

Figure 5.2: Comparison of Average Execution Time Across Different Implementations

tion, Osize is 11% more efficient. Tiny AES is 1.8x more efficient than Gladman AES and 1.1x more efficient than MbedTLS.

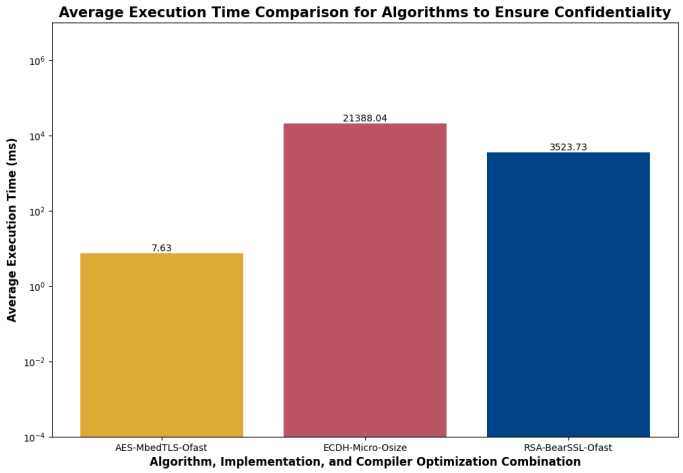


Figure 5.3: Average Execution Time Across Crypto Algorithms

The most energy-efficient implementation of SHA256 shown in Figure 5.7b is Saddi at Osize compiler optimization level. This choice of compiler optimization is 20% more efficient than at Ofast. Comparing the implementations of SHA256, Saddi SHA is 2.5x more efficient than Gladman and 1.2x more efficient than MbedTLS.

For ECDH164+, the most energy-efficient implementation is Micro at Osize compiler optimization level as seen in Figure 5.7c. This is 10% more efficient than at Ofast. Among different implementations, Micro ECDH is 1.7x more efficient than MbedTLS ECDH and 9x more efficient than Tiny ECDH.

In the case of RSA shown in Figure 5.7d, only one implementation was successfully incorporated in the testbed, which is BearSSL RSA. The compiler optimization level with the least energy consumption is at Ofast and this is 1.5x lower than Osize compiler optimization.

For HMAC, the most energy-efficient implementation and compiler optimization level is MbedTLS at O0 optimization. CMAC's fastest implementation and compiler optimization level pair is MbedTLS at Osize optimization. These values are shown in Figure 5.7e and

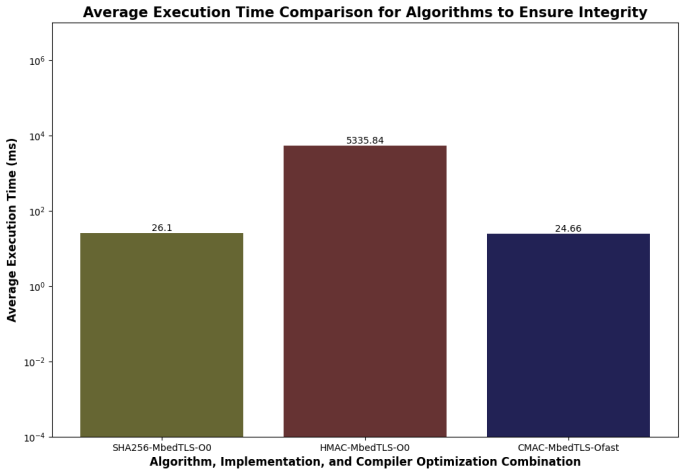
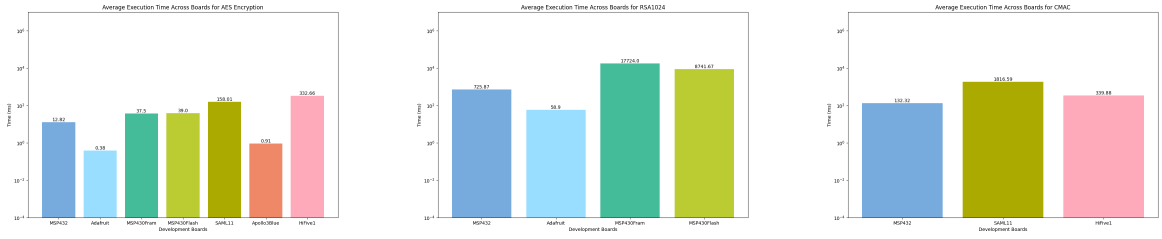


Figure 5.4: Average Execution Time Across Crypto Algorithms



(a) Confidentiality-Shared Keys (b) Confidentiality-Public Keys (c) Integrity

Figure 5.5: Best Board (Execution Time) for Confidentiality and Integrity

Figure 5.7f, respectively.

