Thesis Title

Ricardo Manuel Teixeira Pires
ricardo.teixeira.pires@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

September 2020

Abstract

Nowadays portable flash devices like USB drives and SD cards are used globally. They are used to carry personal or corporate data without any protection. These devices are easy to get lost or stolen. Thus, an attacker can easily access the potential sensitive data stored in them. This study aims to tackle this problem, with the use of an external micro-controller to handle security operations and data management. We defined the system's architecture and how each of its components benefits from each approach and algorithms analyzed. From the initial prototype we developed further tuning and improvements in terms of functionality and performance, to abide by the requirements and function within our hardware restrictions. The performance results show the compromise and inverse relation between the amount of security operations performed and the system's performance, further accentuated by the tight hardware restrictions. In conclusion, this work is a significant step towards achieving digital security in flash based devices with just the use of software and cheap hardware. This work offers a solution with more digital security robustness than market alternatives which provide no integrity or authenticity while charging steep prices for limited amounts of storage.
Keywords: Flash memory; File System; SD Card; Cryptography; Micro-controller;

1. Introduction

In today's world portable devices likes universal serial bus USB drives or SD cards are used globally by millions of people. These devices offer a good choice for storage since they can hold several gigabytes of data. The storage capacity of these devices keeps increasing every year, as such its expected that they remain popular. At the same time, they are very small in size which gives them great usability since they can transport large amounts of information, while providing great comfort for its users.

However there are some problems with this kind of devices. They provide no digital security whatsoever. All the information is stored without any digital protection, so anyone who has possession of the device can access all of its data. Since these devices have small dimensions, they are easy to lose or to get stolen.

The objective of this project is to create a secure portable storage device with the use of a microprocessor. The idea is to implement a software solution on a micro-controller, so that all the files that are stored in the portable storage will be secure against any attacks in case it gets lost.

2. Background

The Background section features algorithms that exist today and can be used to accomplish different objectives of this project. The algorithms are analysed, pros and cons are presented and some brief comparisons are made.

2.1. File Systems

The FAT32 file system is one of the most used file systems for flash media storage, as such compatibility is it's major advantage. However this file system has a severe downfall in this project. This is due to the fact that the Arduino Library only supports the 8.3 file name convention. This limits the possibilities of encryption of the name of the file. The fact that the filename can only use 8 characters maximum compromises the confidentiality of the files' metadata.[6]

The exFAT file system is a newer version of the FAT file system. It has several advantages over the former. It allows the media storage to be over 32 GB, has a faster storage allocation, and allows file sizes bigger than 4 GB.[18]

The F2FS is a new Linux file system created by Samsung and designed specifically to perform well on flash storage devices, by increasing performance and lifetime. In the research conducted it is shown to outperform the Ext4 file system. To achieve this, the design had some key aspects to it. It avoids unnecessary and costly data copying by matching the on-disk data structures to the NAND flash memory layout. It has a cost-effective

index structure. Supports multi-head logging and adaptive logging which turns random writes into sequential ones or uses multi-threaded writing in high storage utilisation.[16]

Flash memory has a drawback called the WAF. In short this happens when a page of memory has to be modified, at the physical level the entire block has to be re-written. If one page has 8KB of memory and a block 64KB of memory, than the WAF is 8. In other words it takes 8 times more writes than what the file system sees at a virtual level. If poor optimisation algorithms are used to minimise the WAF then the SD card may have a significantly shorter life cycle, since the memory will wear down much faster.[12]

2.2. Encryption Algorithms

We focused on symmetric algorithms since these are more efficient, and designed to encrypt large files. Symmetric encryption uses one key to encrypt a file, this key is the same used in encryption and decryption. The algorithms usually cipher blocks of 128 bits which means they must be combined with a cipher mode algorithm. Most are compatible with the most common ones like ECB, CBC, CFB and others.

Asymmetric encryption has some disadvantages like the fact that it can only encrypt plain-texts up to 2048 bits, in the case of the RSA. It would be necessary to construct a cipher-block mechanism to overcome this limitation. Keys are also larger, NIST recommends keys of at least 1024 bits, which is 8 times the necessary size for symmetric encryption algorithms like AES also known as Rijndael.[5] Since users can have files that have gigabytes in size, it is preferable to use symmetric encryption which can handle these sizes.

