Journal of Interdisciplinary Undergraduate Research

Volume 11 *Journal of Interdisciplinary Undergraduate Research*

Article 5

2019

UI-SPEED: Uniquely Identifiable Self-Contained Pass-Through Enhanced Encryption Device

Alexander Ennis
alecennis@southern.edu

UI-SPEED: Uniquely Identifiable Self-Contained Pass-Through Enhanced Encryption Device

Alexander Ennis

Abstract: USB encryption is an effective method for securing data. Current available solutions are limited to host-side, drive-side, and neither-side encryption. Host-side encryption must be downloaded on every host, and drive-side encryption is expensive. Current neither-side implementations either provide weak protection or provide it at the cost of unneeded multi-step processes. For USB encryption to be portable, inexpensive, and secure, it must have a straightforward locking mechanism, be able to encrypt any drive, and be distinguishable from the encryption of every other drive. The solution proposed here adds an identifiable aspect to each drive that is encrypted, isolates the encryption from the host and the drive, and adds an on-board locking mechanism.

I. INTRODUCTION

The USB flash drive market is expected to exceed 64 billion dollars by 2021 [1]. The information held in these small devices for thousands of users can be personal or sensitive and needs to be protected. There are some encryption options to choose from: host-side software encryption, drive-side hardware encryption, or neither-side hardware encryption (also known as a pass-through adapter). All of the current options for drive-side encryption require the purchase of a specialized USB drive that encrypts itself, while the options for host-side encryption are limited to software on the possibly "untrusted operating system" [2].

Some users may need to use computers that do not have cryptographic software available. A self-encrypted USB drive solves this problem, but it is more expensive because of its encryption capabilities. A pass-through adapter solves both of these problems by allowing any computer to use the device and by encrypting the device without having to use more expensive on-board encryption hardware for each drive. Unfortunately, there are not currently any pass-through adapters that do not rely on inputted information from the host. Also, all of the current pass-through adapters use the same key to encrypt the data from each drive, which can lead to faults in the encryption reliability, as shown in Section II-B.

To solve this problem, there needs to be a uniquely identifiable self-contained pass-through enhanced encryption device (UI-SPEED). It should be uniquely identifiable, meaning it should encrypt every drive with a unique key that distinguishes it from others encrypted with the same device. It must also be self-contained so that it does not rely on information from the host for its encryption.

This scheme allows the user to only worry about one PIN number for all of their drives, while still maintaining the uniquely identifiable aspect of each separate drive due to each being encrypted with a different key.

II. BACKGROUND

A. Competitive Analysis

The current solutions to USB flash data encryption come from three main sources: the host, the drive, or a device in between them. These are host-side, drive-side, and neitherside, respectively. The host relies on encryption software that is usually run by an operating system to secure the data, while drive-side and neither-side encryption rely on an embedded processor running a cryptographic algorithm on either the drive itself or a device in between.

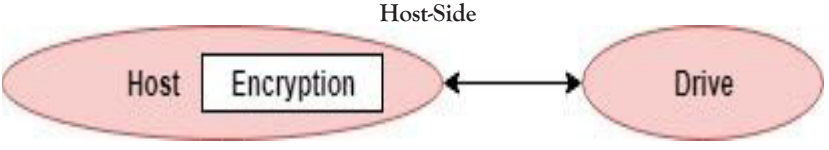

Fig. 1. Encryption with the host holding the cryptographic function isolates the encryption from the drive.

1) Host-Side: Encryption software is only as good as the operating system or processor that it runs on, as shown with the problems with Windows XP [8] and the Spectre/ Meltdown processor vulnerability [13]. Because most host-side encryption options are limited to either on-board operating system based encryption or downloaded software, both of these options are insecure simply by the nature of them being run by a host [14], [15]. Since not all operating systems have on-board encryption, every host without it must download encryption software to have USB security.

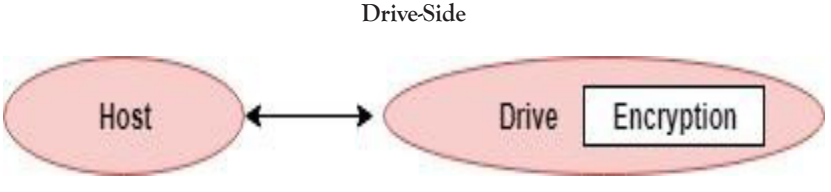

Fig.2. Encryption on the drive-side isolates the encryption from the host.