To answer our third question, we compare the three algorithms to ensure confidentiality which are AES, ECDH, and RSA. For shared keys, AES is the most energy-efficient, whereas for public keys, ECDH is the most energy-efficient, as shown in Figure 5.8

Continuing on with our third question, we compare the three algorithms to ensure integrity which are SHA, CMAC, and HMAC. The best-performing hash function in terms of energy consumption is CMAC as shown in Figure 5.9.

Lastly, question four is to answer which one of the development boards are the most energy-

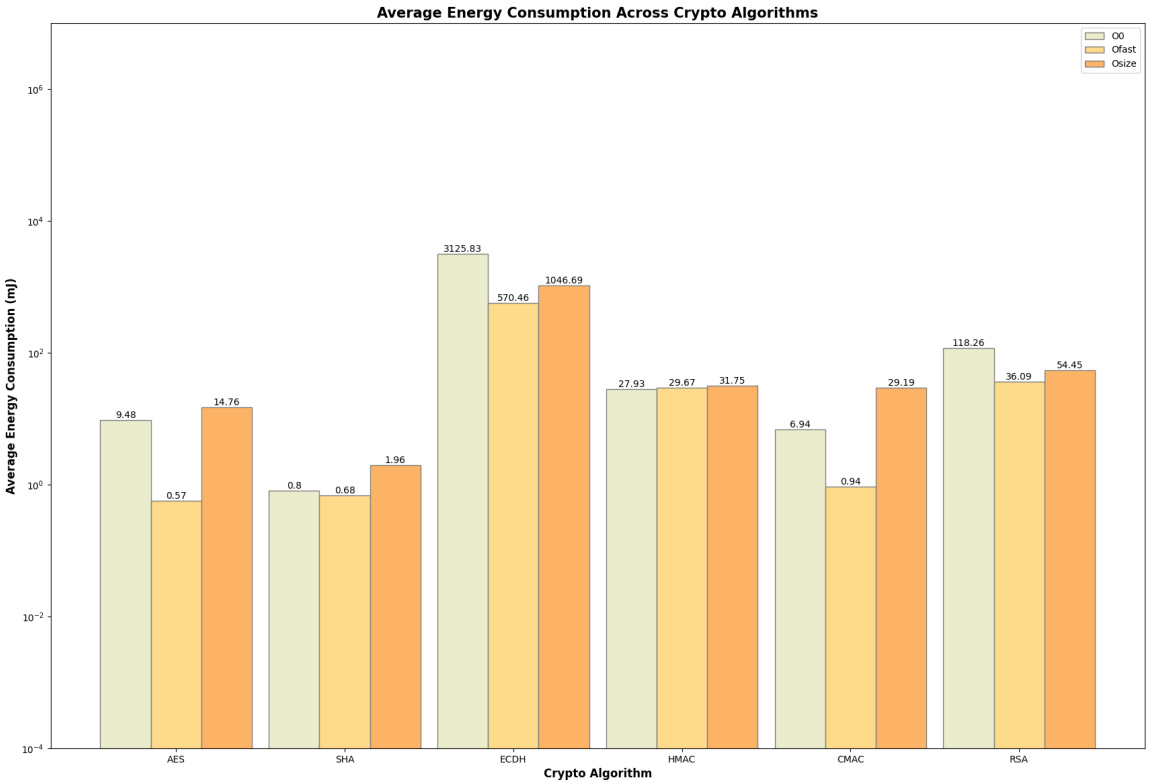
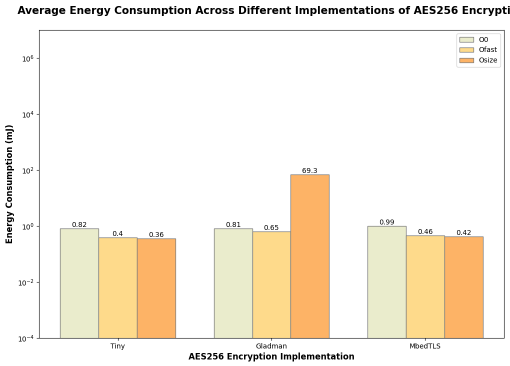
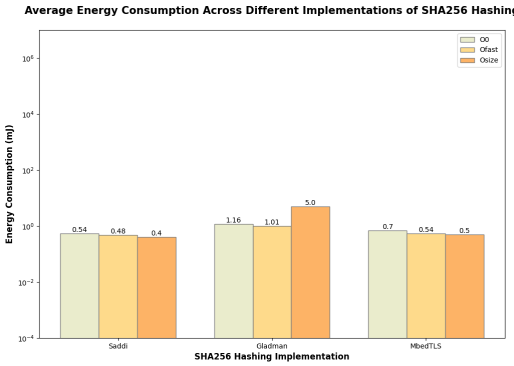


Figure 5.6: Average Energy Consumption Across Crypto Algorithms.

