A TRIDENT SCHOLAR PROJECT REPORT

NO. 505

A Framework to Evaluate Embedded Security Implementations on the Basis of Common Implementation Mistakes and Vulnerabilities

by

Midshipman 1/C Philip C. Gatbonton, USN

UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND

USNA-1531-2

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
7/12/21		

4. TITLE AND SUBTITLE

A Framework to Evaluate Embedded Security Implementations on the Basis of Common Implementation Mistakes and Vulnerabilities

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Gatbonton, Philip C.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Naval Academy
Annapolis, MD 21402

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

Trident Scholar Report no. 505 (2021)

12. DISTRIBUTION / AVAILABILITY STATEMENT

This document has been approved for public release; its distribution is UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Modern embedded devices are under attack at an unprecedented rate. These devices exist in every facet of society from mobile phones to hard drives and use encryption to protect their data, but attackers may still be able to defeat these protection mechanisms. This is a result of vulnerabilities in the implementation of the encryption algorithm. Our contribution is a set of methods to detect the existence of a particular set of vulnerabilities in a chosen embedded device based on power analysis. This work focuses specifically on Solid State Drives (SSDs) that protect data with the Advanced Encryption Standard and a vulnerability within the SSD's ATA Security Unlock command that has been exploited in previous work. This work analyzes three commonly implemented versions of the SSD's ATA Security Unlock command: (1) a string comparison of the passwords, (2) a hash comparison of the passwords, and (3) a key derivation function that generates a decryption key based on the password. The first two methods are known to have weaknesses in authentication implementations, while the key derivation method is recognized as providing stronger protection of authentication credentials This work implements these three SSD unlock functions on an open source SSD board (Jasmine OpenSSD) and demonstrates the feasibility of detection and classification of these vulnerabilities through power analysis. This work also analyses and detects these vulnerabilities through firmware analysis. Applying these findings to proprietary devices, this work also demonstrates the ability to classify drives based on the specific SSD unlock function implemented in the Crucial family of SSDs.

15. SUBJECT TERMS

Solid state drives, encryption, power analysis, JTAG, reverse engineering

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**		30	**19b. TELEPHONE NUMBER** *(include area code)*

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

U.S.N.A. --- Trident Scholar project report; no. 505 (2021)

A FRAMEWORK TO EVALUATE EMBEDDED SECURITY IMPLEMENTATIONS ON THE BASIS OF COMMON IMPLEMENTATION MISTAKES AND VULNERABILITIES

by

Midshipman 1/C Philip C. Gatbonton
United States Naval Academy
Annapolis, Maryland

Certification of Advisers Approval

Associate Professor T. Owens Walker
Electrical and Computer Engineering Department

Assistant Professor Dane A. Brown
Cyber Science Department

Acceptance for the Trident Scholar Committee

Professor Maria J. Schroeder
Associate Director of Midshipman Research

USNA-1531-2

Abstract

Modern embedded devices are under attack at an unprecedented rate. These devices exist in every facet of society from mobile phones to hard drives, and breaches in their security result in loss of capital and sensitive data. These devices use encryption to protect their data, but attackers may still be able to defeat these protection mechanisms. This is a result of vulnerabilities in the implementation of the encryption algorithm. Our contribution is a set of methods to detect the existence of a particular set of vulnerabilities in a chosen embedded device based on power analysis. This work focuses specifically on Solid State Drives (SSDs) that protect data with the Advanced Encryption Standard and a vulnerability within the SSD's *ATA_Security_Unlock* command that has been exploited in previous work. This work analyzes three commonly implemented versions of the SSD's *ATA_Security_Unlock* command: (1) a string comparison of the passwords, (2) a hash comparison of the passwords, and (3) a key derivation function that generates a decryption key based on the password. The first two methods are known to have weaknesses in authentication implementations, while the key derivation method is recognized as providing stronger protection of authentication credentials This work implements these three SSD unlock functions on an open source SSD board (Jasmine OpenSSD) and demonstrates the feasibility of detection and classification of these vulnerabilities through power analysis. This work also analyses and detects these vulnerabilities through firmware analysis. Applying these findings to proprietary devices, this work also demonstrates the ability to classify drives based on the specific SSD unlock function implemented in the Crucial family of SSDs.

Keywords— Solid State Drives, Encryption, Power Analysis, JTAG, Reverse Engineering

Acknowledgments

I would like to thank my advisors for the all their work and help they have given me since sophomore year. I would also like to thank the Computer Engineering and Cyber Security Research (CECSR) Group, for their considerable contributions, I am thankful to be a part of such a hard working team.

Contents

1 Motivation and Introduction

Every time someone makes a phone call, sends an email, or saves a file, they are broadcasting their data out into the open. If the data is something sensitive, such as bank information, social security numbers, or military orders, ensuring that an attacker cannot access the data is crucial. The way we protect the data is through encryption. Encryption takes our data (the plain text shown in Fig. 1) and a password (the key) and applies a set of operations to produce the protected data (the cipher text). Using the key, operations are performed on the plain text, so an attacker cannot recover the plain text from the cipher text without knowing the original key. The set of operations that the encryption process follows is called the encryption algorithm.

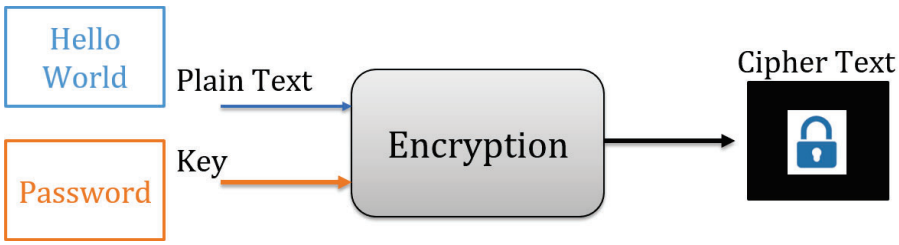

Figure 1: Encryption Overview

The focus of this work is on embedded devices, which have their own processor that implements the Advanced Encryption Standard (AES) algorithm (shown in Fig. 2). AES is the standard encryption algorithm for the government and many businesses. AES is a public algorithm that has been thoroughly analyzed by the cryptography community and no significant vulnerabilities have been found in the algorithm itself . AES has a solid mathematical foundation and sufficiently large keys. For the smallest size key, a 128 bit key, an attacker must compute the 2^{128} possible combinations, which makes successful brute force attacks unlikely. The bottom line is that it is unlikely that a properly implemented AES algorithm can be broken [1].

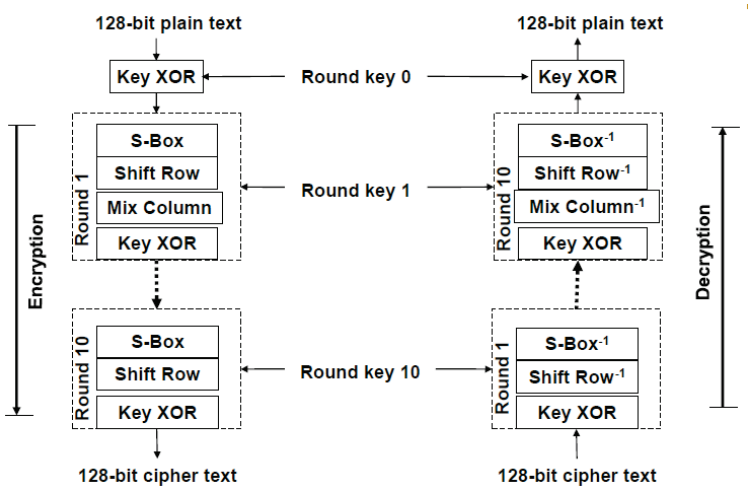

Figure 2: AES Algorithm Overview [14]

However in recent years, an alarming number of attacks have been effective in breaking or bypassing the encryption on various devices. For example, Wired Equivalent Privacy (WEP) is a wireless protocol released in 1997 that uses the RC4 encryption algorithm. However, after a couple of years, security analysts had shown multiple possible attacks to predict traffic, leverage week initialization vectors, and recover the RC4 key [2]. In 2008, Debian project announced that the OpenSSL package on the Debian Operating System had a significant vulnerability in their encryption library which created weak cryptographic keys [3]. In 2015, Meijer and Gastel [4] demonstrated that the implementation of hardware AES encryption on popular Solid State Drives (SSDs), such as Crucial and Samsung, contained significant issues with key management and storage.

If the encryption algorithm is secure, how are attackers breaking or bypassing the encryption? These breakdowns in security target the underlying encryption scheme in the attacked system. In each case, the vulnerability is not caused by the encryption algorithm, but by flaws in the implementation of the encryption algorithm. The encryption algorithm is cryptographically secure, but the issues occur when software engineers and developers implement the algorithm. These implementation errors pose a significant problem to information security. According to the 2019 Veracode State of Software Security Report, cryptographic issues are the second most common type of flaw in all software applications, and they are consistently among the top three most common security flaws over the past decade [6].

2 Project Goals and Objectives

Our objective is to systematically evaluate encryption implementations within embedded devices and create a framework based on the most common encryption implementation flaws and vulnerabilities for a specific device. This work will also develop a set of tests to detect each vulnerability. This work begins by examining SSDs.

