American International Journal of Computer Science and Information Technology
Volume.9, Number 1; January-March, 2024;
ISSN: 2837-388X | Impact Factor: 7.49
https://zapjournals.com/Journals/index.php/aijcsit/
Published By: Zendo Academic Publishing

SECURING CONSUMER ELECTRONICS: A DEEP DIVE INTO CRYPTOGRAPHIC FILE SYSTEM ALGORITHM PERFORMANCE

Gabriella Lorraine Makovsky[1]

Article Info

Keywords: Consumer Electronics, Information Security, Internet of Things (IoT), Lightweight Cryptography, Data Encryption

DOI
10.5281/zenodo.10533755

Abstract

The rapid proliferation of consumer electronic devices with limited capabilities over the last decade has ushered in a remarkable era of growth, necessitating the establishment of a more robust computing, storage, and communication framework to ensure heightened security. This surge in consumer electronics has permeated diverse sectors such as communication, finance, entertainment, and other services, imposing a pressing need to adhere to stringent information security standards. The Gramm-Leach-Bliley Act, including the Financial Services Modernization Act and Federal Financial Institutions Examination Council, underscores the imperative of encrypting electronic customer information during transmission or storage on networks and systems vulnerable to unauthorized access.

As the consumer electronics landscape continues to expand, the integration of devices into the Internet of Things (IoT) poses a formidable challenge in devising security solutions capable of safeguarding the exchanged data. In the contemporary realm of digital communication networks, ensuring the privacy and security of transmitted data has emerged as an indispensable prerequisite for effective communication. Over time, there has been a concerted effort to explore data security schemes that furnish secure, real-time data storage across various applications, taking into account an array of potential attacks.

In this context, the focus has shifted towards the exploration and implementation of lightweight cryptography algorithms, particularly in the context of resource-constrained devices within the IoT. These lightweight cryptography algorithms serve as a crucial line of defense, mitigating security risks associated with devices operating under resource constraints. This paper delves into the evolving landscape of data security, emphasizing the significance of encryption protocols and the adoption of lightweight cryptography in addressing the unique challenges posed by the IoT ecosystem.

[1] Dept. of Electrical Engineering Technology

1. Introduction

Over the last decade, consumer electronic devices, which are limited in capability, have experienced a period of exceptional growth and require creating a more secure computing, storage, and communications environment. While the consumer electronics market is growing and influencing wide fields of communication, financial institutions, entertainment, and other services must satisfy their requirements and suggestions for information security. The Gramm-Leach-Bliley Act, also known as the Financial Services Modernization Act and Federal Financial Institutions Examination Council [1] suggests encryption of electronic customer information while it is in transit or storage on network or systems to which unauthorized individuals may have access. As the consumer industry has grown, devices connected to the IoT require a significant challenge in preparing security solutions to protect the data exchanged [2]. In today's world of digital communication networks, the privacy and security of the transmitted data has become a necessity for communication [3]. Data security schemes have been increasingly explored over time to provide secure, real-time data storage for a variety of applications.[4, 5] and with respect to various attacks [6]. Recently many lightweight cryptography algorithms have been used in securing the resource constraint devices in IoT

File system encryption is typically used to protect data in storage providing security requirements for electronic commerce transactions to make e-commerce transactions more secure [8]. Developers always desire to add new features to file systems that offer a clean and efficient data access mechanism transparent to user applications [9]. Using the Linux Crypto API [10] tools, integrated into the kernel, you can quickly create new valuable features, such as encryption, for efficient file systems without changing kernels or existing file systems. The Crypto API is an approach to unify the interface between kernel modules using crypto routines and kernel modules providing crypto routines.

This paper investigates the encryption/decryption performance of Crypto API cipher algorithms and demonstrates their applications in file system encryption using Cryptolop [11] and Dm-crypt tools [12, 13]. Customers who are still on the fence about adopting open-source software will find detailed instructions and information that could help them to move ahead with open-source strategies.

2. Consumer Electronics File Systems

Linux offers several different file systems to work with, such as cramfs and jffs2, which are highly used in consumer electronic devices. Flash memory is an increasingly common storage medium for CE devices as well. The cramfs is a read-only compressed file system, designed for simplicity and space efficiency [14, 15]. The cramfs uses a translation layer on flash devices to emulate a block device [16]. The Journaling Flash File System, version 2 (jffs2) [17], is a read/write compressed flash file system, commonly used in the embedded systems of consumer electronic devices. The jffs2 was specifically designed for embedded applications requiring persistent storage in flash memory. It places the file system directly on the flash chips to achieve this task. It can be mounted read/write and changes are preserved after a reboot [18]

The jffs2 file system was designed for relatively small flash chips but has serious problems when it is used on large flash devices. The ext2 file system is the most portable of the native Linux file systems that we used in tests for comparison reasons.