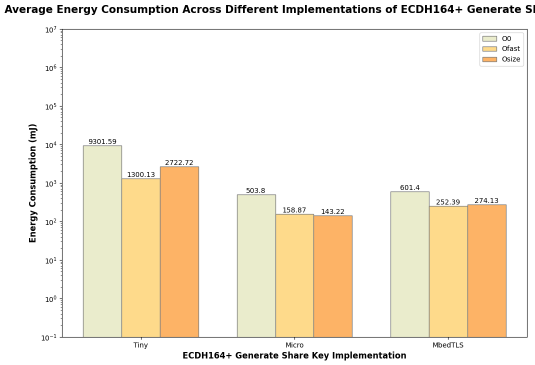
efficient board for each task; confidentiality and integrity. The best board to ensure confidentiality with shared keys is Adafruit Metro M0 Express. In comparison, the best board to ensure confidentiality with public keys is MSP432P401R. The best board to be most energy-efficient and to ensure integrity is MSP432P401R. The overview of the board comparison is shown in Figure 5.10



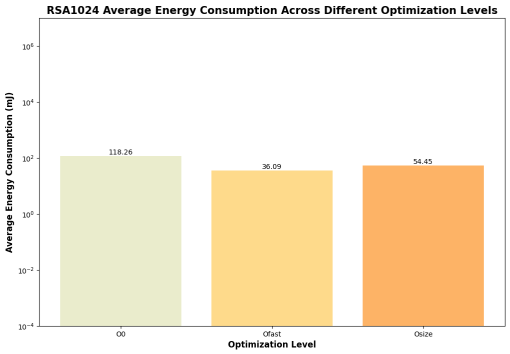
(a) AES256



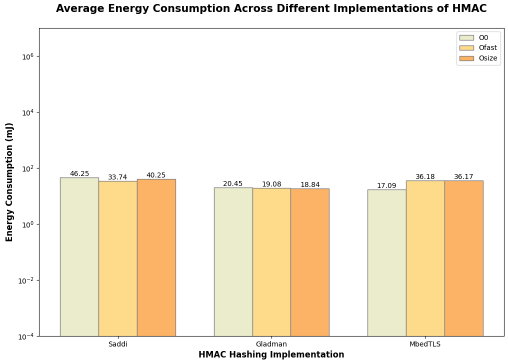
(b) SHA256



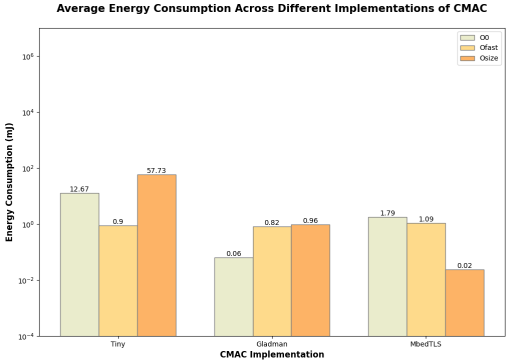
(c) ECDH164+



(d) RSA1024



(e) HMAC



(f) CMAC

Figure 5.7: Comparison of Average Energy Consumption Across Different Implementations

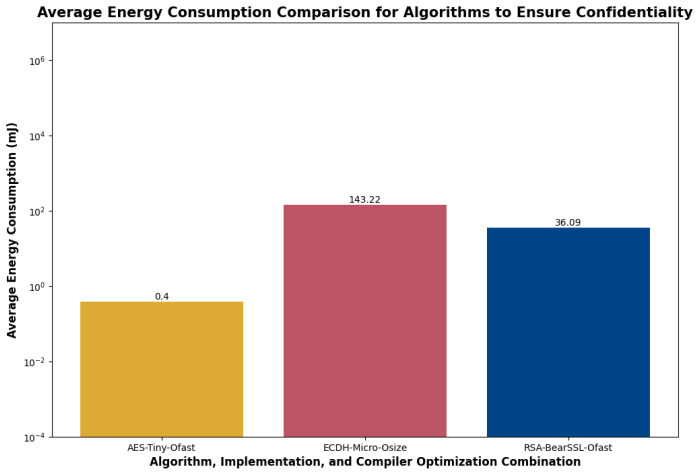


Figure 5.8: Average Energy Consumption Across Crypto Algorithms

5.2.3 Binary Size

The binary size for all of the executables are large because it encompasses a huge library of different crypto implementations, and the executable is run within an Integrated Development Environment. ECDH algorithm and RSA algorithm show an opposite pattern with an optimization level of O0 yielding the smallest executable for ECDH and the biggest for RSA as shown in Figure 5.11.

To answer our first two questions for evaluation, we can take a look at the average binary size across different implementations of AES256, SHA256, ECDH164+, RSA1024, HMAC, and CMAC. From **??**, we see that the implementation with the smallest binary size is MbedTLS at Osize optimization level. Compared to Ofast of the same implementation, Osize is 4% smaller. MbedTLS AES is 1.4x smaller than Gladman AES and 1.2x smaller than Tiny AES.