2) Drive-Side: There are many products for drive-side encryption,, with most offering encryption with AES-256. These
drives allow for both on-board and host-side key input. For example, the Kingston Datatraveler 2000, uses an on-board key input, while the PNY Sentry uses host-side key input. Driveside key input is usually done with a PIN-pad or some biometric, like a fingerprint. Fingerprints, due to their nature and continued use on the device, are easily spoofed [9], [10].

One of the simplest, though, time-consuming, attacks on a PIN-based locking mechanism is brute force (trying all possible combinations). To mitigate this problem, some products, like Apple's iPhone, put a restriction on the number of incorrect PINs a user can enter. This incorrect PIN lockout will lock the device when a certain number of incorrect PINs have been entered. Though most products account for it, some drives do allow wear on their PIN-pad button. This can help an attacker infer information about what the PIN is, making it easier to brute force. Another downside to on-board PIN-pads is that because of the small size of USB drives, individual buttons can lead to mistaken buttons presses [10]. These problems, combined with a lack of PIN display, allow for wrong input of PINs. If a USB drive does incorporate incorrect PIN lockout this can be disastrous for the user.

Furthermore, one of the most glaring detriments to using drive-side encryption is the cost. Some companies that produce USB drives also produce drives with on-board encryption. However, with a price difference of at least $120, as shown in Table I, it is more cost effective to use a universal method to encrypt all USB drives, even those without on-board encryption.

TABLE I
PRICES FOR 16GB USB DRIVES

	On-Board Encryption	
Company	*With*	*Without*
Kingston	DataTraveler 2000: $146	DataTraveler SEH9 $5.99
Apricorn	Aegis: $129	N/A
Datalocker	Sentry: $129	N/A
SanDisk	N/A	Ultra Flair: $7.99
PNY	N/A	Attache: $5.99

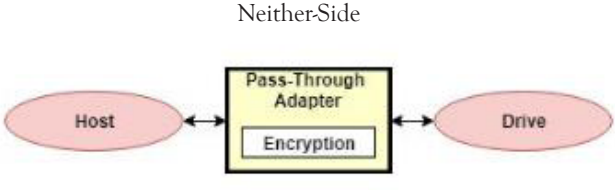

Fig. 3. Encryption with neither-side having the cryptographic function isolates both the host and the drive from cryptographic responsibility.

3) Neither-Side: There are only two products on the market for pass-through adapters: Enigma made by Enova and CipherUSB made by Addonics. Both of these are so similar in look and function that we will be treating them as the same product made by different companies and will refer only to Enigma. The steps to use this product are as follows: plug the USB drive into the Enigma module, insert both into the host, execute the enigma.exe code utility, then go through two-step initialization process.

That sets a recovery password. Then unplug and replug the pair [22]. When that is done the device encrypts and decrypts as long as it is plugged in. The product can encrypt any number of drives [22], and because of that, each is encrypted with the same recovery key. This product utilizes AES-256 to encrypt and decrypt data flowing to and from the connected USB drive. Because the device does not have an onboard key input method, it is not suitable for secure traveling, except by setting and resetting the recovery password through the host.

The current solution to the problem of neither-side encryption does not take into account the fact that for a drive to be useful it must continue to stay with the adapter. Because there is no locking mechanism on the device, any security gained from AES 256 is nullified because if an attacker can get to the drive it is not unlikely that they can also get to the device. For Enigma there is, however, a somewhat lengthy process by which a user can clear the recovery password before removing the drive and resetting it to the original recovery password after connecting the drive. This process takes more steps than necessary and still uses the host to initiate the encryption.

B. Encryption Algorithms

This section describes the encryption algorithms: DES, 3DES, Blowfish, RSA, ECC, AES, and OTP. These algorithms are, or have been, the standards in the cryptographic world. Some variations are more secure than others — one of which is provably secure, but impracticable in reality.

1) OTP: A One Time Pad (OTP) is the only, to date, perfect cryptographically secure encryption algorithm. Unlike all other algorithms, which are only computationally secure, the OTP is "information-theoretically secure" [16], which means, given the ciphertext and any amount of time and computing power, the correct plaintext cannot be known. To use an OTP, one must generate a string of truly random bits for a key. The key must be as long as the message. The key is then XOR'ed withthe plain text to produce the ciphertext.

Any violation of the conditions in [14] means that the cipher is no longer an OTP cipher [7]. Though the algorithm is proven to be secure in [4], there are some real-world problems with it. First, the encryption scheme must generate truly unpredictable keys. This can be mitigated through the use of pseudorandom number generators, but even then, those are not truly random. The next problem is that the key must be as long as the plaintext, which requires the generator to produce the keys without stretching a smaller key. The last problem stem from the first, in that the key cannot be reused. Although a random generator can, in theory, generate two of the same key, the key still cannot be used more than once, because it violates rule 3 in [14]. This aspect of random generation limits the number of plaintexts of a certain size that can be sent.