1. Develop method for detecting vulnerabilities using power analysis
   (a) Implement AES and PBKDF2 on the Jasmine Open SSD
   (b) Implement a vulnerability
   (c) Measure and collect data on power consumption
   (d) Train classifier and test method on commercial device
2. Develop method for detecting vulnerabilities using hardware debugging interfaces
   (a) Test for the presence of the hardware debugging interface
   (b) Develop the process to use the hardware debugging interface
   (c) Develop the process to recover the firmware
   (d) Identify vulnerability in firmware through the hardware debugging interface

3 Related Work

The findings of the authors, Meijer and Gastel [4], serve as a strong motivation for this work. They demonstrate that the security implementations on popular SSDs, such as Crucial and Samsung, contained significant issues with key management and storage. The researchers reverse engineered the SSDs and revealed weaknesses, which allowed them to exfiltrate the protected data. The researchers point to the fact that hardware encryption schemes often depend on proprietary designs that are harder to verify and implement than open-source or software based solutions. The problems found by the researchers were so prevalent in commercial SSDs that Microsoft advised against using hardware encryption and switch to Bitlocker's default setting to use software encryption instead [5]. A summary of their work is shown in Fig. 3. Multiple compromising vulnerabilities were found in the Crucial, Samsung, and Sandisk SSDs.

Drive	1	2	3	4	5	6	7	8	9	Impact
Crucial MX100 (all)	✗	✗	✗		✗		✓	✓		Compromised
Crucial MX200 (all)	✗	✗	✗		✗		✓	✓		Compromised
Crucial MX300 (all)	✓	✓	✓		✗	✗	✓	✓		Compromised
Sandisk X600 (SATA)	✓	✓	✓		✗	✗	✓	✗		Probably compromised
Samsung 840 EVO (SATA)	✗	✓	✓		✓		✓		✓	Depends
Samsung 850 EVO (SATA)	✗	✓	✓		✓		✓	✓	✓	Depends
Samsung 950 PRO (NVMe)	✗	✓	✓		✓		✓	✓	✓	Probably safe
Samsung T3 (USB)				✗			✓	✓		Compromised
Samsung T5 (USB)				✗			✓	✓		Compromised

[1] Derivation of the DEK from the password in ATA Security (High mode)
[2] Derivation of the DEK from the password in ATA Security (Max mode)
[3] Derivation of the DEK from the password in TCG Opal
[4] Derivation of the DEK from the password in proprietary standard
[5] No single key for entire disk
[6] Not vulnerable to ATA Master password re-enabling (only if derivation is present)
[7] Randomized DEK on sanitize and sufficient random entropy
[8] No wear leveling related issues
[9] No DEVSLP related issues

Figure 3: A summary of the findings from [4]

Publicly available frameworks, such as [7–10] serve as a guide to penetration testing. The most well known is the Penetration Testing Execution Standard (PTES) include steps such as pre-engagement interactions, intelligence gathering, threat modeling, vulnerability analysis, exploitation, post exploitation, and reporting [7, 10]. PTES provides an exhaustive list of possible vectors of exploitation, tools, and resources to accomplish this. NIST's guide for information security testing [11] serves as a practical guide for conducting technical security examinations.

However, the majority of these frameworks list all of the possible vulnerabilities and tools for every device. These frameworks provide a guide for large organizations penetration testing. This work aims to create a framework for embedded devices. As a result, the tests and vulnerabilities will be different for PTES and NIST than our framework. Our framework will be based on the most common encryption implementation mistakes and

vulnerabilities for a specific device. Our framework will also include possible methods to discover these vulnerabilities. This will significantly narrow down the number of possible attacks for researchers and penetration testers.

Using side channel analysis and the power consumption to determine the functionality and security of SSDs has been demonstrated in previous works. In [32], the authors use power consumption to detect malware in programmable logic controllers and [31] detects modified firmware on an open source board. The authors of [29] demonstrate the ability to classify firmware versions. The power consumption has also been used to detect the ongoing operations of SSDs. The authors of [26] and [27] demonstrate the ability to classify on read, write, and trim operations. The current draw has also been used to identify the file system [28] and the encryption algorithm present [30].

4 Background

In 1997, the National Institute of Standards and Technology (NIST) announced that it was accepting new encryption algorithms to replace the older Data Encryption Standard (DES). The new encryption algorithm had to meet NIST standards; it had to be a public algorithm, symmetric encryption, block cipher of 128 bits, and be capable of supporting key sizes of 128, 196, 256 bits. NIST evaluated each of the proposed algorithms in terms of security, cost, efficiency, and algorithm and implementation characteristics. After a few rounds of evaluation by NIST and the cryptography community, NIST announced that the winner of the contest was the Rijndael algorithm, which became known as AES [13]. AES is a ten rounds, where each round is composed of a substitution, shift, mix column, and key XOR step except the tenth round which does not include the mix column step. An overview is shown in Fig. 2.

4.1 Common Encryption Implementation Mistakes and Vulnerabilities

In order to determine the strength of an encryption implementation, the framework evaluates the security of an encryption implementation on the basis of detecting the most common vulnerabilities that may exist in the device. The authors of [15] categorized the most common cryptographic issues from the Common Vulnerabilities and Exposure site (CWE-310) [16]. They found that out of the 269 CVEs they looked at, thirty-nine were issues within the cryptographic library and two hundred and twenty three were issues in the application of the cryptographic library. [4] found implementation mistakes in the derivation of the encryption key and other key management issues.

We group the most common implementation mistakes and vulnerabilities relevant to the scope of our project to be key management, API misuse, insufficient entropy, and vulnerability to side channel attacks. Side channel attacks include power analysis, timing attacks, and fault injection attacks.

4.1.1 Key Management

Proper key management is essential to a secure and robust cryptographic system. Key management involves features dependent on the use and storage of the encryption key. Proper key management involves using keys for only one purpose. They should be sufficiently random and unique or an attacker will be able to recover them easily. Keys should also have a crypto-period which limits the time they can be compromised [17].

AES leaves the key management up to the user. However, there are many common mistakes that can lead to a vulnerability, such as hard coded keys in the firmware or memory, reusing the same key for every encryption, or hard coded and reused initialization vectors (IV). For example, one of the main issues in [4,18], was key management. There were issues of hard coded IVs, keys, and non-random keys, which lead to the researchers recovering the full encryption key or bypassing the encryption process entirely.

4.1.2 API Misuse

Figure 4: AES ECB

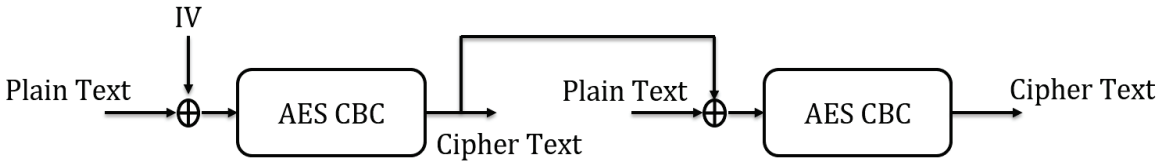

Figure 5: AES CBC

API misuse is poor application of an encryption library. Developers will use an insecure encryption algorithm or mode. For example, devices using the algorithm DES are vulnerable, because computers have become sufficiently powerful to brute force a DES key. Another common API misuse is the use of AES Electric Code Book (ECB) Mode show in Fig. 4. AES ECB is insecure because encrypting identical plain text with the same key will output identical cipher text. Encrypting using AES ECB can result in patterns in the cipher text that reveal information about the plain text.

A graphical representation of the weakness of ECB can be seen in the middle of Fig. 6. The Linux penguin is encrypted using ECB mode, and user's can still see the outline of the penguin. Cipher block chaining, shown in Fig. 5, solves this vulnerability by performing the XOR operation on the current plain text with the previous cipher text (shown on the right of Fig. 6).

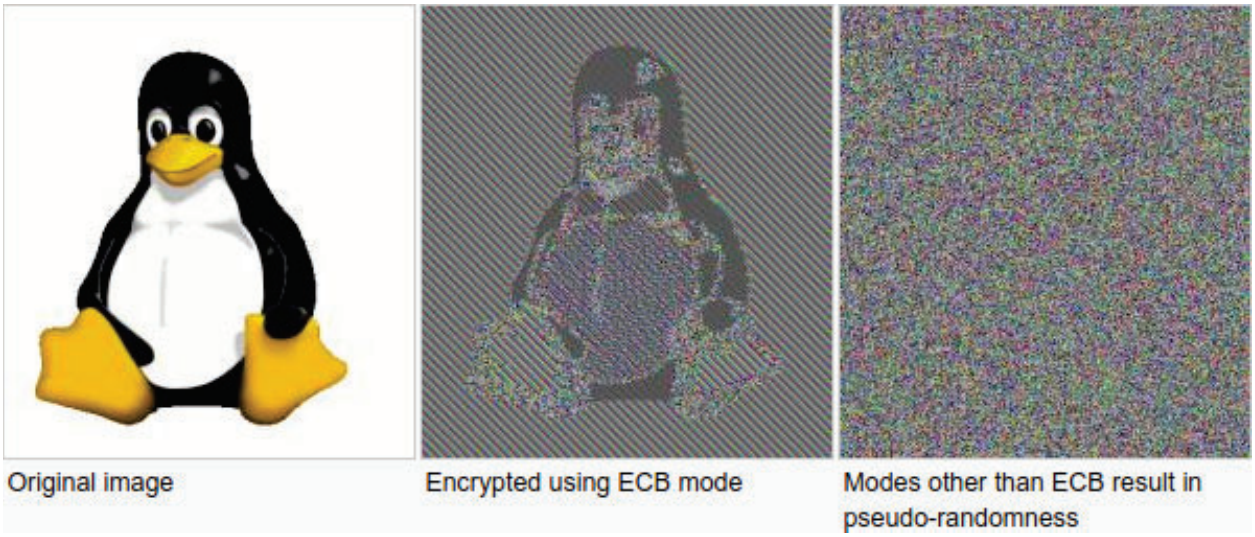

Figure 6: ECB v. CBC Encryption [19]

4.1.3 Insufficient Entropy

Even if the user is using a strong safe and key, cryptographic algorithms do not guarantee information security. A key is only as strong as it is hard for an attacker to guess it (i.e. how random the key is) [20]. Entropy is the measurement of randomness or uncertainty. In order to have a random key, we use true random number generators (TRNG) or pseudo-random number generators (PRNG). For a PRNG to be cryptographically secure, it needs to have pseudo-randomness and it needs to remain secure if the internal state is compromised. The initial seed (state) is taken from an entropy source (shown in Fig. 7). If there is insufficient entropy, then the PRNG is not sufficiently random and therefore the key is not random. As a result, the attacker will have less keys to compute to recover the data [20]. For AES, the key scheduler will produce 10 round keys. However, the round keys are still based off of the initial key, which needs to be sufficiently random.