The first algorithm to consider is 3DES. DES was a very fast and secure algorithm for its time, but now has become vulnerable due to the increase of CPU processing power. Triple DES was then designed to take advantage of DES strengths and still provide a secure encryption. There are two ways of using DES to use three different keys and encrypt with every single one of them, or to use two keys and perform a sequence of encrypt with key one, decrypt with key two and encrypt again with key one. The first technique is vulnerable to key related attacks as demonstrated in [15], and Tripe DES with only two keys also has also been considered a weak cipher by NIST. [4]

The NIST and industry approved encryption algorithm is AES or Rijndael. Rijndael was the candidate chosen to become the Advanced Encryption Standard since it was the best overall algorithm. It offered great security while also achieving an excellent trade-of with performance, since it had the best performance overall. However it is still important to check if Rijndael still has the best performance in microprocessors, or if another of the finalists has the edge in this field.[9]

According to a study conducted by Toshiba Technology Centre[19] to evaluate the performance of the different AES candidates on a smart-card using software based implementations in C language, it is clear that the Rijndael algorithm, commonly known as AES, has the best performance in smartcards using a software implementation. It can be implemented with extremely limited resources since it only consumed 66 bytes of RAM and required 980 bytes for storage. It is also the fastest algorithm by far. The MARS algorithm required a complex and difficult implementation to work on a smart-card, due to its complex structure. MARS also had other disadvantages of being implemented on a smartcard that causes the algorithm to be vulnerable to timing attacks.

2.3. Ciphers

CBC uses an IV to combine (XOR) with the first plain-text, then it ciphers the combination and uses the resulting cipher-text as the next IV. The IV can be known, but it's generation should be randomised and a different IV should be used for each encryption. Since the next block to encrypt depends on the previous one, parallelism is not possible in CBC. It is possible in the decryption process though, since all blocks are available. CBC is prone to padding oracle attacks, which try to exploit the padding that the algorithm requires.[10]

The XTS is a cipher block algorithm used to encrypt entire disks. It is based on XEX. XEX uses two keys. Key one is used to encrypt XORed plain-texts. The other is used to XOR plain-texts and the cipher-texts resulting from the encryption with the first key. For each block, to compute the XORs, it is necessary to compute a derived key from the second key. Differing from XEX, XTS uses cipher-text stealing to allow for sector sizes with non-divisible block sizes. It must still use two keys. The major advantage of this block cipher is the fact that it allows random writes without need to re-encrypt the entire file, which is particularly useful for log files for instance.[17]

3. State of the Art

The TCFS is a solution proposed for file transfer between a user and a remote server. The main idea is to have multiple users connect to a remote file server to access their files, since their workstations would have very limited disk space. This file system guarantees that the files are not readable by users other than the owner of the file, including the superuser of the file system. Communication between the user and the remote server is

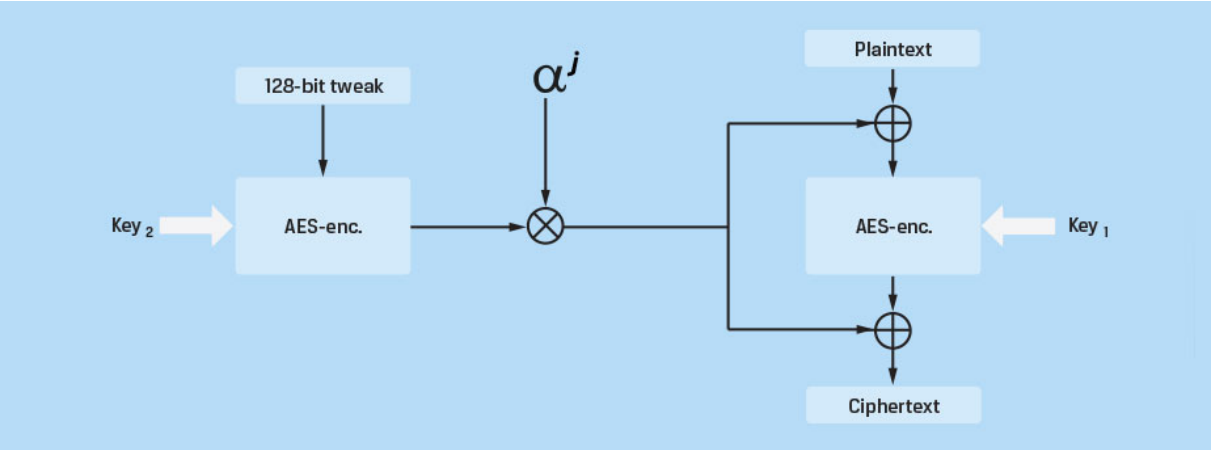

Figure 1: XTS cipher mode - Extracted from [2]

secure.[8]