Key Reuse: With all encryption algorithms, especially OTP, there rises the problem of key reuse. If a key is reused over many plaintexts, it can lead to information leakage [21]. When these multiple ciphertexts are compared, information about the underlying structure of the key can be gleaned. Though the time to decipher the ciphertext is still greater than having the key beforehand, with enough similarly encrypted ciphertexts, security starts to fail [14], [21].

2) DES: Data Encryption Standard (DES) was the first mainstream Feistel Network-based encryption algorithm. It uses a key of 56 bits to interleave, permute, and

substitute the plaintext in blocks of 64 bits. This is done for a total of 16 rounds to obtain the cipher text. Because of the low key length the algorithm is now obsolete [3]. For this reason, it is no longer recommended by the Federal Information Processing Standards publication (FIPS) 140-2, the standards publication for recommended and proven methods for encryption.

3) 3DES: 3DES is the continuation of DES. Once DES was found faulty the creators added a 3-key system that changed the bit count from 56 to 168. Because of the 3 keys, the average time to encrypt is also multiplied by 3 [3]. Even though it is still one of the recommended algorithms in FIPS 140-2, because it takes three times as long as other block-based algorithms, it is not as suitable for most encryption needs as others.

4) Blowfish: The Blowfish algorithm is another algorithm in the same category as DES and 3DES. It also operates in 64 bit blocks and 16 rounds. The difference lies in the substitutions that the algorithm performs. With three sets of substitution boxes, Blowfish has been shown to be cryptographically secure. Another benefit to Blowfish is that because of the nature of the rounds that it performs, it is fast, only taking 26 clock cycles for one byte on some proposed systems [5]. For this reason it is well adept at hardware encryption. The algorithm was developed as an non-proprietary project by Bruce Schneier, and as such, it is not a recommended standard with FIPS 140-2 [17].

5) RSA: RSA, unlike the previous three algorithms, is a publickey cryptography scheme; meaning it uses a public key to encrypt and a private key to decrypt. It uses the idea of the intractability of factoring large prime numbers. Though it is theoretically more secure than most of the other algorithms, because of the computational overhead for generating prime numbers and the large key size required to be secure, it is not suitable for use in many encryption scenarios [23]. Furthermore, because RSA is a public/private key cryptosystem, it is not necessary for singleuser applications.

6) ECC: Elliptic Curve Cryptography (ECC) is another public/private key scheme for cryptography. It also uses a hard mathematical problem, the elliptic curve discrete logarithm problem, as its basis for encryption. Because the algorithm relies on calculating discrete logarithms, the computational overhead is only minimally smaller than that for RSA, even though the key size is much smaller [23]. This algorithm is also not needed for single-user applications because it uses public/private key cryptography.

7) AES: Advanced Encryption Standard (AES) is the primary standard for fast, reliable encryption. There are three versions: AES-128, AES-192, and AES-256. Each version refers to the key size. Like DES, 3DES, and Blowfish, it uses Feistel networks and combinations of permutations and substitutions for its encryption. Though it has the same basis, because of the larger key size it is much more secure. It was first introduced in FIPS 197 [18] and has replaced DES and another lesser-used algorithm, Skipjack, in FIPS 140-2 for both the public and private sectors [17].

III. PROPOSED PROJECT

A. Overall Functionality

User Stories: When a user enters the correct PIN, the Key-generator will generate the key and give it to both the encryption and decryption algorithms. The host will then enumerate with the drive, asking it for various identifiers and data. Once enumeration is complete, data can be transferred back and forth from host to drive.

When a user enters three incorrect PINs the Key-erase will clear the on-board key register and all other information registers. This will lock the device to prevent further brute force attacks.

When a user wishes to change the PIN, they will enter the old PIN then press and hold the Enter button. This will clear the previous PIN and allows the user to enter a new one.

B. System: software implementation

The proposed project will consist of the eight modules shown in Fig. 4: Host protocol, Drive-protocol, Encryptionalgorithm, Decryption-algorithm, Key-generation, PIN-input, Key-erase, and PIN-change. The Host-protocol and Driveprotocol modules deal with the transfer of data and commands back and forth from host to drive. The Encryption-algorithm, Decryption-algorithm, and Key-generation modules deal with securing the data. The PIN-input, Key-erase, and PIN-change modules deal with user periphery and additional functions of the device.