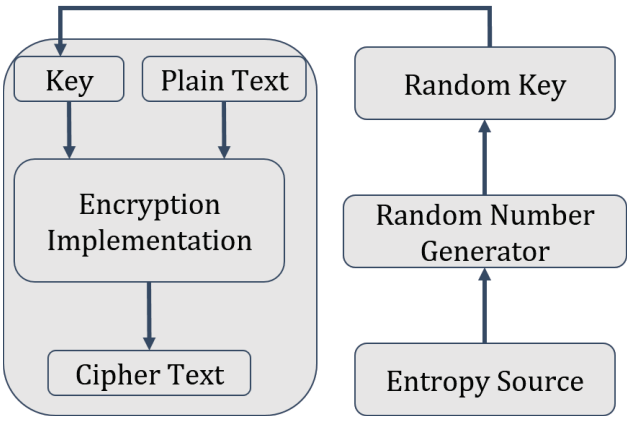

Figure 7: Entropy in an Embedded Device

4.2 SSD

SSDs are a type of storage device that are beginning to replace the traditional hard disk drives (HDDs). SSDs offer a number of advantages over HDDs: superior performance, durability, and efficiency. The main difference that gives SSDs superior performance is that SSDs do not have any moving parts. HDDs use a spinning magnetic disk and a moving read and write arm, while SSDs use NAND Flash memory made up of floating gate transistors. As a result, SSDs have lower read and write latencies, lower power consumption, and more durability.

SSDs consist of two main parts, the flash controller and the flash memory. The flash controller interfaces between the host and the flash memory. The flash controller consists of a microprocessor loaded with firmware that will handle read and write requests from the host, error correction, garbage collection, and redistributing memory blocks to optimize NAND flash memory life. The processor will also control the encryption on the device. Typically many SSDs, called self encrypting drives (SEDs), use hardware encryption on a dedicated co-processor. This frees up the SSD processor for other processes. However, the main processor on the SSD still interfaces with the co-processor and controls the encryption process. The firmware on the main processor will control the key management and timing of encryption of the data [4].

ATA Command	Instruction			OP
SECURITY SET PASSWORD	SecuSetPsw	SESP		F1h
SECURITY UNLOCK	SecuUnlock	SEUL		F2h
SECURITY ERASE PREPARE	SecuErasePrep	SERP		F3h
SECURITY ERASE UNIT	SecuEraseUnit	SEEU		F4h
CFA WEAR LEVEL	CfaWearLevel	CFWL		F5h
SECURITY FREEZE LOCK	SecuFrzLock	SFZL		F5h
SECURITY DISABLE PASSWORD	SecuDisPsw	SEDP		F6h
READ NATIVE MAX ADDRESS	RdNativeMax	RNMA		F8h
SET MAX ADDRESS	SetMaxAddr	SMXA	SMAX	F9h
SET MAX FREEZE LOCK	SetMaxFrzLock	SMFL		F9h
SET MAX LOCK	SetMaxLock	SMLK		F9h

Figure 8: SSD ATA Table

The Advanced Technology Attachement (ATA) standard defines the physical interface for connecting SSDs to a host computer. The ATA standard also defines a security feature set, which allows the user to set a password and lock the SSD with it [21]. The security commands and the opcodes are shown in Fig. 8. However, ATA Security feature set does not specify the how the user would be authenticated [4].

The newer standard for SEDs is the TCG Opal Specification [22]. The TCG Opal Specification overlays the ATA Standard. TCG Opal specifies that the SSD use AES-128 or AES-256. An example implementation overview of the TCG Opal Specification is shown in Fig. 9. This is based on Crucial's implementation of the TCG Opal Specification found in the Crucial MX300 and MX500 [4]. The SSD typically implements a multiple range Disk Encryption Key (DEK) and a Media Encryption Key (MEK). The data is encrypted and decrypted with the DEK. The DEK can be generated on the SSD or on the host CPU, and either way, the DEK needs to be generated by a system with a high level of entropy. How random the DEK is highly dependent on the level of entropy in the device. When a user sets the password on the SSD, they are setting the MEK. The MEK is generated from the

password using a password derivation function. When the user enters the password, the host computer will send the ATA Security Unlock command to the SSD. The SSD will generate the MEK from the password using a key derivation function like PBKDF2. The produced MEK will decrypt the DEK. The DEK will decrypt the test data, and if the DEK decrypt the test data correctly, then the rest of the drive can be decrypted with the DEK [4].

However, with both of these standards, there does not exist a verification process to ensure that manufacturers are implementing them correctly. Neither does there exist a example of a secure implementation. Manufacturers are left to implement the ATA and TCG Opal standard leading to mistakes and vulnerabilities in the security implementation.

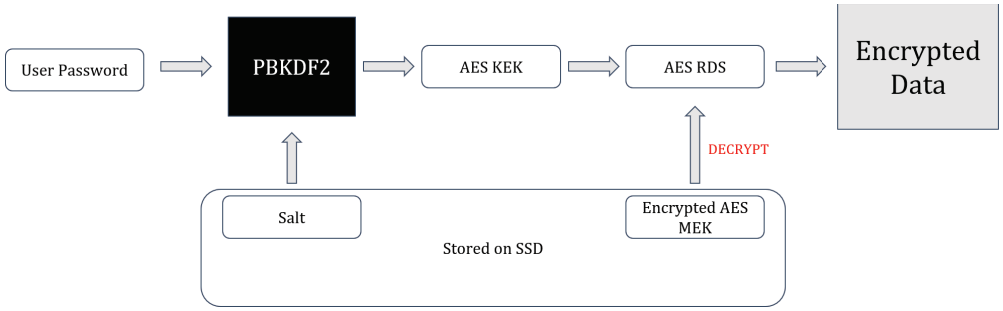

Figure 9: SSD Encryption Engine

4.3 Power Analysis

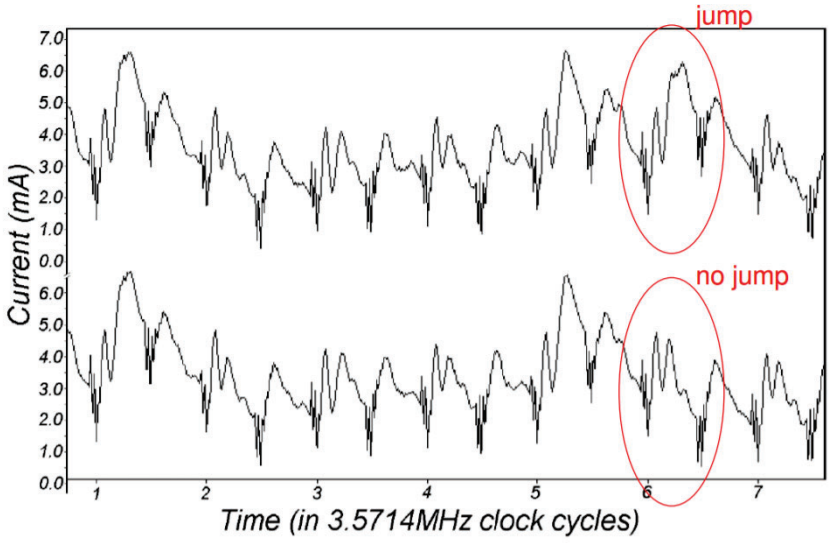

Figure 10: Simple Power Analysis Example [23]

All embedded devices will leak some information through power, timing, electromagnetic emanations, or sound. Attackers can analyze the information leakage to extract data, information about the system, and encryption keys. We will be performing Simple Power

Analysis (SPA) on vulnerable devices. SPA involves measuring the power consumption of a device over time. As the device operates, the power consumption will fluctuate as a result of the operations of the CPU. This can be used to discover the device's operations or even its encryption key. In Fig. 7, the authors in [23] are examining the power consumption of DES implementation. The main difference between the two power traces are the spike in power at 6, which indicates that the jump was taken in the DES algorithm.

4.4 Hardware Debugging Interface

Joint Test Action Group (JTAG) is a common hardware interface on embedded devices. It is used by developers to verify circuit designs and test embedded devices. In [4], the authors showed that many SSDs manufacturers left the JTAG accessible to attackers. They were able to use JTAG to identify and exploit vulnerabilities that existed on the SSDs. JTAG and UART are debugging ports used to interface with the CPU and memory. JTAG can provide an attacker with full control over the device. Through JTAG, an attacker can halt the processor, step through the firmware, and extract memory.

Figure 11: Key Derivation Process Overview

5 Vulnerability: Key Management

Key management is the initial focus of this work, specifically the key derivation process. This section describes three possible implementations and susceptibility to attack. As discussed in Section 4.2, the key derivation process occurs in the Unlock command of the ATA Security Feature Set of many commercial drives.