Keypad is a file system designed to offer security in remote devices such as laptops, USB drives, and others. It also provides the user with an auditing mechanism to detect whether or not there have been access attempts to the files in case the device gets lost. Furthermore, in case the remote device is lost, the user still has the capability to disable future file accesses even in the absence of device connectivity. Keypad uses a similar strategy applied in the TCFS. It uses different keys for each file and stores those keys on a remote auditing server. After the transaction, the key is securely deleted. When a user wants to access a certain file, he has to download a specific key from the server. All file operations made by a user are registered in a log file. This log allows a user to audit his remote devices and check if there have been attempts to discover the password. As a last resort the user can chose to delete all random keys for the lost device which also makes the main password obsolete.[11]

Iris is a file system created to support workloads of large enterprises while offering authentication, file integrity and freshness. The system sits in between the client and the cloud. A network of distributed ”portals” are responsible for managing the inbound and outbound data from the company. These portals are composed of different modules, different types of cache, audit modules and others. These portals store in cache data and metadata recently accessed by users, they also check the integrity and freshness of all files. For freshness the system also authenticates the version of each block. This prevents rollback attacks where users are presented with an earlier state of the file. [21]

PurgeFS is a file system designed to make sure that deletion of data is permanent and is impossible to recover or restore it. It’s designed to be able to be integrated with any running Operative System. This system overwrites file data and metadata more than once using a number of patterns, like the ones approved by NIST. PurgeFS can erase data in asynchronously or synchronously. In synchronous mode it waits for all the overwrite operations to conclude, in asynchronous mode it remaps data pages to a temporary file and overwrites them later. It can also eliminate only sections of a file. The system is highly portable since it was designed as a file system extension and can easily be added to most existing file systems. It performs data purging almost immediately after a file’s deletion. In non-workload environments it is shown to be non-intrusive and quite efficient due to its asynchronous mode.[13]

A Secure Flash Card Solution for Remote Access is one solution to the problem of guaranteeing security in a SD card. This solution relies on two main components, a tamper resistant module embedded in the SD card, and a server for authentication. The tamper resistant module contains user authentication information and cryptographic keys. The communication between the user and the SD card is also encrypted. In this system, the client must first authenticate himself in the server. After establishing a secure connection, the SD card proceeds to transmit it’s signature to the server, to which the server replies with the encryption key. It utilises asymmetric encryption for the signature and symmetric encryption for the data files. This system also has an additional feature. In the event of brute forcing attempts, of the security areas, it will start to erase all data within the file. This process cannot be stopped once set in motion, even by cutting the power to the card since it will resume once the power is reestablished.[14]

In ”The State of Embedded-Device Security” published in 2012, the article reports that millions of embedded devices connected to the internet have little to no security and can easily be hacked by malicious individuals. It also denotes that mobile devices are going to start to be targeted

more often due to their increase in computational power.[22]

In 2016 a study that used Optical Fault Injection attacks against the flash memory of smart cards showed that it was possible to extract sensitive information from these devices. These attacks were carried out using a device capable of analysing the power consumption of the card, locating the flash region and control logic region, identifying sensitive points to the laser and then performing the Bumping Attack [20]. By analysing the power consumption of the card it is possible to control the time to execute attacks. Then the physical location to conduct these attacks is mapped by using optical fault injection, this is done with lasers. The final step is to confirm that is it possible to change the value of bytes of data when it is being transferred from flash memory to registers.[7]

Today in 2020 the current solutions for this problem are encrypted USB drives, like the Kingston IronKey D300 or the Aegis Secure Key 3z. These USB drives offer data encryption with AES256-XTS and anti-tamper protection with the use data purging in the case of too many failed logins. However these drives offer no data integrity protection and also have very steep prices, even with small storage spaces like 4GB.[3] [1]

4. Architecture

The objective of this project is to provide a low cost system that can guarantee digital security to flash based portable memory devices, like SD cards or USB flash drives. It is necessary to first develop an architecture of the system. the architecture should encompass three main criteria.

- Functionality;
- Digital Security;
- Ease of use.

Functionality is the most basic requirement for this project. It guarantees that the system is usable. The user must be able to issue commands and it must be possible to read or write to the SD card. This involves studying possible methods of transferring files, methods of communication between the microprocessor and the user. It is also necessary to study how the SD card manages file allocation and how the file system manages the files. The most basic functions that a user should be able to do are:

- Read;
- Write;
- Delete;
- List;

Digital security is guaranteeing that the system protects the user data adequately. This involves studying which cryptographic algorithm is best suited for encryption. Studying ways of providing freshness, integrity and authenticity to the files. Also how the user should authenticate himself to the system.