System Diagram

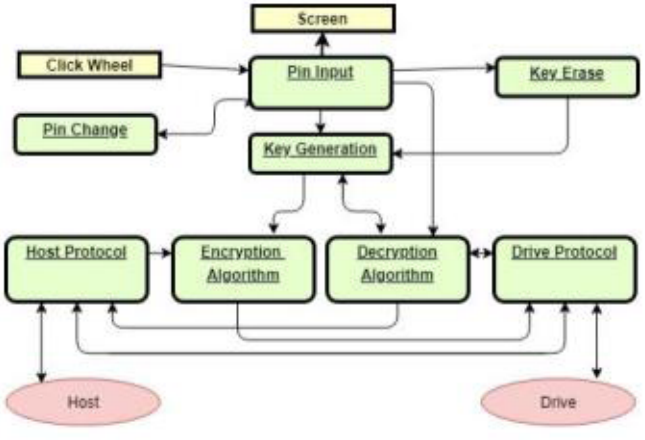

Fig. 4. The system diagram showing the flow of data and commands between the host, drive, on-board periphery, and modules for the pass-through adapter.

Modules: All modules will use a PCB board and the Beaglebone as their platform.

1) Host-protocol: The Host-protocol module will use the BeagleBone for its operations. The module will receive commands from the host and will transfer them to the Driveprotocol module. The module will receive data from the host, and that data will be given to the Encryption-algorithm module contained in the same device. The module will also receive decrypted data from the Drive-protocol module and will transfer that to the host.

2) Drive-protocol: The Drive-protocol will use the AT90USB162 microcontroller for

its operations. The module will receive responses from the drive and transfer them to the Host-protocol module. The module will receive data from the drive, which will be given to the Decryption-algorithm module contained in the BeagleBone. The module will also receive encrypted data from the Host-protocol module and will transfer that to the drive.

3) Encryption-algorithm: The Encryption-algorithm will use the BeagleBone for its operations. The module will receive data to be encrypted from the Host-protocol module. The module will encrypt the data using AES-256 with the key from the Key-generation module. The data will then be transfered to the Drive-protocol module.

4) Decryption-algorithm: The Decryption-algorithm will use the BeagleBone for its operations. The module will receive data to be decrypted from the Drive-protocol module. The module will decrypt the data using the inverse AES-256 with the key from the Key-generation module. The data will then be transfered to the Host-protocol module.

5) Key-generation: The Key-generation module shown in Fig. 5 will use the BeagleBone for its operations. The module will receive the PIN from the PIN-input module and the device identifier from the Drive-protocol module. The module will then combine and hash them with TupleHash function from the SHA-3 specification [19] to produce the key. Then the module will transfer the key to the Encryption-algorithm and the Decryption-algorithm.

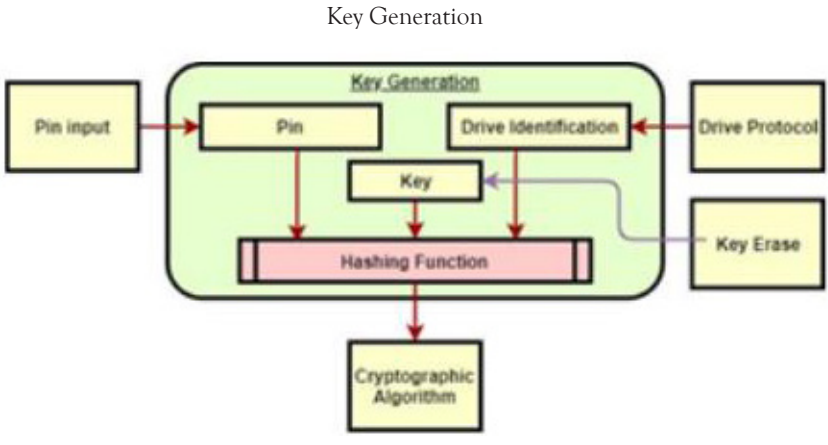

Fig. 5. Key-generation diagram showing the integration and hashing of the three parts of the cryptographic key.

6) PIN-input: The PIN-input module will use the BeagleBone, the 7-Segment display, and the Click Wheel for its operations. The module will start in the *Locked* state as shown in Fig. 6. When the device is plugged in, the state will be changed to *Ready* and the digit selection on the 7-Segment display will be set to 0. The user will scroll with the Click Wheel through the digits until the correct one is displayed. Each increment of the scroll portion of the Click Wheel increments the digit count by one.