3. Tools to Create an Encrypted File System

The IoT platform consists of tiny low-cost consumer electronics devices continuously exchanging data that are usually limited in terms of hardware, memory capacity, and processing power [19].

Many devices use file systems to store critical information that requires absolute reliability and protection. Among several Linux tools that allow you to create an encrypted file system, we consider Cryptoloop and Dm-crypt specifically. They make it possible to create encrypted file systems within a partition or another file in the file

system. These encrypted files can be moved to a CD, DVD, or USB memory stick, etc. Dm-crypt is based on the device mapper and is considered to be a much cleaner code write, providing much more configuration flexibility than Cryptoloop. Cryptoloop and dm-crypt are based on the Crypto API and offer pretty much the same functionality. They can be used to add encryption to any of the standard Linux file systems without changing the file system code itself.

The user can specify one of the Crypto API symmetric ciphers with any allowed size key to create and mount an encrypted file system. Without the key, you can't access data in the file system. As a practical matter, we limited our search to well-known cipher algorithms integrated into the Linux kernel, such as aes, blowfish, cast6, serpent, and twofish [20]. This allows us to share encrypted file systems with others with a minimum amount of hassle.

4. Timing Commands

To get the timing of a code we used the Linux time shell command and a time-stamp counter. We developed a more accurate way to use a time-stamp counter, which keeps a count of every cycle that occurs on the processor [21]. The timestamp counter is a 64-bit register that is incremented every clock cycle. It can be read from both kernel space and user space, and the time-stamp counter is set to zero on reset. We used the read time-stamp counter rdtsc(low_var, high_var) macro, which reads the low half of the register into a 32-bit variable low_var, and the high half of the register into a 32-bit variable high_var. The rdtsc measures cycles.

We then used the rdtsc macro as timers, calling it before and after the section of code we want to time. The difference, converted from ticks to seconds, became the elapsed time for one run of the code. The elapsed times are accumulated by timers as a cryptographic component of the completed instructions. These instructions include: create an encrypted file system, mount a file system, and read, write, copy, and remove a file, etc. We then created a special kernel module and tested this timing method by encrypting and decrypting of a 1024-byte array using CryptoAPI cipher algorithms, as shown in Fig. 1.

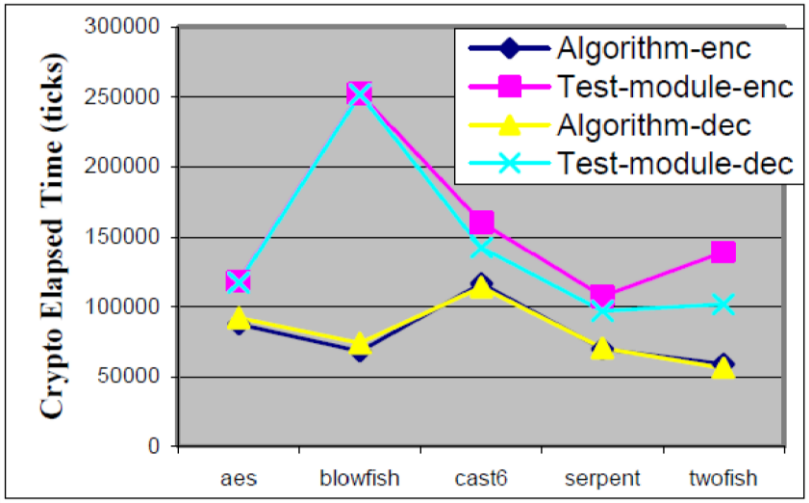

Fig. 1. Cryptography of 1024-byte array using testing timing method.

We patched both cryptoloop.c and dm-crypt.c with created timers.

We controlled the high half of the register to recover from overflows, but we did not need to access the whole register. Here is one of the code sections in the modified Cryptoloop module, to measure the encdecfunc(tfm, &sg_out, &sg_in, sz) procedure elapsed time: unsigned long long ticks_elapse_total; //global timer unsigned long begl, begh, endl, endh;//local #include <asm/msr.h> // Machine-specific registers #include "cryptolapse.h" //External variable definition

rdtsc(begl, begh);

```
encdecfunc(tfm, &sg_out, &sg_in, sz);
rdtsc(endl, endh);
. . .
ticks_elapse_total += endl - begl;
```

The ticks_elapse_total timer accumulates the encdecfunc procedure elapsed time during the encryption/decryption performance.

The low half of the register was sufficient in all our cases. Our 430-MHz system will overflow a 32-bit counter once about every 8.5 seconds, and a 2400-MHz system about every 1.7 seconds. The elapsed time we were benchmarking, reliably takes less time. To avoid a counter overflow influence, we controlled a series of procedure elapsed times and ignored anyone that exceeded twice the previous elapsed time.