Ease of use is how the user interacts with the system. It is a criteria dedicated to studying ways to increase the comfort of the user in interacting with the system. It has less priority then the other two but it can enrich the work done since the easier the system is to use, the more people can use it.

Figure 2 represents a diagram of the functions that are necessary to implement. The functions on the client are mere remote calls to the corresponding functions on the Arduino. They do not perform any critical actions. The functions on the user end only transmit or receive information, commands, or parameters like file size, filename, etc. The Arduino is responsible to execute all the critical actions such as encrypting files, writing them in an SD card or validating the user login. The SD card already has its built in functions, for this project the read, write and delete functions are the most important. The Arduino interacts with the SD card with the use of a file system. This file system is responsible for managing virtual block allocation and making API calls to the SD card.

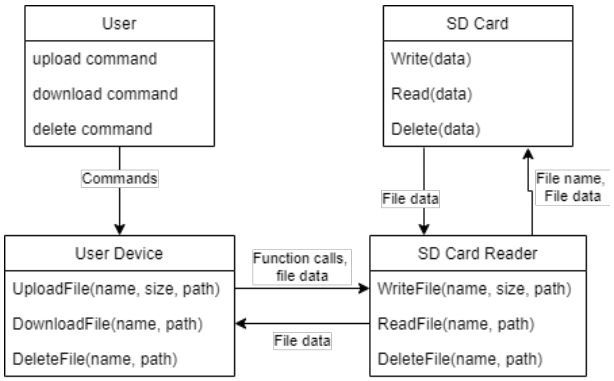

Figure 2: High Level Concept

4.1. High Level Concepts

As depicted in figure 2 the four main components of this solution can be observed. First the user which interacts with the system. The user only interacts with his user device (smartphone, laptop, etc..) by sending commands. this can be done by typing them in a terminal window, or by simply dragging and dropping files similarly to how a USB drive is used today. This depends on the level of development of the end-user interface.

The user device is the device that the user chooses to interface with the system. It does not

do any processing related to the data. It only invokes functions on the SD card reader and sends or receives file data.

The SD card reader is the name used to designate the middle system between the SD card and the user device. The SD card reader, is the microcontroller that will do the bulk of the data management and processing. It is responsible for deciding how the files are written in the SD card by implementing a file system, encrypting the data before it gets written, validating and assuring the integrity and authenticity of the contents in the SD card. In short, it is responsible for reading, writing, deleting and assuring the digital security of the files in the SD card. In this solution we assume the SD card reader has limited resources, like small amount of RAM and low processing capabilities. As such, the transfer of the file will take significantly longer than transferring a file via USB.

The SD card is the portable storage device where the files get stored. It has built-in functions from its manufacturer. For this project the important functions are related to the allocation and writing of blocks. The way the SD card manages blocks is decided by the algorithms and heuristics the manufacturer has designed. It is beneficial to not interfere with these processes since the manufacturer of the card has expert knowledge of the physical composition of the card. Interfering with these processes can cause the blocks the become unevenly worn out or the write amplification factor might not be as efficiently controlled and this may lead to a large waste of space at the physical level in the SD card.

Figure 2 depicts the functions that can be executed by each member. These represent the basic functions necessary to guarantee that the system is usable. With these functions the user can utilise the SD card as a portable storage device, which is its intended purpose.

4.2. Overview

Figure 3 describes how the system would process the upload of a file to the SD card. First the user writes in the terminal, that is executing the program to communicate with the Arduino, the desired command. His device, in this case a laptop, relays that command to the Arduino. The process of transferring data begins, the Arduino receives chunks of data from the laptop and encrypts those chunks before writing them into the SD card. This process loops itself until the entire file is transferred. Then the Arduino sends an ACK to acknowledge that the file has been received without problems. The user can then select another command, from the ones showed to him in the interface of the terminal.