The implementation with the smallest average binary size in SHA256 shown in **??** is Saddi at Osize compiler optimization level. This choice of compiler optimization is 9% more efficient

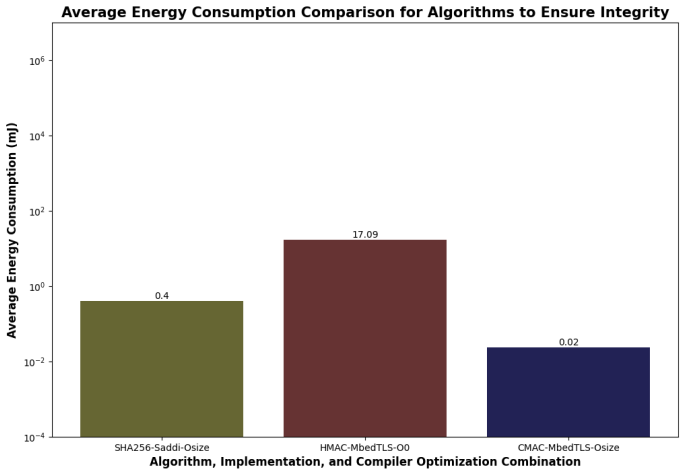
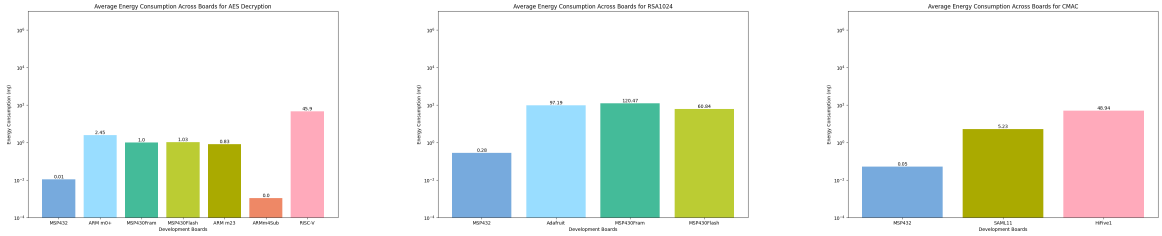


Figure 5.9: Average Energy Consumption Across Crypto Algorithms



(a) Confidentiality-Shared Keys (b) Confidentiality-Public Keys (c) Integrity

Figure 5.10: Best Board (Energy Consumption) for Confidentiality and Integrity

than at Osize. Comparing the implementations of SHA256, Saddi SHA is 7% smaller than Gladman and 4% smaller than MbedTLS.

For ECDH164+, the implementation with the smallest binary size is Tiny at Osize compiler optimization level as seen in **??**. This is 2% smaller than at Ofast. Among different implementations, Tiny ECDH is 1.3x smaller than MbedTLS ECDH and 6% smaller than Micro ECDH.

In the case of RSA shown in **??**, only one implementation was successfully incorporated in the testbed, which is BearSSL RSA. The compiler optimization level with the least bin size

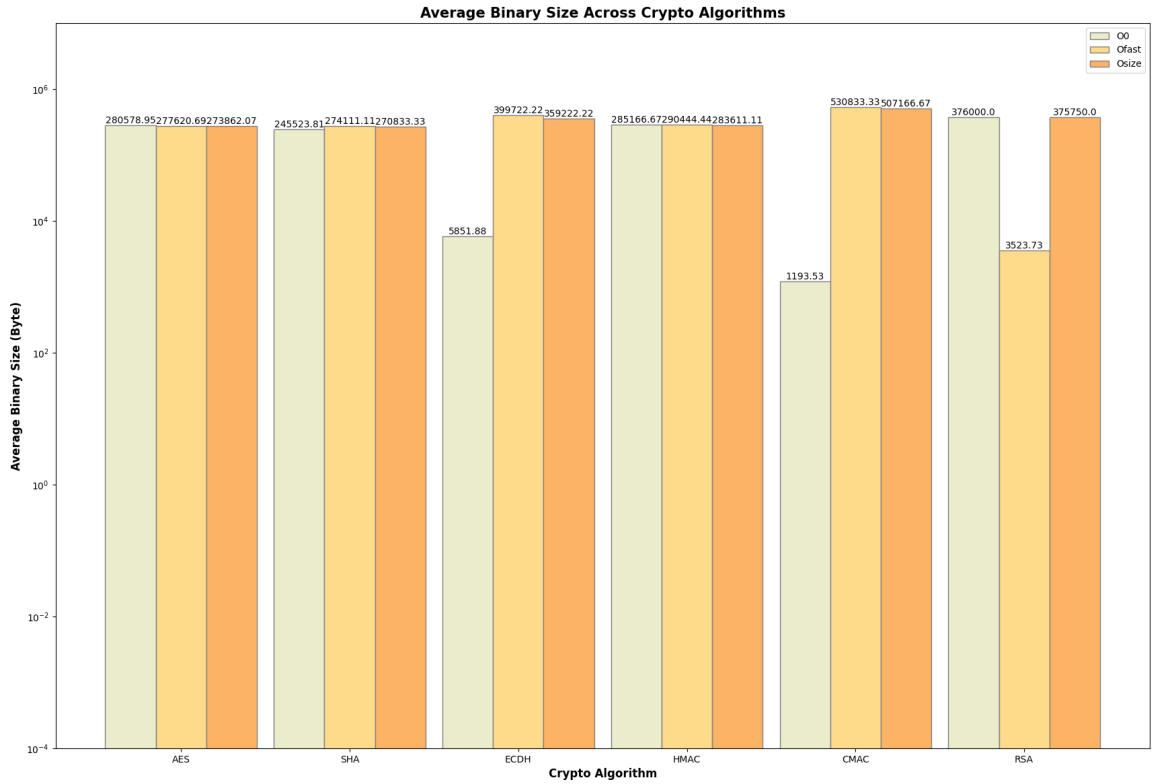
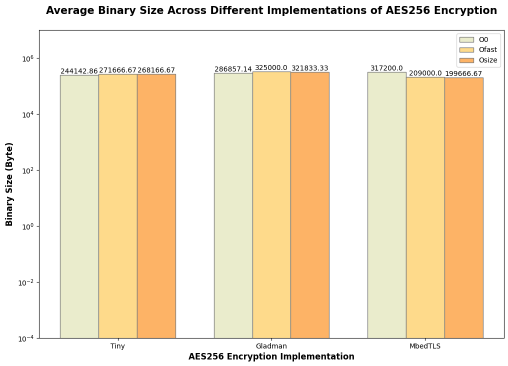