When the correct digit is displayed, the user enters the digit by pressing Select Digit

on the Click Wheel, this returns the state to *Ready* with the digit selection set to 0. This process is repeated until all of the digits are entered, then the user presses the Enter button and the state is changed to *Unlocked.*

If the Enter button is pressed, and the PIN is incorrect, the PIN-input module updates the Key-erase module with the current *Incorrect Count* and the state returns to *Locked*.

7) Key-erase: The Key-erase module will use the Beagle-Bone for its operations. The module will start with an incorrect count of zero. As shown in Fig. 6, each time the PIN-input

State Diagram: PIN-input

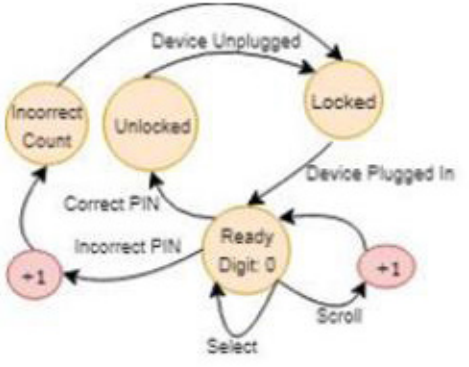

Fig. 6. The state diagram showing the possible states of the PIN-input module and the transitions between.

module sends it an update, the Key-erase module will add one to the current incorrect count and the state will return to *Locked*. When the incorrect count reaches three, the Keyerase module will clear the on-board key register and the PIN comparison register.

8) PIN-change: The PIN-change module will use the BeagleBone for its operations. The module will give and receive communication from the PIN-input module about the state of the periphery (the 7-Segment and the Click Wheel).

The module will start in the *Locked* state as shown in Fig. 7. When the user enters the correct PIN, the state will change to *Unlocked*. When the user holds the Input button for one second, the state will change to *Ready*, the PIN comparison register will be cleared, and the digit count will be set to zero. While the module is in the *Ready* state the user can enter a new PIN. The state will follow the same sequence as the one for entering a regular PIN. When the user holds the Enter button again, the state will be changed to *Unlocked*, and the new PIN will replace the old PIN in the PIN comparison register.

State Diagram: PIN-change

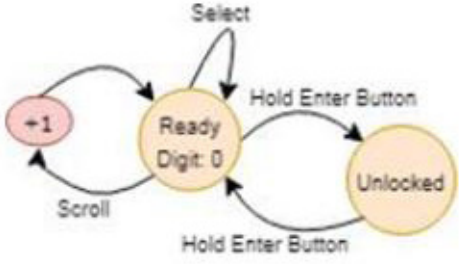

Fig. 7. The state diagram showing the possible states of the PIN-change module and the transitions between.

Distinguishability: The Drive-protocol - Decryptionalgorithm- Key-generation - PIN-input integration handles the acquisition and decryption of the drive identifier so it can be used by the Key-generation module. This is done while the drive is enumerating with the host. While the enumeration happens, the Drive-protocol will capture the drive identifier and give it to the Decryption-module, which will decrypt

TABLE II
PRELIMINARY PARTS LIST (SOME QUANTITIES ARE IN BULK)

Part	Cost	#
AT90USB162 µC	$1.89	1
BeagleBone	N/A	1
Click Wheel	$13.95	1
7-Segment Display	$0.49	1
PCB Board	$8.00	1
Male USB port	$2.50	13
Female USB port	$2.50	13
Other Board Parts	$10.00	20
TOTAL	39.33	51

it with the PIN received from the PIN-input module. Once the drive identifier has been decrypted, it will be given to the Key-generation module, which will hash it to produce the key.

C. System: hardware implementation

The hardware side of this project, shown in Fig. 8, will utilize the BeagleBone development platform for a proof of concept implementation of our solution to neither-side encryption. Using this allows us to focus on the building the system, rather than building the host USB stack from the ground up. To facilitate the device side of the system we will use an AT90USB162 microcontroller. This is because, from our research, the BeagleBone is not able to perform host and device functions simultaneously. The AT90USB162 microcontroller, along with all of the added peripherals, will be mounted

on the BeagleBone using a cape for easy access and debugging.

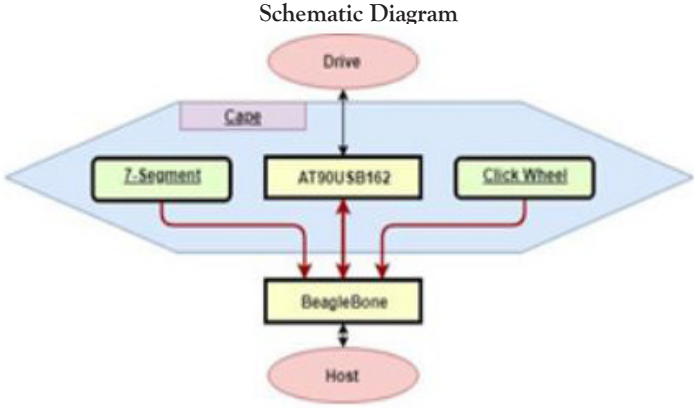

Fig. 8. The schematic diagram showing the BeagleBone development board and its added cape for peripherals.