In Figure 4 it is possible to see the tree structure

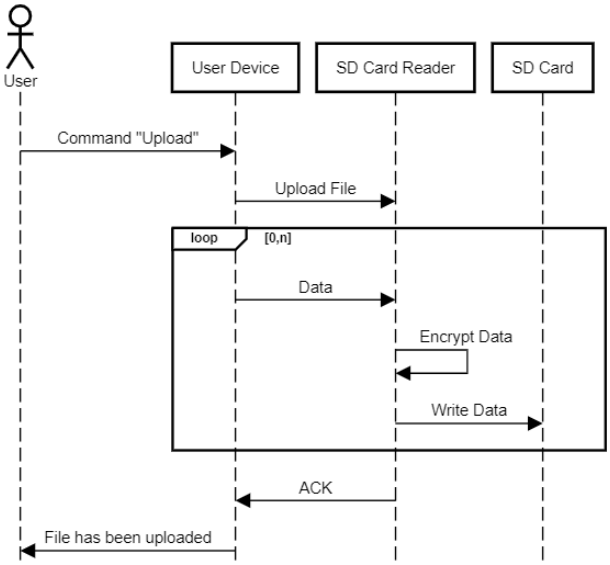

Figure 3: Overview Diagram

used to guarantee integrity. Inside the folder SYSTEM, a replica of the file system is created. But only file MACs and folder MACs are stored. The R_MAC is the root's MAC which is copied to the internal memory of the Arduino to guarantee data freshness. This replica of the file system allows the reading, insertion and deletion of file MACs to be done in O(1). It is only necessary to add the prefix "SYSTEM/" to the path defined by the user and the file MAC is obtained. The calculation of the folder MAC is done in O(n).

4.3. Hardware

For this project we will use an Arduino Uno. There are other more powerful variants like Arduino Mega or Due, but Uno has enough processing power and RAM to accomplish the requirements of this solution. It is also a cheap solution, which is important for this project since it aims to not make hardware a necessity or a deterrent. If the software developed can run on an Arduino Uno then it will be able to be scaled to any other type of hardware, like a smartphone, other more powerful micro-controllers, or even dedicated servers. By choosing such a restricted hardware environment we are guaranteeing that this solution is extremely scalable. This choice also does not restrict the user devices. The user has multiple choices of hardware to connect to the Arduino. It can be a desktop computer, a tablet, smartphone or any other thing capable of accepting USB connections. Another reason is also that Arduino features a large online community with help forums where many bugs and problems are discussed and resolved. This made the development of software easier, since many problems had already been addressed by others which reduced the time needed to resolve bugs. The on-

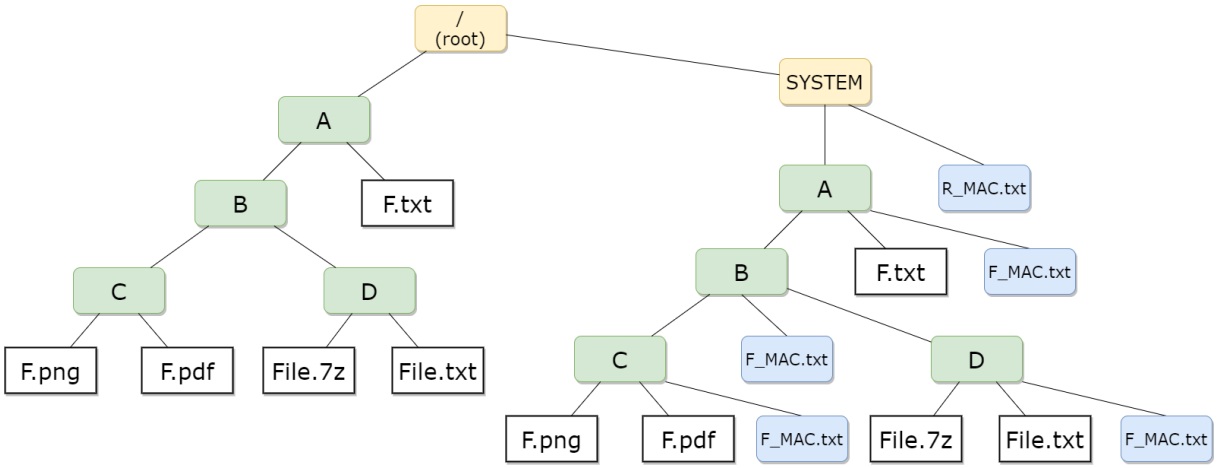

Figure 4: Integrity tree structure

line community is also quite active with constant bug fixing and software innovation. New libraries keep being created and added to the official forums. This helps to keep the solution maintainable and updatable in case it is necessary. The Arduino is connected to the user device via a USB cable, which also acts as a power cable for the Arduino.

4.4. Development

In this section the development methodologies used are presented. This work follows a three iteration process, where each iteration adds significant changes and functionality to the solution.