A large number of the vulnerabilities in Fig. 3 found by [4] involved the derivation of the DEK. This work identifies three different implementations found in commercial SSDs and identifies them using reverse engineering and power analysis. The first implementation is shown in Listing 1. This implementation takes in a user password and compares it to the password stored on the SSD. If the two are equal, then the SSD will unlock the SSD. The second version, shown in Listing, 2 is similar to the first, except that prior to comparing the passwords, the password is hashed using SHA256. After hashing the password, it is compared to the stored password hash. The third version is a key derivation function that generates a decryption key based on the password. As discussed in Section 4.2, the first step in the key derivation process is the host computer sends over the user password in the form of the ATA Security Unlock Command. From the user password, the SSD derives a AES key which will decrypt the AES DEK.

Listing 1: ATA Unlock Command (String Comparison)

```
// ATA Unlock Function
    // 1. Request user password
    // 2. Compare incoming password to password stored on SSD
    if(user_pwd != stored_pwd) {
      // 2a. If not equal command aborted
    }
    // 3. If equal, unlock SSD and produce AES DEK to decrypt SSD
```

Listing 2: ATA Unlock Command (SHA256 Comparison)

```
// ATA Unlock Function
    // 1. Request user password
    // 2. Hash user password
    hashed_user_password = sha256(user_password)
    // 2. Compare incoming password hash to the stored password
    //    hash on SSD
    if(hashed_user_password != hashed_stored_pwd) {
      // 2a. If not equal command aborted
    }
    // 3. If equal, unlock SSD and produce AES DEK to decrypt SSD
```

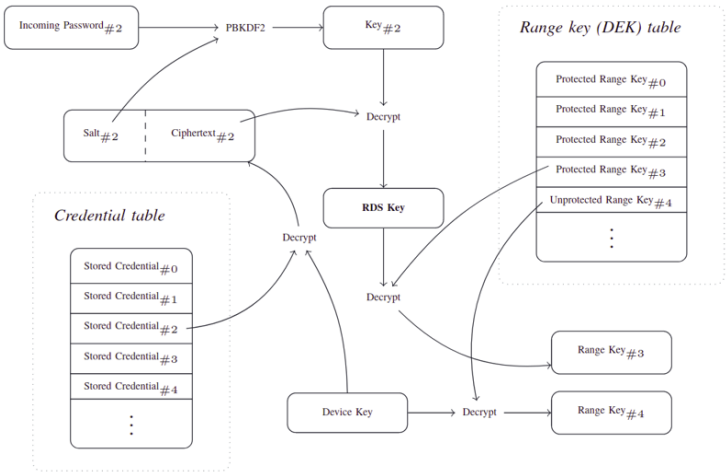

Figure 12: ATA Unlock Command (PBKDF2) [4]

6 Methodology

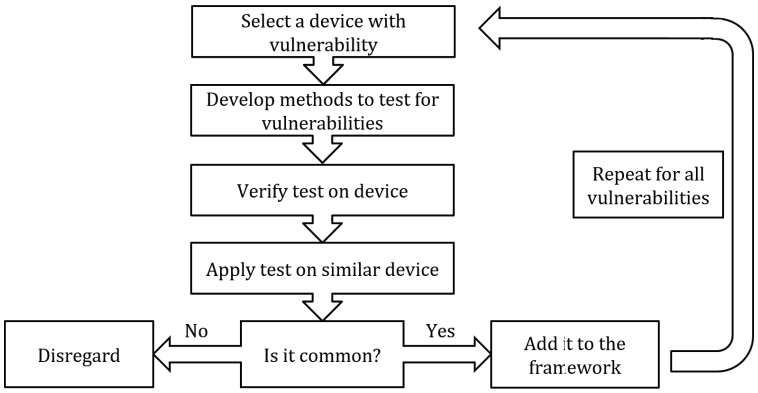

Figure 13: Method to generate framework

This section discusses the general approach to developing the method. The process followed is shown in Fig. 13. The first step is to select a device to develop and test our methods on. For this, the Jasmine Open Source SSD, a development board is used. The Jasmine board is an open source board that emulates commercial SSDs. The Jasmine components include an Indilinx Barefoot SSD controller, an ARM7TDMI-S core, and NAND flash memory. Using an open source board allows for the development of firmwares that include security protocols and vulnerabilities on the device. The board can be used as a device where the vulnerabilities are known. As shown in Fig. 14, an unverified method can be tested on a device in which the vulnerability is known. This allows verification of the methods on a known device. The next step in the framework is to develop methods that can detect the known vulnerability on the Jasmine. This work focuses on two approaches: (1) Power analysis and (2) using hardware debugging interfaces. Once the method has been verified, it can applied it to a similar commercial device in which it is not known if the vulnerability exists. The last step in the process is to see how common the vulnerability is. Does the vulnerability exist in more than one commercial device? After testing this, we can start compiling a list of vulnerabilities that are common to SSDs. This work will follow this process for all vulnerabilities discussed in Section 4.1.

6.1 Jasmine Software Development

In order to develop power analysis and JTAG as a method to detect the vulnerability within the ATA Unlock command, the ATA Standard and the TCG Opal Specification had to be implemented first on the Jasmine. The following additions were made to the Jasmine Open Source Library in order to implement the security specification.

1. Integrate an Open Source AES Library with Jasmine firmware
2. Integrate Open Source PBKDF2 with Jasmine firmware
3. Implement the ATA Security Feature Set

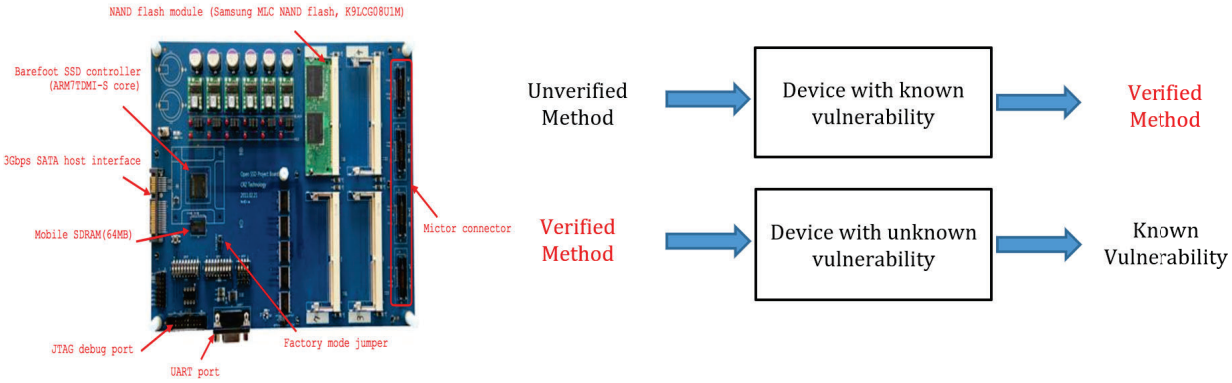

Figure 14: Using the Jasmine to develop methods to detect vulnerabilities

(a) Implement ATA Security Unlock

(b) Implement ATA Security Set Password

(c) Implement ATA Security Disable Password

4. Implement the three different ATA Unlock Functions shown in Listings 1 and 2 and Fig. 12

The original Jasmine Source Code did not include an embedded SHA256, PBKDF2, or AES function. This work used the tiny-AES-c embedded open source library and the library PKCS5_PBKDF2. These libraries were integrated into the Jasmine firmware. Also, the ATA Security Feature Set was implemented according to [21]. The ATA Security Unlock, ATA Security Set Password, and ATA Security Disable Password functions were implemented and added to the ATA opcode table. The ATA opcode table is a common feature in ATA SSDs that includes a list of the implemented ATA commands and the pointers to the function implementing it. One feature that was not implemented was the master and user password bit as we were only testing for the presence of key derivation function.

6.2 Power Analysis of Key Derivation Scheme

The power consumption of the device can provide insight into the inner operations of an SSD [?,27–29]. This work tests for differences in power consumption traces using the classification techniques enumerated in [?, 27–29]. The set up is shown in Fig. 15. The host computer reads, writes, and sends the ATA commands to the SSD. The recorder is the Gen3i. The power is measured through a current probe looped over the 5v power line into the SSD. The developing computer receives debugging messages from the device.

After implementing the two firmwares on the Jasmine Board, data was collected using the set up shown in Fig. 15. The data collection on the Jasmine Board was manual and randomized. The process used to collect the data for training and testing follows. First, a collection schedule is created by generating random number list of N number collections desired of numbers between 0 and T minutes every tn minutes to collect data [0, t1, ..., T].

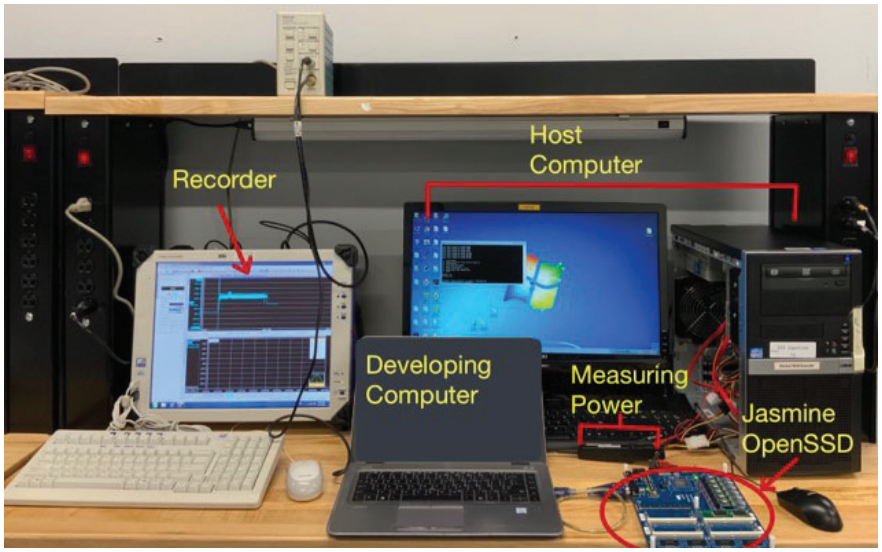

Figure 15: Power Analysis Set up

A random number list of N number collections of numbers 0 and 1 is generated where 0 represents a non-PBKDF2 firmware collection and 1 represents a PBKDF2 firmware collection. After creating the collection schedule, the collection process follows Fig. 16.