Figure 5.11: Average Binary Size Across Crypto Algorithms.

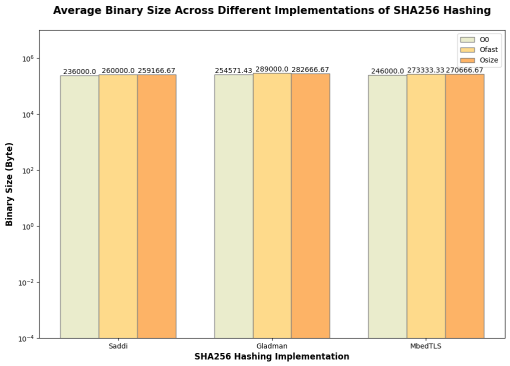
is at Ofast, and this is 106x smaller than O0 compiler optimization.

For HMAC, the implementation yielding the smallest binary size at a certain compiler optimization level is Saddi at O0 optimization. CMAC's fastest implementation and compiler optimization level pair is MbedTLS at Osize optimization. These values are shown in Figure 5.12e and Figure 5.12f, respectively.

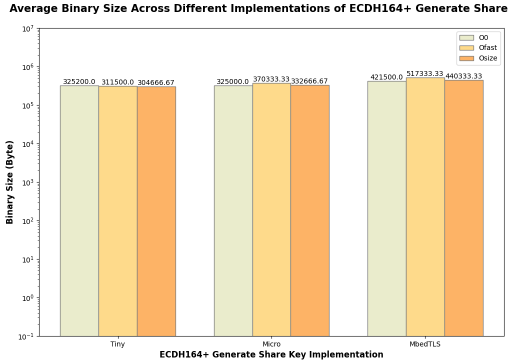
To answer our third question, we compare the three algorithms to ensure confidentiality which are AES, ECDH, and RSA. For shared keys, AES yields the smallest, whereas, for public keys, ECDH is produced the smallest binary size, as shown in Figure 5.13