The tasks of the first iteration consisted in, studying and choosing the most appropriate file system library to use, creating file transfer functions specifically upload, download and file listing. Multiple file systems were analysed during this task. The first library used was the SdFat library. This library is compatible with exFAT but it is still in beta. While it was possible to create and write files with it, it had some bugs. It also had a heavy implementation, which when combined with the security libraries, would use too much RAM and the Arduino would simply not work. The next libraries used were FatFS and PetitFS. FatFS was quickly dismissed since it provided no advantage over the default SD library. The PetitFS library has a very small implementation size since it only included basic functions, however it only supported FAT32 or FAT16, it also only capable of writing files of up to 512 bytes. Since there was no library which supported exFAT and was lightweight, the default SD FAT3 library was used. It is the most tested and stable library so it is a good candidate.

In the first iteration of the solution it was necessary to create everything from scratch. Even though Arduino has a large online community with lots of guides, there was no work done on transferring entire files to the SD card via Arduino. The only similar work done consisted in sending keystrokes to the Arduino via its own built-in terminal. It was necessary to develop two scripts. One on the Arduino side which will become the main program. The second script is client side and is used to receive and transmit user commands and file data. During this iteration the main priority is to be able to transfer files between the SD card and the Arduino, encryption is the second priority.

The second iteration focuses on providing the necessary digital security requirements. This is the iteration where the encryption, file integrity, authenticity, data freshness, secure deletion will also be added. The tasks for this iteration involve creating the necessary functions to guarantee the security requirements. The implementation of the security algorithms was achieved with the use of a library. The library used was Arduino Cryptography Library. This library provides an extensive number of not only encryption algorithms but also integrity functions to chose from. As such it is a vastly superior choice when compared to the other options available which frequently only offer a single algorithm. The library is public, this increases its effectiveness since the community can help to verify and correct potential bugs in the implementation. For this reason it is preferable to use a public implementation of AES-XTS rather than creating it from scratch.

After selecting the library the encryption was added. The encryption and decryption calls are called in between the reception and the writing of a file chunk. When a file chunk is received by the Arduino, the block is encrypted and only then written to the SD card. With the encryption working, the next step was to add integrity. The integrity is done by creating a hash and updating it with the encrypted file chunk. After the entirety of the file is received, a file MAC is computed. The MAC consists of a concatenation of the hash of the encrypted file

and its file key, both encrypted with the user's master key. However this is when difficulties related to RAM space started to occur. The Arduino Uno's RAM was being used past its limit and the program was not executing properly. The creator of the library suggested to delete some functions that weren't being used and needlessly occupied RAM space. After performing the necessary changes to the library, there was enough RAM space to create and validate file MACs alongside the encryption. However it was still not possible to create and validate folder MACs. This is because to do so, it is necessary to have two open files, the directory and the file MAC, and also to calculate the hash. This problem persisted through weeks which forced the Third iteration to be started before the conclusion of this task. Eventually it was possible to complete this task by disabling compiler optimisation for specific functions. This allowed for less RAM to be used in trade for execution speed. The secure deletion implementation consists of rewriting the file MAC with its decrypted contents generated with a random key, the file is simply removed. To guarantee data freshness, the root MAC is copied into the Arduino's internal memory in every log off, and is verified in each log in.

The third and final iteration consisted of bug fixing and the creation of a simple GUI. This GUI allows the user to see the contents of its SD card in a tree like structure, which is updated after every file transaction. The GUI, which can be observed in Figures 5 and 6, also allows the user to press buttons to send commands. The main advantages of using the GUI are the fact that the user no longer needs to write the commands in a terminal, which reduces input errors. Makes users more comfortable when using the system since the information stored in the card is update in real time. The users no longer have to rely on prints to verify the contents of the SD card.

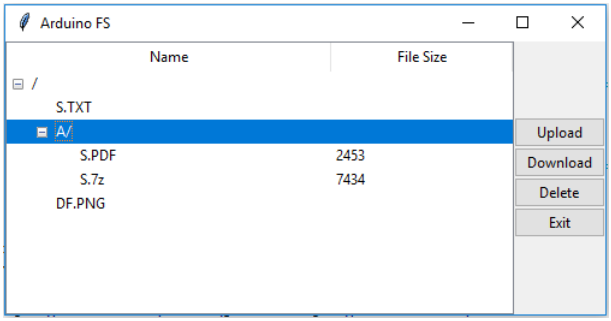

Figure 5: Graphical User Interface

5. Results and discussion

In this section we will discuss the results from the tests conducted. The file transfer rate is the metric to be studied since it shows the efficiency of the