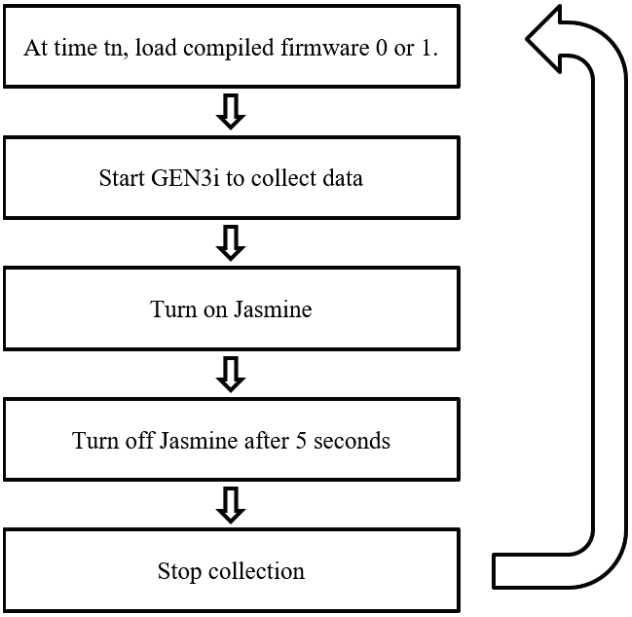

Figure 16: Jasmine Power Analysis Collection Process

The randomness in the data collection prevents each individual data collection from being correlated. In [**?**, 27–29], the authors found that the time of day affected the classification accuracy of the data. If the data was collected in batches, then the data would be correlated through the time of the day it was collected in. Batched collection is when data is collected all at one time or in a row. For example, collecting ten of one firmware then ten of the other firmware would be batch collection.

After the data was collected, the next step was to explore the data and manually identify key features in both plots. After extracting these features, this work trained a MATLAB R2020b classifier. The main plateau feature is present in every trial, when comparing a PBKDF2 plot and a non-PBKDF2 plot, which allows an accurate classification. After testing the classifier on the Jasmine, it is then applied against the Crucial family of SSDs.

In order to accomplish this, the first step was to collect power consumption data. The data collection of commercial drives was automated following a similar process to the data collection on the Jasmine shown in Fig. 15. The collection begins by issuing the start collection command and the host computer sending the ATA Set Password command to the SSD. The host computer sleeps for three seconds and then sends the ATA Unlock Command. After, the host computer sleeps for 5 seconds and the end collection command is sent. The commercial drives were collected two at a time using two amplifiers and current probes. The Crucial SSD data was manually analyzed and compared to the power consumption of the Jasmine. After analyzing the Crucial family, other manufacturers were analyzed to demonstrate the ability to identify ATA Unlock command implementation across a variety of drives.

6.3 Hardware Debugging Interface

In order to analyze the firmware, JTAG (run time analysis or dynamic analysis) and Ghidra (static anaylsis) are used. JTAG is developed as method to identify vulnerabilities in the

firmware using the same process outlined in Fig. 13.

The initial device used to develop the method was the Jasmine. First, the Jasmine's JTAG connection needed to be set up. Then, the JTAG commands (halt processor and read memory) needed to be tested on the Jasmine. After tesing these commands, the firmware is extracted through JTAG. After obtaining the firmware, the binary is analyzed through Ghidra.

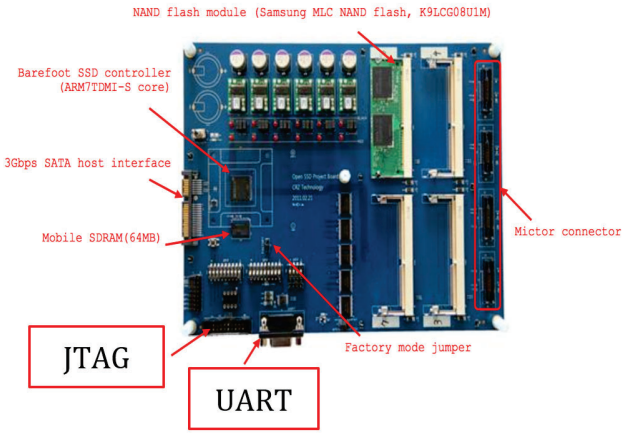

Figure 17: Jasmine JTAG and UART Ports

The set up for this method is the same as in Fig. 15. The first step in the method was to set up the connection to JTAG. The Segger J-Link is used, which is a JTAG emulator for ARM processors. The Segger J-Link allows the developing computer to interface with the Jasmine core. Since the Jasmine is a development board, the JTAG port is labeled in the manual shown in the bottom left of Fig. 17. The Jasmine has a standard ARM 20 pin connector. After the Segger J-Link was connected directly to the Jasmine, the J-Link software automatically configured the settings to interface with the Jasmine. The next step was to to test the connection and configuration by halting the processor and reading from memory. The processor was successfully successfully halted and various locations of memory were read off.

After testing the JTAG connection, the firmware's location was identified and recovered. From the Jasmine manual [24], the firmware resides at location 0x10000000 in memory during normal operation. The data from this memory location through the JTAG dump_image command. Fig. 18 shows a comparison of the memory dump versus the actual firmware that was compiled and loaded on the Jasmine. Initially, the memory extracted from the Jasmine looks similar to the actual firmware. However, starting at 0x28, the memory dump begins to differ from the firmware.

Analyzing the two binaries through Ghidra, the NSA reverse engineering software, shows more definite differences in the assembly. In Fig. 19, the assembly before addresses 0x00002558 on the left (actual firmware) and 0x100003c3 on the right (memory dump) match. However, after these addresses, the assembly begins to differ. For example, there exists an branch command at 0x10000c5c that does not exist in the actual firmware. An important note is that the addresses the assembly commands exist at are arbitrary. The problems with JTAG were solved by using alternative methods to extract the firmware [4].

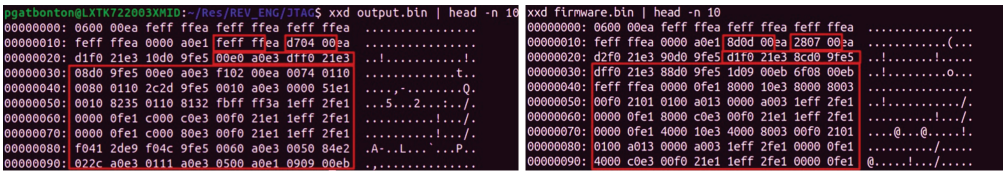

Figure 18: Jasmine JTAG Memory Dump v. Actual Firmware

Another method is to recover the firmware through firmware updates. For the Jasmine, this is trivial, because the source code is included and compiled by the user. Through this, the Jasmine Firmware was analyzed statically in Ghidra. In [4], they identified the sector information and the ATA function table as a stepping stone to successfully reverse engineer a SSDs firwmare. The sector information would lay out the organization of the binary, where the different source code files lay in memory. The ATA function table is a table of ATA commands and the function pointers that implement them. Taking inspiration from this, these steps were followed to analyze the Jasmine firmware.

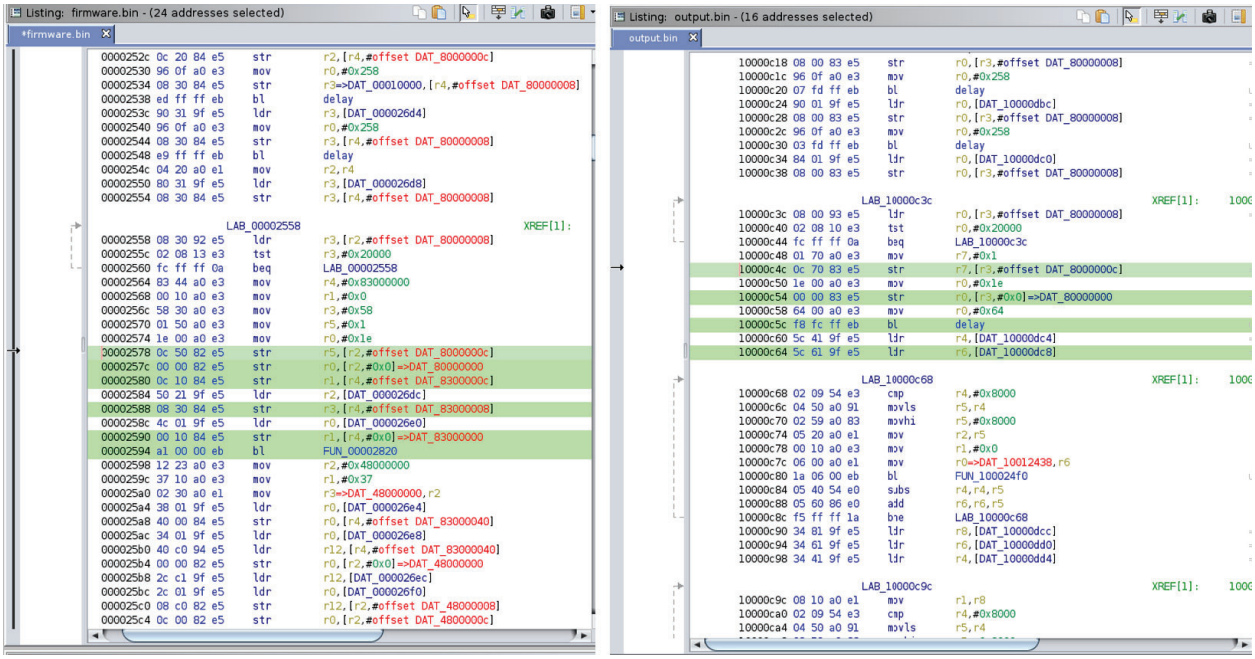

Figure 19: Jasmine Memory Dump Assembly v. Actual Firmware Assembly

1. Identify sector information
2. Identify ATA function table
   (a) Search for logical programming
   (b) Search memory for Sequential ATA Opcodes in Ghidra
   (c) Each function points to the implementation of the ATA command
3. Analyze ATA Unlock implementation