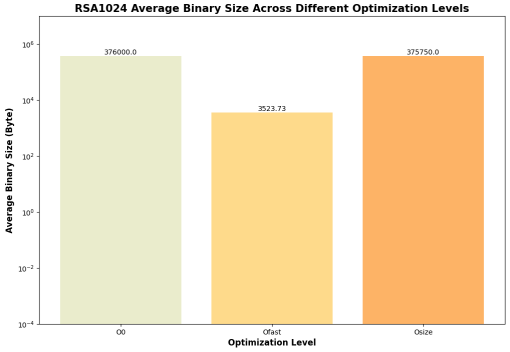
(a) AES256



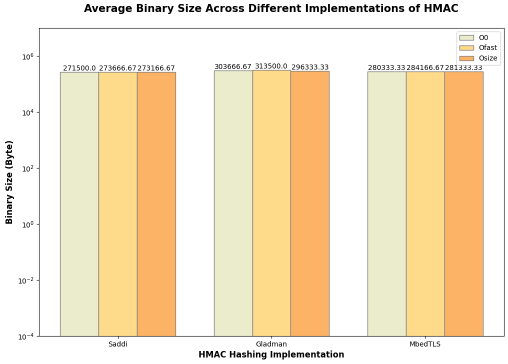
(b) SHA256



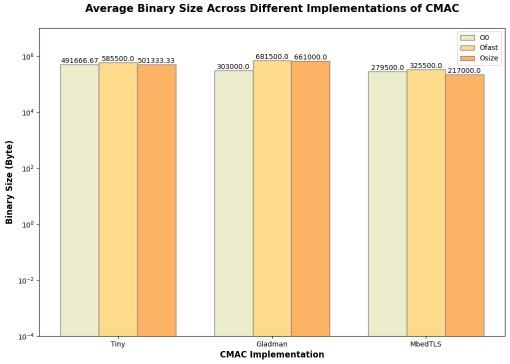
(c) ECDH164+



(d) RSA1024



(e) HMAC



(f) CMAC

Figure 5.12: Comparison of Average Binary Size Across Different Implementations

Continuing on with our third question, we compare the three algorithms to ensure integrity which are SHA, CMAC, and HMAC. The best-performing hash function in terms of binary

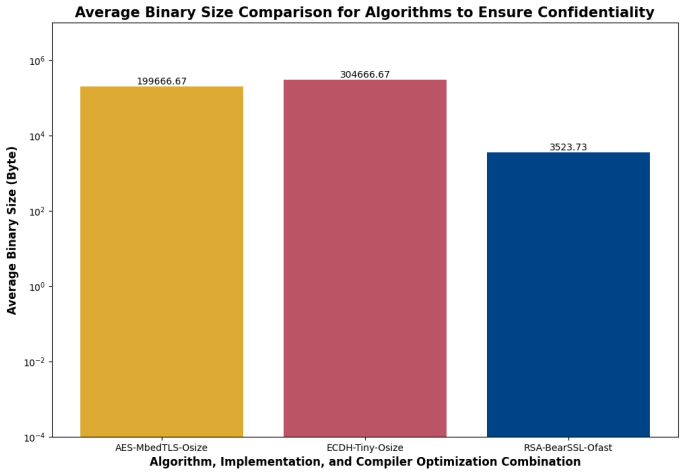


Figure 5.13: Average Binary Size Across Crypto Algorithms

size is CMAC, as shown in Figure 5.14.

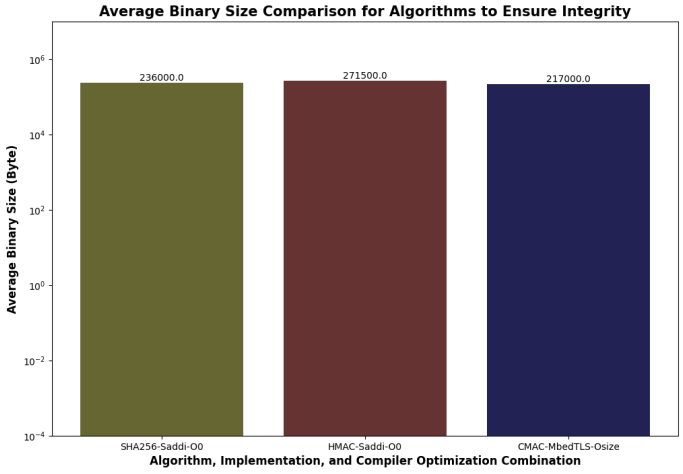
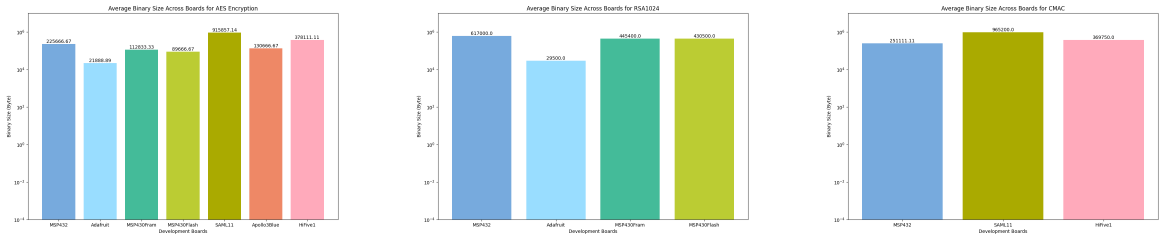


Figure 5.14: Average Binary Size Across Crypto Algorithms

Lastly, question four is to answer which one of the development boards is the most binary-efficient board for each task; confidentiality and integrity. The best board to ensure confidentiality with shared keys and confidentiality with public keys is Adafruit Metro M0 Express. The best board to have the smallest binary size and to ensure integrity is MSP432P401R.