IV. TESTING/EVALUATION PLAN

A. *Acceptance*

1) Correctness Survey: We will survey 10 people from the Southern Adventist University population. They will be given a drive and an adapter with the PIN already entered. They will be asked to perform a range of tasks: create a text file, modify the text file, read the text file, and delete the text file. Then they will be asked to plug in the drive without the adapter. After that they will report in the rubric from Table V whether or not each task was successful. We expect at least eight of

TABLE III
PROJECT PROPOSAL: SPRINTS 1-5

Sprint #	Module(s)	Module #	Time (hr)
Sprint 1	System design		4
	Schematic design		12
	Board layout		10
Sprint 2	Board fabrication		4
	Board assembly		10
	Board bring-up		3
Sprint 3	Testing: Board		5
	Host-protocol	1	7
	Drive-protocol	2	8
	Testing: Modules 1 - 2		1
Sprint 4	Encryption-algorithm	3	5
	Decryption-algorithm	4	5
	Key-generation/Hash	5	3
	Testing: Modules 3 - 4 - 5		1
Sprint 5	PIN-input	6	6
	Key-erase	7	2
	PIN-change	8	6
	Testing: Modules 6 - 7 - 8		3
	Testing: Integration / Modules		3
	Testing: Acceptance / System		2
	TOTAL		100

TABLE IV
USB TIMING SPECIFICATIONS

Request Type	Time
All requests	5s
Time for Standard Device requests without a data stage	50ms
Time between request and a device request with a data stage.	500ms
Time between data packet and successful previous packet	500ms
Time between status and the transmission of last data packet	50ms
Time for SetAddress to process command and return status.	50ms
Time for the device to change address before next request	2ms

the people to answer *yes* to all five questions. The remaining two people should answer *yes* to at least four of the questions.

B. *System*

Unless otherwise stated, all system testing will be done ten times, and nine out of ten are expected to pass.

1) *Speed (Drive USB protocol - Encryption algorithm - Decryption algorithm - Host USB protocol):* To test the speed of the cryptographic algorithm an oscilloscope will be connected to the host and drive side of the data path. We will measure the time for some specific data to propagate between the two sides through the Encryption algorithm and Decryption-algorithm. The time measured should be within 5 seconds as per the specifications from Table IV.

TABLE V
RUBRIC OF CORRECTNESS FOR THE USER

Section	Yes	No
Able to create a file		
Able to modify a file		
Able to read a file		
Able to delete a file		
Not able to read without adapter		

2) Key erasure (Key erasure - PIN-input - Key-generation): To test for key erasure, we will enter three incorrect PINs, then read the on-board key and PIN comparison registers and verify that they have been cleared. This will be done 10 times, and we expect that all of them should be cleared.

C. *Integrations*

Unless otherwise specified, all integrations testing will be done ten times, and nine out of ten are expected to pass.

1) Host-protocol - Drive-protocol: This integration will be tested by simulating the 17 requests from the Host-protocol to the Drive-protocol, and from the Drive-protocol to the Hostprotocol. After each request, we will evaluate how the other protocol responds.

We will simulate the back and forth communication for enumerating USB. The

requests will be made in the same order and speed as normally is required for drive enumeration.

2) Key-generation - Encryption-algorithm - Decryptionalgorithm: To test this integration, we will arbitrarily conjure a PIN that will be hashed with the key and drive identifier to produce a key. We will feed this key to the Encryption-algorithmand the Decryption-algorithm modules. The modules should generate the appropriate key expansion for the given PIN.

3) PIN-input - Key-erase - PIN-change: This integration will be done in two parts: PIN-input - Key-erase and PINinput- PIN-change. For the first portion of the testing we will enter three incorrect PINs. We will then verify that the onboard key and PIN comparison registers have been cleared. The second part will be tested by changing ten different PINs to new PINs.

4) Drive-protocol - Decryption-algorithm - Key-generation- PIN-input: This integration will be tested by enumerating a number of drives and verifying that the correct drive identifier is decrypted and sent to the Key-generation module for hashing.