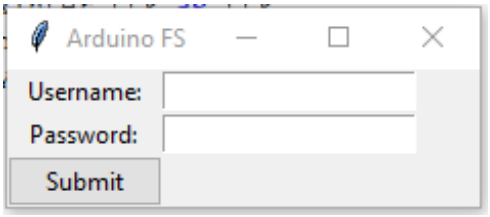

Figure 6: Login Menu

whole program. The faster the transfer rate, the more users who are likely to use this file system. In terms of transfer rates, the tests were conducted by measuring the time it took to either send a file to the SD card or to send a file to the laptop. With the file size and the time of transfer, it is possible to calculate the transfer rate in bytes per second. The file sizes increased exponentially so as to achieve a sizeable file and get a realistic transfer time that is not influenced by overheads or hardware execution fluctuations. Each file size was tested 100 times, which means each value in the graphs is the average value of those 100 testes. Three different scenarios were considered. The transfer time with the full system in place, which means encryption and integrity checks.

In the laptop to SD card transfer rates, the security operations overhead is noticeable. By encrypting the information as it is received the transfer rate is reduced by approximately 30%. With encryption and integrity checks and computations, we see that the transfer rate suffers a reduction of approximately 70%. This penalty is due not only to the fact that the data is being encrypted and the hash is being computed in real time, but also because it was necessary for some functions to turn off compiler optimisations to save RAM space in exchange for execution time. The slow file transfer times with smaller files can be attributed to file operations. The transfer time was measured until the interface received the final confirmation from the Arduino that the file has been closed and properly written to the SD card.

Overall, the maximum file upload rate is approximately 10091 bytes per second, this means that the file transfer achieves approximately 70% of the baud rate maximum transfer capacity.

The tests conducted in Figure 8 consisted in sending a file to a folder with one hundred files in the SD card. The test was conducted one hundred times, like the previous ones, with a file of 2000 bytes. With this test it is possible for us to calculate the time it takes for the Arduino to calculate the folder MAC of a folder with n files. The formula to calculate a folder MAC and update it is

$$time = 0.0419 * n$$

With these times it is possible to calculate the time

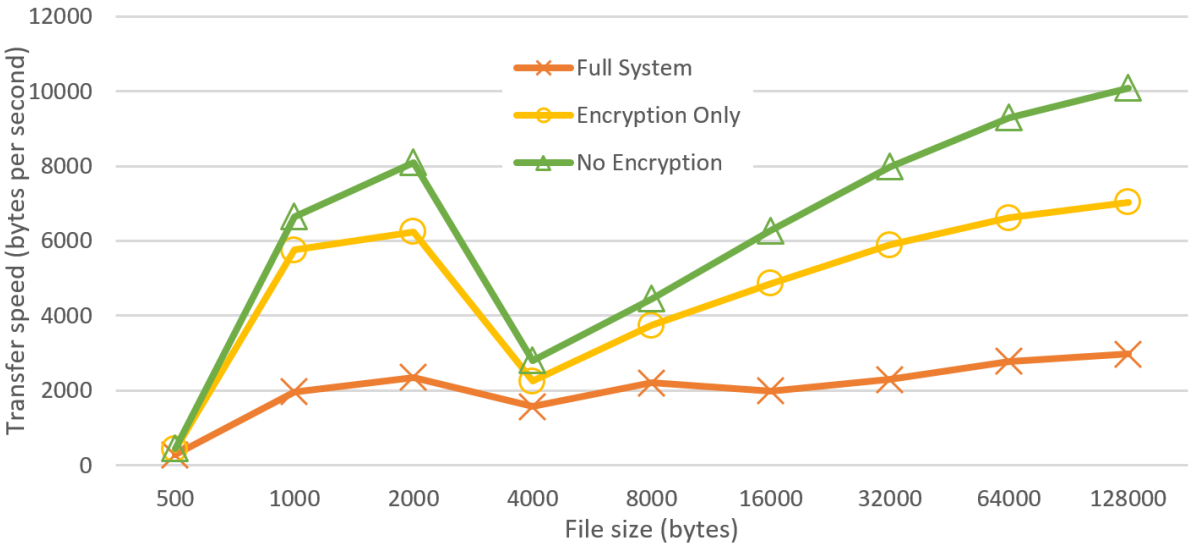

Figure 7: Transfer Rates (Laptop to SD card)