For the Jasmine, it was unnecessary to identify the sector information. The firmware files were placed sequentially in memory. Ghidra was able to properly disassemble and decompile the binary. The first step taken was to identify the ATA Opcode table. An example of the ATA opcodes are shown in Fig. 8. The process taken to identify the ATA opcode table in the Jasmine firmware was to look for logical programming. A logical programmer would implement the opcodes from Fig. 8 in sequential order in an array. The function pointers would be 8 byte pointers allocated next to the Opcode array. After identifying the ATA function table, the ATA Unlock function was located and analyzed. In order to identify the presence of PBKDF2, the constants used in SHA256 were located. The PBKDF2 function can be identified by searching for parent functions of the SHA256 function.

After successfully reverse engineering and analyzing the Jasmine firmware, this method was applied to the Crucial family of SSDs. First the firmware is needed. The firmware update can be downloaded from the website. After obtaining the firmware, the same search for logical programming is applied to the Crucial SSDs. The functions were analyzed according to the [21].

7 Results

In this section, we discuss our power analysis results and hardware debugging ports results on the Jasmine Board and on commercial devices to include Crucial, SanDisk, Micron, and Samsung SSDs.

7.1 Power Analysis

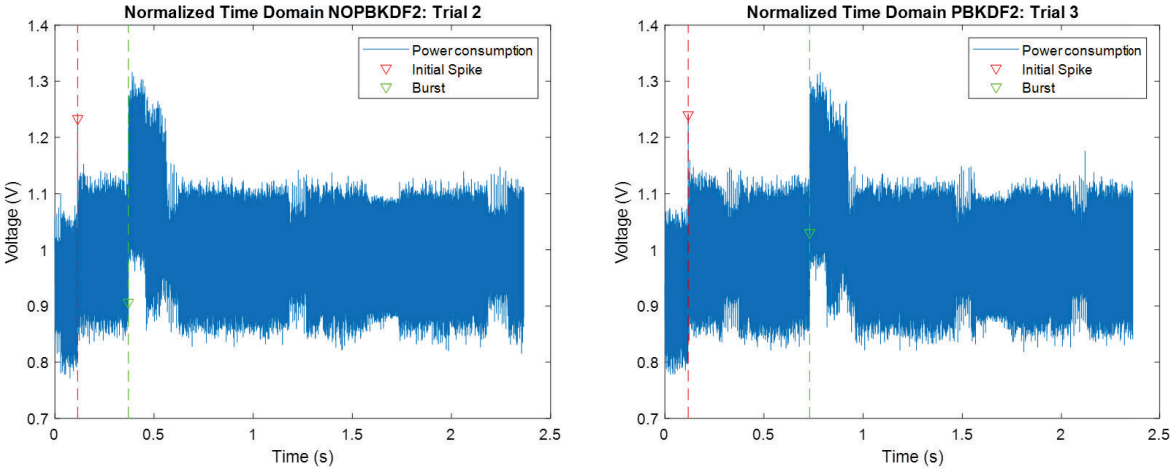

Figure 20: Time Domain Feature: Firmware with PBKDF2 v. No PBKDF2

In this section, we enumerate the results from following the power analysis methods in Section 6.2 for the Jasmine Board, the Crucial family of SSDs, and other manufacturers.

The results from the Jasmine data collection are shown in Fig. 20 in the time domain and Fig. 21 in the frequency domain. Analyzing the power consumption trace in the time

domain, the power consumption starts with the Jasmine off at 0 V. After the Jasmine turns on at time 0, the first function to run on the Jasmine is the Init_Jasmine function. This calls the Ftl_Open function and the Sata_Main function. In the Ftl_Open function, the AES data and key derivation structures are initiated. This is the area of the power consumption that will contain the differences in key derivation process. One obvious feature is the plateau after the initial spike. The plateau in the power consumption trace is much longer in the power consumption of the firmware with PBKDF2 than the firmware with a simple if statement.The feature is calculated by the the difference in time from the initial spike to the the initial burst as shown in Fig. 20. The initial spike is calculated by finding the peaks above the mean level of the signal. The burst is calculated by computing the RMS in a sliding window of the length of the burst. The beginning of the burst is found by subtracting the window length from the max of the sliding window RMS.

The second feature we are classifying on is in the frequency domain. From the time domain, the signal is Fourier transformed using the MATLAB pwelch function to get the Power Spectral Density (PSD). The PSD of the firmware with PBKDF2 is shown overlaid with the PSD of the firmware without PBKDF2 in Fig. 21. The salient feature is a difference in magnitude in the frequency range of 1.80 Hz to 1.86 Hz shown in Fig. 21. We extract this feature by integrating the PSD in the frequency range of interest.

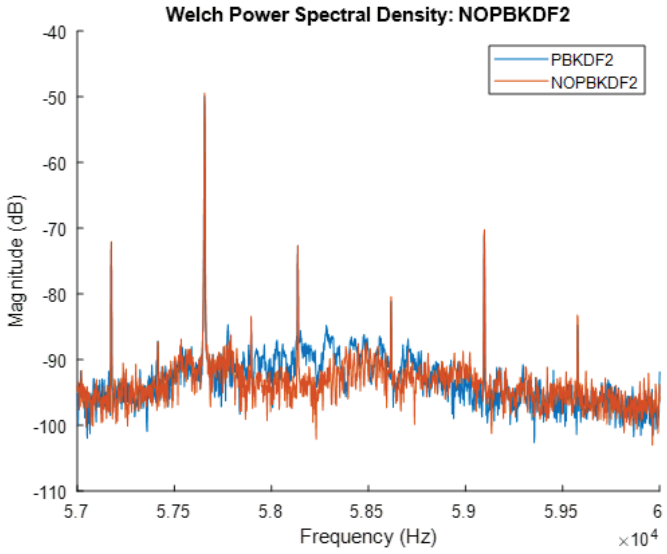

Figure 21: Power Spectral Density Feature: Firmware with PBKDF2 v. No PBKDF2

The MATLAB classifier used the PSD features and time domain features to train a kNN classifier and a binary logistic regression classifier with a 100% classification accuracy. The main plateau feature is present in every trial, when comparing a PBKDF2 plot and a non-PBKDF2 plot, which allows an classification of a 100% across all clasifiers. The success of classification of the presence of PBKDF2 on the Jasmine Board demonstrates that PBKDF2 can be detected through power analysis on a single device where the vulnerability is known.

Applying these findings to proprietary devices, this work also demonstrates the feasibility of classifying drives based on the specific SSD ATA unlock function implemented in the Crucial family. The power consumption trace of the Crucial M4's ATA Unlock command

Classifier	Accuracy
Logistic Regression	100%
Quadratic Discriminant Analysis	100%
K-Nearest Neighbors	100%

Figure 22: Jasmine Classification Accuracy of the Presence of PBKDF2

is shown in Fig. 23 and the trace of the Crucial MX200's ATA Unlock command is shown in Fig. 24. Analyzing the Crucial M4's power consumption trace, the Crucial M4's power consumption is level during the entire trial. The Crucial M4's power is level likely due to the lower power consumption of the string comparison within its implementation of the ATA Unlock function. In comparison, the Crucial MX200 contains a power spike at the time the ATA Unlock command is sent. The power spike in the Crucial MX200's trace is likely due to the SHA256 executing within the ATA Unlock command. The difference between the Crucial M4's level power consumption and Crucial MX200's spike demonstrates the ability to differentiate the string comparison implementation and the hash implementation of the ATA Unlock function. The power consumption trace of the Crucial MX300 and Crucial MX500's ATA Unlock command are shown in Fig. 25. Both contain blocks of raised power during the ATA Unlock command similar to the Jasmine power consumption traces. This is due to the PBKDF2 and AES executing withing the ATA Unlock command. The difference in power consumption traces in between the different versions of ATA Unlock command implementations demonstrates the ability to classify ATA Unlock function across different devices of the same manufacturer. To provide "ground truth," the specific SSD unlock function utilized for each drive in the Crucial family was additionally identified through reverse engineering of the proprietary firmware. This is discussed in Section 7.2.