The overview of the board comparison is shown in Figure 5.15



(a) Confidentiality-Shared Keys (b) Confidentiality-Public Keys (c) Integrity

Figure 5.15: Best Board (Binary Size) for Confidentiality and Integrity

Chapter 6

Review of Literature

Cryptography is an essential technology for protecting sensitive data and securing digital communication. Security researchers have focused on developing new techniques and methodologies to improve the security of different crypto algorithms that are widely used on devices. This review of literature explores six research papers that address various aspects of cryptography, including cache attacks on mobile devices, automated identification and classification of proprietary cryptographic primitives in binary code, high-assurance cryptography, verifying high-performance cryptographic assembly code, high-assurance and high-speed cryptographic implementations, and fast, verified, cross-platform cryptographic provider.

Lipp et al. [14] investigated the vulnerabilities in mobile devices that make them susceptible to cross-core and cross-CPU attacks on ARM CPUs. The authors demonstrate how an attacker can exploit the cache in mobile devices to extract sensitive information such as cryptographic keys. They proposed countermeasures to prevent cache attacks against AES T-tables by hardware instructions. The authors claim that they are the first to attack cryptographic primitives implemented in Java. As a lot of crypto research focuses on Assembly language and C implementations, their research is meaningful in that it presents a breakthrough for techniques that can be used to attack various Android devices.

Meijer et al. [15] presented an automated technique for identifying and classifying proprietary cryptographic primitives within binary code. They demonstrate the effectiveness of

their method by analyzing a large corpus of binaries and combining DFG isomorphism with symbolic execution, and introducing a specialized DSL in order to enable the identification of unknown proprietary cryptographic algorithms falling within well-defined taxonomical classes. This technique provides a valuable tool for auditing and analyzing software that incorporates cryptographic primitives.

Kocher et al. [16] found a new class of side-channel attacks called a Spectre attack that involves inducing a victim to speculatively perform operations that will not occur during normal program execution and which leaks the victim's confidential information from a computer system. The attacks exploit the fact that modern processors speculatively execute instructions that may later be discarded, negatively impacting the victim's memory contents. This attack was a breakthrough in computer security research in that it touched upon an intersection in hardware and software security that was previously not in the mainstream of security research.

Lipp et al. [17] presented Meltdown, which is a hardware vulnerability allowing the attacker to read privileged memory from user space and exploits side effects of out-of-order execution on modern processors to read arbitrary kernel-memory locations that include personal data and passwords. While speculative execution exists to improve performance, the researchers have found that where speculative execution exists, Meltdown vulnerability also exists. The major vulnerabilities, such as Spectre and Meltdown, are the types of security breaches that we are trying to look out for on our ultra-low SWaP devices so that the intermittent computing device is not under attack during their on-and-off cycle.

While there are challenges in developing high-assurance cryptographic systems, Barthe et al. [18] proposed a method for developing cryptographic systems that are resilient to Spectre attacks. Their method involves a combination of formal verification techniques, secure coding practices, and hardware-level protections, ensuring that the implementation is resistant to

side-channel attacks. Following the trend of diving into new defense mechanisms in the Spectre and Meltdown era, their adversarial semantics mainly deal with Spectre-PHT and Spectre-STL attacks. Bond et al. [19] presented Vale, a new programming language and tool that supports flexible and automated verification of high-performance assembly code. To improve the verification process's efficiency, Vale employs several optimizations, including abstraction and parallelization.

Almedia et al. [20] proposed a new framework to build high-assurance and high-speed assembly implementations. The authors delivered assembly code that is provably functionally correct and is protected against side-channel attacks. Their approach involves combining formal verification with implementation-specific optimizations to achieve high performance while meeting the security guarantees. This is an advancement from the research that deals with simply high-assurance or high performance. It created a framework to leverage both criteria against side-channel attacks, one of the most notorious attacks threatening hardware and software security.

Protzenko et al. [21] presented EverCrypt, a comprehensive collection of verified, high-performance cryptographic functionalities available via a carefully designed API. The effectiveness of the API was confirmed by comparing its performance with other cryptographic providers, including hashes, Curve25519, and AES-GCM. The authors' approach provides application developers with the functionality, ease of use, and speed they expect from existing cryptographic providers while guaranteeing the correctness and security of the code.

Previous works provide a general guideline to demonstrate the direction of research in cryptography. Our findings will primarily focus on implementing existing crypto algorithms in energy harvesting-enabled microcontrollers and the security impacts of such implementations. Based on this literature review, we can deduce that our investigation into security vulnerabilities will need to start by looking into the possibility of side-channel attacks.

Chapter 7

Conclusions

This paper gives recommendations for other researchers to decide on the crypto algorithms, algorithm implementations, optimization level, and device for research in embedded cryptography. From the evaluation results, we can get the optimal combination of the conditions. For the choice of the most energy-efficient ultra-low SWaP device, MSP430Fram will be ideal because it stably executes all of the crypto algorithms tested and consumes less energy overall. When it comes to the most time-efficient board, Adafruit Metro M0 Express will be the optimal choice because it stably runs all of the crypto algorithms and is the fastest among those which have a higher success rate in running the algorithms without errors. Binary size did not have a significant influence on the efficiency of the experiment. For future works, having more successful cases of RSA algorithms, in addition to some of the conditions that could not be executed without error, will provide a more reasonable analysis of results. Moreover, adopting statistical measures to analyze the different mean values will make the argument stronger.