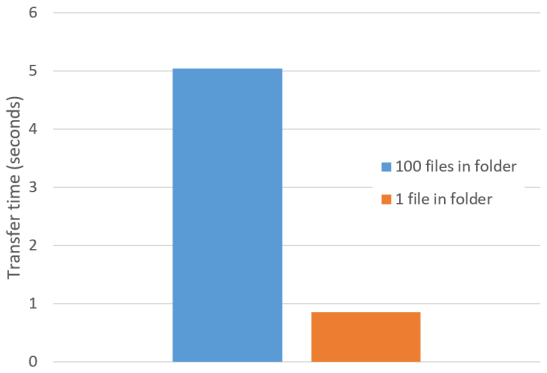

Figure 8: Folder MAC verification and update comparison

it takes to transfer a file of any size, to a folder with n files. The formula for transferring a file from the laptop to the SD card, using the full system, is

$$time = 0.0419n + 925.79ln(x) + 723.99$$

6. Conclusions

The objective of this work was to create a file system that provided digital security to flash based portable devices, using a micro-controller to handle the data management and the security operations. The micro-controller chosen was an Arduino. It is a fairly limited hardware. Despite its constraints, it was possible to implement a solution that used a file system that took into consideration the necessities of flash memory while also achieving a robust secure system. The system is able to encrypt the data with a secure encryption algorithm and cipher while also guaranteeing the integrity of each file, including directories. By doing so, the system also guarantees the freshness of the data, being impervious to attacks that use previous versions of files.

In relation to the system performance, the full system takes a heavy hit in performance, due to having to use last resort measures, such as disabling compiler optimisations.

References

[1] Aegis secure key 3z.

[2] Aes-xts block cipher mode is used in kingston's best secure usb flash drives.

[3] Ironkey d300 secure usb 3.0 flash drive - 4gb–128gb - kingston technology.

[4] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for key management part 1: General (revision 3). *NIST special publication*, 800(57):1–147, 2012.

[5] E. Barker, W. Barker, W. Burr, W. Polk, M. Smid, et al. *Recommendation for key management: Part 1: General*. National Institute of Standards and Technology, Technology Administration, 2006.

[6] W. A. Bhat and S. Quadri. Review of fat data structure of fat32 file system. *Oriental Journal of Computer Science & Technology*, 3(1), 2010.

[7] F. Cai, G. Bai, H. Liu, and X. Hu. Optical fault injection attacks for flash memory of smartcards. In *2016 6th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pages 46–50, 2016.

[8] G. Cattaneo, L. Catuogno, A. Del Sorbo, and P. Persiano. The design and implementation of a transparent cryptographic file system for unix. In *USENIX Annual Technical Conference, FREENIX Track*, pages 10–3, 2001.

[9] J. Daemen and V. Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[10] M. Dworkin. Recommendation for block cipher modes of operation. methods and techniques. Technical report, National Inst of Standards and Technology Gaithersburg MD Computer security Div, 2001.

[11] R. Geambasu, J. P. John, S. D. Gribble, T. Kohno, and H. M. Levy. Keypad: An auditing file system for theft-prone devices. In *Proceedings of the sixth conference on Computer systems*, pages 1–16. ACM, 2011.

[12] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka. Write amplification analysis in flash-based solid state drives. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, page 10. ACM, 2009.

[13] N. Joukov and E. Zadok. Adding secure deletion to your favorite file system. In *Third IEEE International Security in Storage Workshop (SISW'05)*, pages 8–pp. IEEE, 2005.

[14] T. Kato, T. Tsunehiro, M. Tsunoda, and J. Miyake. A secure flash card solution for remote access for mobile workforce. *IEEE Transactions on Consumer Electronics*, 49(3):561–566, 2003.

[15] J. Kelsey, B. Schneier, and D. Wagner. Key-schedule cryptanalysis of idea, g-des, gost, safer, and triple-des. In *Annual International Cryptology Conference*, pages 237–251. Springer, 1996.

[16] C. Lee, D. Sim, J. Hwang, and S. Cho. F2fs: A new file system for flash storage. In *13th {USENIX} Conference on File and Storage Technologies ({FAST} 15)*, pages 273–286, 2015.

[17] L. Martin. Xts: A mode of aes for encrypting hard disks. *IEEE Security & Privacy*, 8(3):68–69, 2010.

[18] Mikben. File system functionality comparison - win32 apps.

[19] F. Sano, M. Koike, S.-i. Kawamura, and M. Shiba. Performance evaluation of aes finalists on the high-end smart card. In *AES Candidate Conference*, pages 82–93, 2000.

[20] S. Skorobogatov. Flash memory 'bumping' attacks. volume 6225, pages 158–172, 08 2010.

[21] E. Stefanov, M. van Dijk, A. Juels, and A. Oprea. Iris: A scalable cloud file system with efficient integrity checks. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 229–238. ACM, 2012.

[22] J. Viega and H. Thompson. The state of embedded-device security (spoiler alert: It's bad). *IEEE Security Privacy*, 10(5):68–70, 2012.