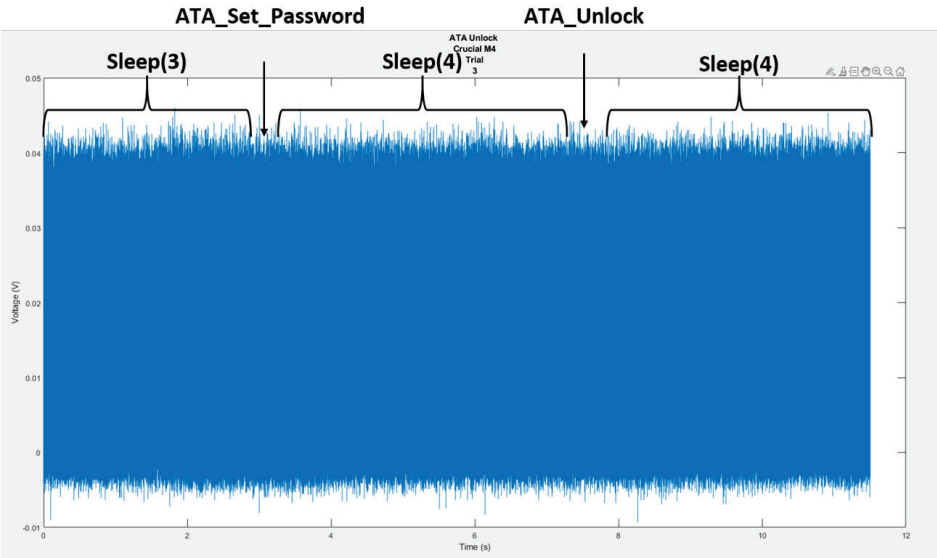

Figure 23: ATA Unlock StrCmp Power Consumption: Crucial M4

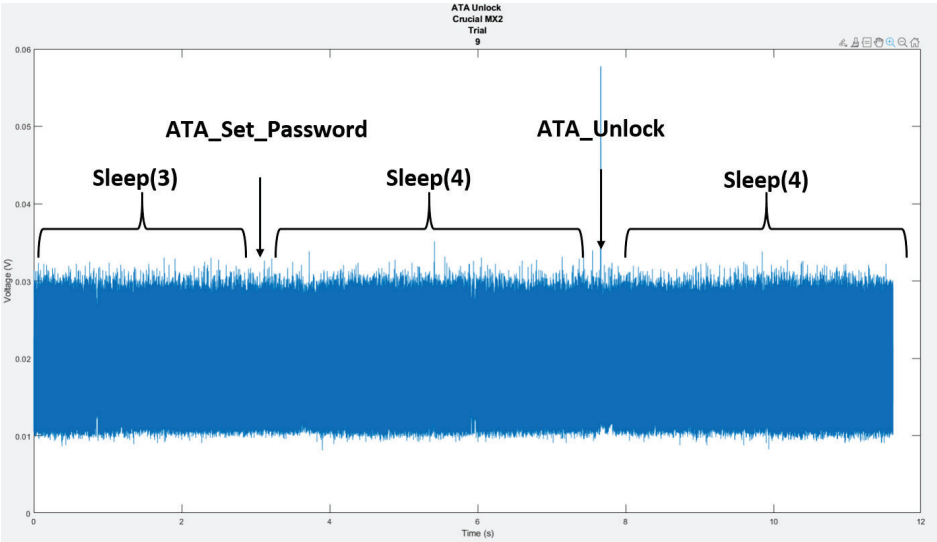

Figure 24: ATA Unlock SHA256 Power Consumption: Crucial MX200

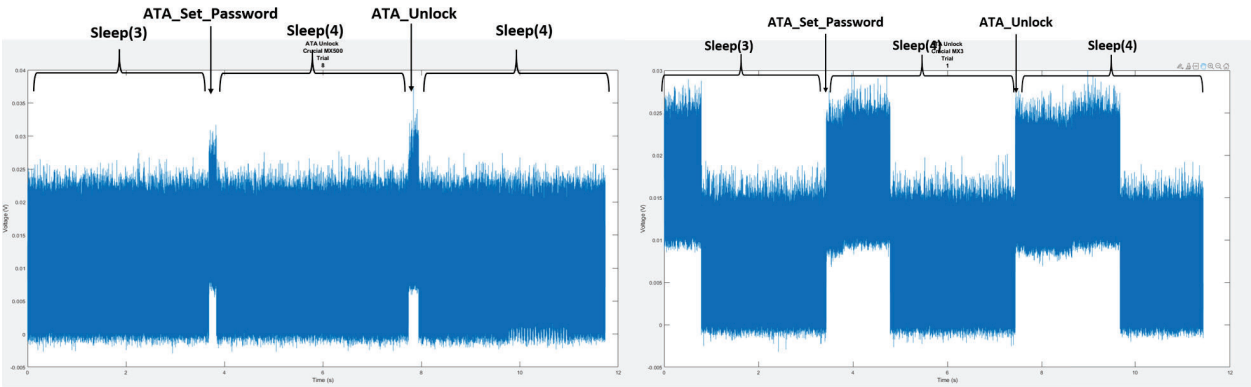

Figure 25: ATA Unlock PBKDF2: Crucial MX300 (left) and Crucial MX500 (right)

After identifying the ATA Unlock function in SSDs of the same manufacturer, this work also demonstrates the ability to classify drives of different manufacturers such as Samsung, SanDisk, Micron, and Adata. Power analysis also indicated that the password-based key derivation function was implemented on the Samsung 840, the Samsung 850, the SanDisk Ultra II, Micron 1300, and Adata SP600. The power consumption trace of these devices are similar to the Jasmine, Crucial MX300, and Crucial MX500 where it was known that these devices used a PBKDF2 function. Though Samsung 840 and Samsung 850 both contain vulnerabilities related to key management, they were not the vulnerabilities that this work was attempting to identify [4]. The Samsung 840 and Samsung 850 ATA Unlock implementation was also identified through reverse engineering. A summary of the results are shown in Fig. 26.

Identified w/ Power Analysis ✖	ATA_Unlock_StrCmp	ATA_Unlock_Sha256	ATA_Unlock_PBKDF2
Crucial M4	✖		
Crucial MX200		✖	
Crucial MX300			✖
Crucial MX500			✖
Samsung 840			✖
Samsung 850			✖
SanDisk Ultra II			✖
Micron 1300			✖
Adata SP600			✖

Figure 26: Overall Power Analysis Classification Results of ATA Unlock Function

7.2 Hardware Debugging Interface and Firmware Analysis

In this section, we enumerate the results from following the hardware debugging interface methods in Section 6.3. This work was unable to successfully connect to the Jasmine's JTAG port. However, by using the firmware binary that is compiled during the Jasmine development phase, a method was developed to reverse engineer SSD firmware. Using this process, the ATA function table in the Jasmine binary shown in Fig. 27 was identified.

From the ATA commands and function pointers, the implementation of the ATA Unlock is analyzed.

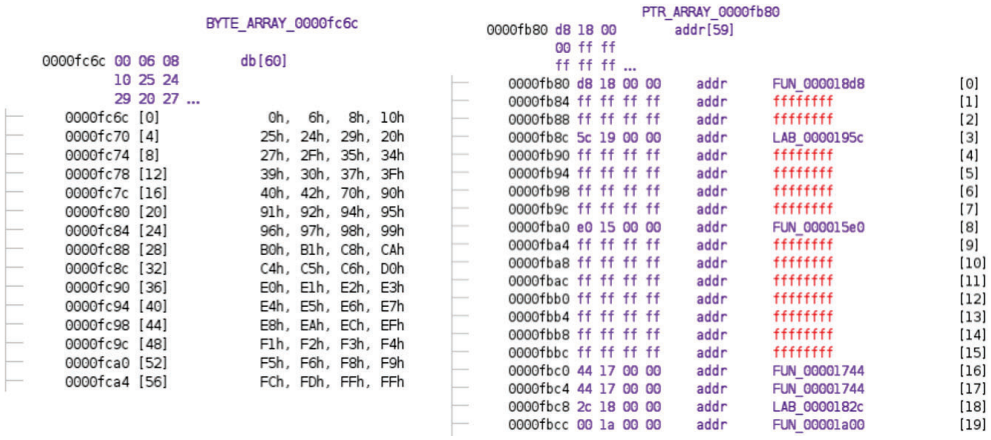

Figure 27: Identified Jasmine ATA opcode (left) and ATA function table (right)

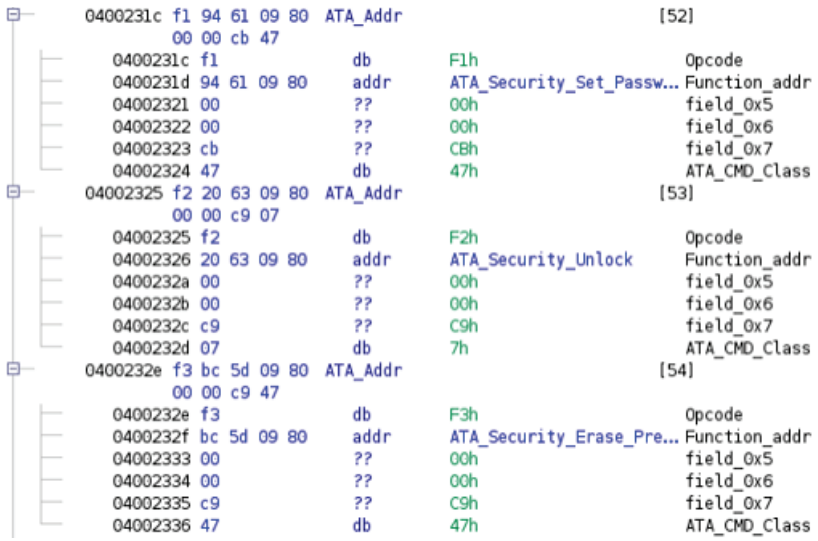

Figure 28: Crucial ATA Table Analyzed through Ghidra

Applying these findings to proprietary devices, this work also demonstrates the ability to classify drives based on the specific SSD ATA unlock function implemented in the Crucial family of SSDs. First, the Crucial MX200 ATA opcode and function table was identified. One significant difference is that in the Crucial, the ATA tables are implemented differently. All of the Crucial SSDs use an array of structs containing the opcode and function pointer shown in Fig. 28. All Crucial SSDs' ATA Unlock command begin by transferring 512 bytes from the host. Contained in these bytes are the password and whether the password is the user or master password. Then the DEK is derived from the user password according to the specific implementation. Fig. 29 shows the disassembly of the Crucial M4's string comparison implementation. The while loop comparing the incoming password to the stored password can be seen at address 0x8000ddec - 0x8000de14. If they are not equal, the command aborts.