Bibliography

[1] Joseph M. Kahn, Randy Howard Katz, and Kristofer S. J. Pister. Emerging challenges: Mobile networking for "smart dust". *Journal of Communications and Networks*, 2(3): 188–196, 2000. doi: 10.1109/JCN.2000.6596708.

[2] Sennur Ulukus, Aylin Yener, Elza Erkip, Osvaldo Simeone, Michele Zorzi, Pulkit Grover, and Kaibin Huang. Energy harvesting wireless communications: A review of recent advances. *IEEE Journal on Selected Areas in Communications*, 33(3):360–381, 2015. doi: 10.1109/JSAC.2015.2391531.

[3] Muhammad Ali Imran, Latif Ullah Khan, Ibrar Yaqoob, Ejaz Ahmed, Muhammad Ahsan Qureshi, and Arif Ahmed. Energy harvesting in 5g networks: Taxonomy, requirements, challenges, and future directions. *ArXiv*, abs/1910.00785, 2019.

[4] *Advanced Encryption Standard*. Alpha Press, 2009.

[5] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. Advanced encryption standard (aes), 2001-11-26 2001.

[6] Elaine Barker. Secure hash standard (shs) [includes change notice from 2/25/2004], 2002-08-01 2002.

[7] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, pages 417–426, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg. ISBN 978-3-540-39799-1.

[8] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[9] Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, and Richard Davis. Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography, Apr 2018. URL https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final.

[10] National Institute of Standards and Technology. The keyed-hash message authentication code (hmac), Jul 2008. URL https://csrc.nist.gov/publications/detail/fips/198/1/final.

[11] Morris Dworkin. Recommendation for block cipher modes of operation: The cmac mode for authentication, Oct 2016. URL https://csrc.nist.gov/publications/detail/sp/800-38b/final.

[12] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, feb 1978. ISSN 0001-0782. doi: 10.1145/359340.359342. URL https://doi.org/10.1145/359340.359342.

[13] Josiah Hester and Jacob Sorber. The future of sensing is batteryless, intermittent, and awesome. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450354592. doi: 10.1145/3131672.3131699. URL https://doi.org/10.1145/3131672.3131699.

[14] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. ARMageddon: Cache attacks on mobile devices. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 549–564, Austin, TX, August 2016. USENIX Association. ISBN 978-1-931971-32-4. URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lipp.

[15] Carlo Meijer, Veelasha Moonsamy, and Jos Wetzels. Where's crypto?: Automated identification and classification of proprietary cryptographic primitives in binary code. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 555–572. USENIX Association, August 2021. ISBN 978-1-939133-24-3. URL https://www.usenix.org/conference/usenixsecurity21/presentation/meijer.

[16] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution, 2018.

[17] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 973–990, Baltimore, MD, August 2018. USENIX Association. ISBN 978-1-939133-04-5. URL https://www.usenix.org/conference/usenixsecurity18/presentation/lipp.

[18] Gilles Barthe, Sunjay Cauligi, Benjamin Grégoire, Adrien Koutsos, Kevin Liao, Tiago Oliveira, Swarn Priya, Tamara Rezk, and Peter Schwabe. High-assurance cryptography in the spectre era. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1884–1901, 2021. doi: 10.1109/SP40001.2021.00046.

[19] Barry Bond, Chris Hawblitzel, Manos Kapritsos, K. Rustan M. Leino, Jacob R. Lorch, Bryan Parno, Ashay Rane, Srinath Setty, and Laure Thompson. Vale: Verifying High-Performance cryptographic assembly code. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 917–934, Vancouver, BC, August 2017. USENIX Association. ISBN 978-1-931971-40-9. URL https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/bond.

[20] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Benjamin Grégoire, Adrien Koutsos, Vincent Laporte, Tiago Oliveira, and Pierre-Yves Strub. The last mile: High-assurance and high-speed cryptographic implementations, 2019.

[21] Jonathan Protzenko, Bryan Parno, Aymeric Fromherz, Chris Hawblitzel, Marina Polubelova, Karthikeyan Bhargavan, Benjamin Beurdouche, Joonwon Choi, Antoine Delignat-Lavaud, Cédric Fournet, Natalia Kulatova, Tahina Ramananandro, Aseem Rastogi, Nikhil Swamy, Christoph M. Wintersteiger, and Santiago Zanella-Beguelin. Evercrypt: A fast, verified, cross-platform cryptographic provider. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 983–1002, 2020. doi: 10.1109/SP40000.2020.00114.