D. Module/Unit

Unless otherwise specified, all module and unit testing will be done ten times, and ten out of ten are expected to pass. All units will be tested for specific data pertaining to their module. These tests will be performed with the μCUnit testing framework from Section IV-E and should cover all function that give and/or receive data.

1) Host-protocol: This module will be tested by having it collect and transfer various commands and data. We will pay special attention to the time that it takes the module to accomplish the transfer of commands. Because of the time sensitivity of enumeration, all timing requirements must be met. However, only the "time for Standard Device requests without a data stage" will be specifically tested, in accordance with the specifications mentioned in Table IV.

2) Drive-protocol: This module will be tested by having it receive all of the commands and responses required for enumeration and some arbitrary data. The module must be able to transfer all of the appropriate commands and data within the time specifications in Table IV, but only the "time for Standard Device requests without a data stage" will be specifically tested.

3) Key-generation: This module will be tested by sending it 30 combinations of PIN, on-board key, and device identifier. We expect the internal hashing function to generate the correct key 30 out of the 30 times.

4) Encryption-algorithm: This module will be tested by giving it the initialization from [18] and verifying that, for each of the rounds and sub-functions, the values presented match the values given in [18].

5) Decryption-algorithm: This module will be tested similarly by giving it the initialization from [18] and verifying that the values presented both match the values given in [18] and are the inverse of the values from the previous test.

6) PIN-input: This module will be tested by inputting 10 different PINs and verifying that, for each, the module
changes its state to Unlocked. While being inputted, each of the PINs inputted will be "backspaced" at least halfway and then retyped to show the backspace functionality.

7) Key-erase: To test this module we will simulate 10 sets of 3 incorrect keys. We will

verify that the on-board key and PIN comparison registers have been cleared.

8) PIN-change: This module will be tested by entering the old PIN then changing the old PIN to the new PIN. We will then lock the device and verify that the new PIN unlocks the device.

E. Testing Frameworks

1) Oscilloscope: An oscilloscope will be used to test the physical connections between the modules such as Driveprotocol and Host-protocol. In particular, this will be used to test the speed at which the pass-through adapter operates.

2) μCUnit: μCUnit is a unit test framework for microcontrollers. It will be used to test the workings of the other modules, submodules, and units, such as parts of the Encryptionalgorithm, Decryption-algorithm, and Key-generation modules.

V. CONCLUSION

Because USB drives are so prevalent, easy encryption and security are in high demand. Since host-side encryption is not viably portable and drive-side encryption is expensive, we look to neither-side encryption. To solve the problems with current neither-side implementations of USB encryption, this paper proposes a proof of concept for a device that will allow a user to distinguishably secure their data with an on-board PIN. This effectively removes security responsibilities from both the host and the drive

REFERENCES

[1] "Globalflash memory market 2013-2021 — Statistic," *Statista*, 2018. [Online]. Available: https://www.statista.com/statistics/553556/worldwideflash-memory-market-size/.

[2] M. Henson and S. Taylor, "Memory Encryption - A Survey of Existing Techniques" *ACM Computing Surveys*, vol. 46, no. 4, pp. 1–26, Mar. 2014

[3] G. Singh and Supriya, "A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security," *International Journal of Computer Applications*, vol. 67, no. 19, pp. 33–38, Apr. 2013.

[4] M. Rosulek, *The Joy of Cryptography*. Corvallis, OR: Drafted, 2018.

[5] N. Valmik and V. K. Kshirsagar, "Blowfish Algorithm," *IOSR Journal of Computer Engineering*, vol. 16, no. 2, pp. 80–83, 2014.

[6] S. Manku and K. V. Vasanth, "BLOWFISH ENCRYPTION ALGORITHM FOR INFORMATION SECURITY," *ARPN Journal of Engineering and Applied Sciences*, vol. 10, no. 10, pp. 4717–4719, Jun. 2015.

[7] C. E. Shannon, "Communication Theory of Secrecy Systems*," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, Oct. 1949.

[8] O. Alhazmi, Y. Malaiya, and I. Ray, "Security Vulnerabilities in Software Systems: A Quantitative Perspective," *Data and Applications Security XIX*, S. Jajodia and D. Wijesekera, Eds. 2005, pp. 281–294.

[9] P. Swierczynski, M. Fyrbiak, P. Koppe, A. Moradi, and C. Paar, "Interdiction in practice—Hardware Trojan against a high-security USB flash drive," *Journal of Cryptographic Engineering*, vol. 7, no. 3, pp. 199–211, Jun. 2016.

[10] D. Jagos, "Security analysis of USB drive," M.S. thesis, Department of Digital Design, Prague, 2018.