If the two passwords are equal, then the drive is unlocked. This is the only authentication the Crucial M4 has. The Crucial M4's disassembly can be compared to Fig. 30, which is Crucial MX200's SHA256 implementation. The SHA256 function identified by this work is at address 0x800a0318. After hashing the incoming password, the Crucial MX200 compares it to the stored password has at address 0x800a0324 - 0x800a033c. Both of these are susceptible to exploits that have been published [4]. The use of the key derivation function is widely recognized as providing the strongest guarantee of data protection.

Figure 29: Disassembly of Crucial M4's ATA Unlock Function

Figure 30: Disassembly Crucial MX200 ATA Unlock Function

Identified w/ Reverse Engineering ✖	ATA_Unlock_StrCmp	ATA_Unlock_Sha256	ATA_Unlock_PBKDF2
Crucial M4	✖		
Crucial B550		✖	
Crucial MX100		✖	
Crucial MX200		✖	
Crucial MX300			✖
Crucial MX500			✖

Figure 31: Overview of Reverse Engineering Results

An overview of the results are shown in Fig. 31.

8 Conclusion and Future Work

This work analyzed three commonly implemented versions of the Security_Unlock command, which is a part of the ATA Command Set implemented by most modern disk drives: (1) a string comparison of the passwords, (2) a hash comparison of the passwords, and (3) a key derivation function that generates a decryption key based on the password. The first two versions are susceptible to exploits that have been published in prior literature, while the use of the key derivation function is widely recognized as providing the strongest guarantee of data protection. However, for the ATA command set, there does not exist a verification process to ensure that manufacturers are implementing them correctly. Neither does there exist an example of a secure implementation. Manufacturers are left to implement the ATA standard leading to mistakes and vulnerabilities in the security implementation. This work implements these three SSD unlock functions on an open source SSD board (Jasmine OpenSSD) and demonstrates the feasibility of detection and classification of these vulnerabilities through power analysis. Applying these findings to the commercially available Crucial family of SSDs, this work also demonstrates the ability to classify proprietary drives based on the specific SSD unlock function implementation. To provide "ground truth," the specific SSD unlock function utilized for each drive in the Crucial family was additionally validated through reverse engineering of the proprietary firmware. The firmware was obtained from the Crucial website and then disassembled using Ghidra, the reverse engineering tool released by the National Security Agency. Through power analysis and firmware disassembly, the Crucial MX100, MX200, and M4 drives were all found to implement the vulnerable string comparison and hash comparison functions. In contrast, the Crucial MX300 and MX500 were both found to implement the more secure key derivation function. Power analysis also indicated that the password-based key derivation function was implemented on both the Samsung 840 and the Samsung 860.

Future research in this area can immediately follow the framework developed in this work using power analysis to identify other vulnerabilities such as API Misuse, poor entropy generation, and other key management issues. One additional key management issue that can be identified through power analysis is the use of one MEK or a range of MEK as specified in the TCG Opal Specification. Another area related to this work is using ATA functions to create a power consumption fingerprint of SSD functions such as read, write, and erase. The ATA command will remove the need to infer what the drive is doing at that moment. Future work can also continue to reverse engineer commercial firmwares and identify the

exact AES implementation. JTAG continues to be a viable option to analyze the device and future work can use these hardware debugging interfaces to confirm the vulnerabilities found in this work.

References

[1] I. Grigg and P. Gutmann, "The Curse of Cryptographic Numerology", IEEE Security & Privacy Magazine, vol. 9, no. 3, pp. 70-72, 2011.

[2] A. Sari and M. Karay, "Comparative Analysis of Wireless Security Protocols: WEP vs WPA", International Journal of Communications, Network and System Sciences, vol. 08, no. 12, pp. 483-491, 2015. Available: 10.4236/ijcns.2015.812043.

[3] Debian, "DSA-1571-1 openssl – predictable random number generator", 2008.

[4] C. Meijer and B. van Gastel, "Self-encrypting deception: weaknesses in the encryption of solid state drives (SSDs)," Radboud University and Open University of the Netherlands, Tech. Report, 2018.

[5] Microsoft, "ADV180028 — Guidance for configuring BitLocker to enforce software encryption", 2019.

[6] Veracode, "Veracode's 10th State of Software Security Report Finds Organizations Reduce 'Security Debt' via Devsecops, Special Sprints", 2019.

[7] The Penetration Testing Execution Standard. (2012). Retrieved January 08, 2020, from http://www.pentest-standard.org/index.php/Main-Page

[8] Penetration Testing Framework 0.59, [online] Available: http://www.vulnerabilityassessment.co.uk/Penetration%20Test.html

[9] Information System Security Assessment Framework, [online] Available: https://ht.transparency.tools/FileServer/FileServer/whitepapers/issaf/issaf0.2.1A.pdf

[10] OWASP Testing Guide v4, [online] Available: https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_ofContents.

[11] Karen Scarfone, Technical Guide to Information Security Testing and Assessment.

[12] S. Arzt, S. Nadi, K. Ali, E. Bodden, S. Erdweg, "Towards secure integration of cryptographic software", Proceedings of the 2015 ACM International Symposium on New Ideas New Paradigms and Reflections on Programming and Software (Onward! 2015), 2015.

[13] FIPS Publication 197, "Advanced Encryption Standard", November 26, 2001.

[14] N. Karimi, 'Introduction to Cryptography', University of Maryland, Baltimore County, 2017.

[15] D. Lazar, H. Chen, X. Wang, and N. Zeldovich. "Why does cryptographic software fail? A case study and open problems." In Proc. of the ACM Asia-Pacific Workshop on Systems (APSys), pages 7:1–7:7, 2014.

[16] The MITRE Corporation. Common vulnerabilities and exposures (CVE). http://cve.mitre.org/

[17] NIST Special Publication 800-57, "Recommendation for Key Management Part 1: General (Revision 3)", January 2012.

[18] M. Egele, D. Brumley, Y. Fratantonio, C. Kruegel, "An Empirical Study of Cryptographic Misuse in Android Applications", Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 73-84, 2013.

[19] Wikipedia: Block cipher mode of operation, [online] https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

[20] NIST, "True Randomness Can't be Left to Chance: Why entropy is important for information security".

[21] C. Stevens, "AT Attachment 8-ATA/ATAPI Command Set – 4 (ACS-4)", Working Draft American National Standard Revision 14, 2016, [online] Available: http://www.t13.org/documents/UploadedDocuments/docs2016/di529r14-ATAATAPI_Command_Set_-_4.pdf.

[22] TCG storage security subsystem class: Opal specification version 2.01, 2015.

[23] P. C. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," J. Cryptographic Engineering, vol. 1, no. 1, pp. 5-27, 2011.

[24] "Jasmine openssd technical manual," http://www.openssd-project.org,The OpenSSD Project, 2011.

[25] S. B. Diagram, "http://codecapsule.com/wp-content/uploads/2014/02/ssd-architecture.jpg."

[26] J. Shey, R. Rakvic, H. Ngo, O. Walker, T. Tedesso, J. A. Blanco, and K. Fairbanks, "Inferring trimming activity of solid-state drives based on energy consumption," in 2016 IEEE International Instrumentation and Measurement Technology Conference Proceedings, May 2016, pp. 1–6.

[27] J. Canclini, J. McMasters, J. Shey, O. Walker, R. Rakvic, H. Ngo, and K. D. Fairbanks, "Inferring read and write operations of solidstate drives based on energy consumption," in 2016 IEEE 7th Annual Ubiquitous Computing, Electronics Mobile Communication Conference(UEMCON), Oct 2016, pp. 1–6.

[28] J. Melton, R. Rakvic, J. Shey, H. Ngo, K. Fairbanks, J. Blanco, D. Brown, and R. Shumba, "Inferring file system of solid state drives based on power consumption," 2017.

[29] Z. Johnson, A. Varon, J. Blanco, R. Rakvic, J. Shey, H. Ngo, D. Brown, O. Walker, "Classifying solid state drive firmware via side-channel current draw analysis", 2018 IEEE 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), pp. 943-948, 2018.

[30] R. S. Mcdowell, H. Ngo, R. Rakvic, O. Walker, R. W. Ives, and D. Brown, "Using current draw analysis to identify suspicious firmware behavior in solid state drives," in Proc. IEEE Int. Conf. Comput. Sci. Eng. (CSE) IEEE Int. Conf. Embedded Ubiquitous Comput. (EUC), Aug. 2019, pp. 92–97.

[31] Dane Brown, Owens Walker, Ryan Rakvic, Robert W. Ives, Hau Ngo, James Shey, and Justin Blanco. 2018. Towards detection of modified firmware on solid state drives via side channel analysis. In Proceedings of the International Symposium on Memory Systems (MEMSYS '18). Association for Computing Machinery, New York, NY, USA, 315–320. DOI:https://doi.org/10.1145/3240302.3285860

[32] C. A. Gonzalez and A. Hinton, "Detecting malicious software execution in programmable logic controllers using power fingerprinting," in Proc. Int. Conf. Crit. Infrastruct. Protection. Berlin, Germany: Springer, 2014, pp. 15–27.