[11] N. Tihanyi, "Fault-injection based backdoors in Pseudo Random Number Generators*," *Studia Scientiarum Mathematicarum Hungarica*, vol. 52, no. 2, pp. 233–245, Jun. 2015.

[12] A. Muffett, et al. "RSA-155 Is Factored!" RSA Laboratories,1999, web.archive.org/web/20061230233723/http://www.rsasecurity.com/rsalabs/node.asp?id=2098.

[13] A. A. Chien, "Computer architecture: Disruption from above," *Communications of the ACM*, vol. 61, no. 9, pp. 5–5, Sep. 2018.

[14] D. Rijmenants, "THE COMPLETE GUIDE TO SECURE COMMUNICATIONS WITH THE ONE TIME PAD CIPHER," *Cipher Machines & Cryptology*, ed. 7.5, Jun. 2018, pp. 1–27.

[15] K. Nayak, D. Marino, P. Efstathopoulos, and T. Dumitras¸, "Some Vulnerabilities Are Different Than Others," in *Research in Attacks, Intrusions and Defenses*, 2014, pp. 426–446.

[16] Christian Matt and Ueli Maurer, "The One Time Pad Revisited," *ETH Zurich: Swiss National Science Foundation*, pp. 1–5. Elliptic Curves and Cryptography

[17] SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES, FIPS PUB 140-2, 2001.

[18] *Announcing the* ADVANCED ENCRYPTION STANDARD (AES), FIPS PUB 197, 2001.

[19] *SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and Parallel-Hash*, NIST SPEC PUB 800-185, 2016.
[20] *Recommendation for Block Cipher Modes of Operation*, NIST SPEC PUB 800-38A, 2001.
[21] *Using AEC CCM Mode with IPsec* ESP, RFC 4309, 2005.
[22] "Securing Data at Rest Safely and Easily," Enova Technology Corporation, 2012 [Online]. Available: http://www.enovatech.net/support/download Enigma%2 White%20Paper%2008032012.pdf
[23] "Elliptic Curve Cryptosystems," RSA Laboratories T, 1997. [Online]. Available: http://networkdls.com/Articles/elliptic curve.pdf
[24] "Elliptic Curve Cryptography," Auto-ID Labs, The University of Adelaide, 2014 [Online].Available:http://cocoa.ethz.ch/downloads/2014/06/None AUTOIDLABS-WPHARDWARE-026.pdf White:ECC

APPENDIX A
REQUIREMENTS

Functional

1) As a drive holder, I want to have a transparent interaction between host and drive (2).
2) As a drive holder, I want to be able to interact with my files (2).
3) As a drive holder, when I plug in a device and drive without the PIN it will not enumerate (2).
4) As a drive holder, When I plug in the device and drive with the PIN it will enumerate (2).
5) As a drive holder, I want the Encryption-algorithm to only encrypt data (4).
6) As a drive holder, I do not want the Encryption-algorithm to encrypt commands (4).
7) As a drive holder, I want the whole drive encrypted, not just the files (4).
8) As a drive holder, I want my data to be decrypted before getting to the host (4).
9) As a drive holder, I want the system to clear any relevant data from buffers/RAM etc. (5).
10) As a drive holder, I want to be able to input a PIN on the device (5).
11) As a drive holder, I want my PIN to be outputted (5).
12) As a drive holder, I want the on-board key to be deleted when too many incorrect PINs are entered into the device (5).
13) As a drive holder, I want to be able to change my PIN (5).

Nonfunctional

1) The system shall have a Click Wheel selection system for scrolling through digits and selecting them (1).
2) The system shall output the current digit selection (1).
3) The system shall output the previous digits (1). 32
4) The system shall allow for 4 to 32 digits for a PIN (5).
5) The system shall use one LCD display to output the current digit selection (1).
6) The system shall allow the user to scroll back through the previous selected digits (5).
7) The system shall generate the encryption key from three sources and hash it (4).
8) The system shall get the name of the USB drive from the drive (4).
9) The system shall use AES-256 for encryption (4).
10) The system shall encrypt data going from host (4).
11) The system shall not encrypt USB commands going to the drive (4).
12) The system shall use AES-256 for decryption (4).
13) The system shall decrypt data going from drive (4).
14) The system shall not decrypt USB responses going to the host (4).
15) The system shall utilize USB 2.0 protocols (2).
16) The system shall be totally self-contained, requiring no unnecessary host interaction (1).

Tyson Hall, Ph.D, Professor of Computing, acted as supervisor and consultant for this undergraduate research project and provided guidance to the student author.