5. System Setup

To provide encryption, we used the Linux loopback device driver to present a file as a block device, optionally transforming the data before it is written and after it is read from the native file. Cryptoloop and Dm-crypt use a cryptographic framework, CryptoAPI, which exports a uniform interface for all ciphers we tested.

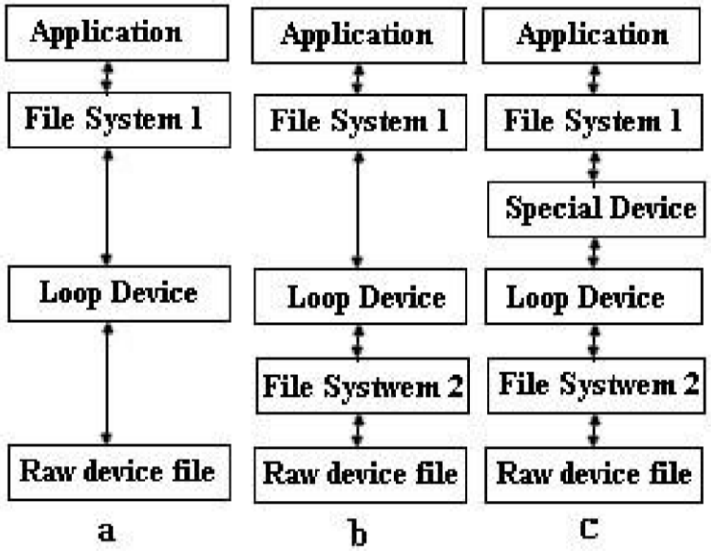

Fig. 2. Benchmarks.

We investigated three benchmarks shown in Fig. 2 for the loopback driver: (a) a raw device (e.g., /dev/hda9 and /dev/sda1), (b) a pre-allocated file, (c) another pre-allocated file and a special device that is required by Dm-crypt and jffs2 file system, such as MTD block and device-mapper [22]. The major difference between a raw device and a pre-allocated file is that a file system, created in a file, could be copied to flash memory.

Both Cryptoloop and Dm-crypt, jffs2 encrypted file systems, follow benchmark (c) shown above. An encryption structure for dm-crypt commands is shown in Fig. 3.

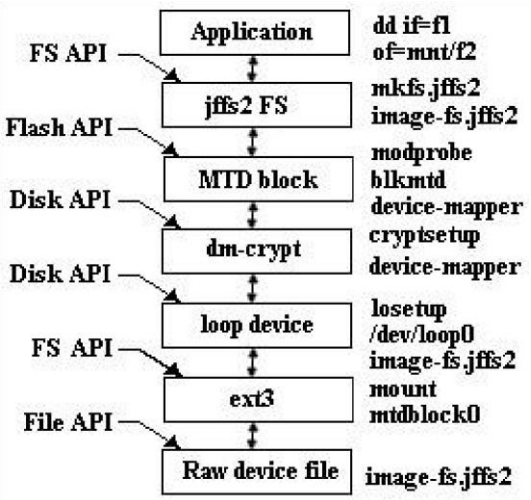

Fig. 3. Dm-crypt jffs2 FS encryption structure.

Here are complete instructions to create a Dm-crypt encrypted jffs2 file system on the pre-allocated file imagefs.jffs2:

losetup /dev/loop0 image-fs.jffs2

cryptsetup -c aes -y create image-fs.jffs2 /dev/loop0

mkfs.jffs2 -e128 -p33554432 -d empty -o /dev/mapper/image- fs.jffs2 modprobe blkmtd erasesz=128 device=/dev/mapper/image-fs.jffs2 mount -t jffs2 /udev/mtdblock0 mnt

The instructions to create Cryptoloop encrypted jffs2 file system on the pre-allocated file image-fs.jffs2: losetup -e aes-256 /dev/loop0 image-fs.jffs2

mkfs.jffs2 -e128 -p33554432 -d empty -o /dev/loop0 modprobe blkmtd erasesz=128 device=/dev/loop0 mount -t jffs2 /udev/mtdblock0 mnt

Both cramfs and ext2 file systems, encrypted by Cryptoloop, follow benchmark (b) in Fig. 2. We created ext2 file systems on the pre-allocated file image-fs.ext2 using these instructions:

losetup -e aes-256 /dev/loop0 image-fs.ext2 mkfs.ext2 -b 1024 /dev/loop0 mount -t ext2 /dev/loop0 mnt

We then created cramfs file systems on the pre-allocated file image-fs.cramfs using these instructions: losetup -e aes-256 /dev/loop0 image-fs.cramfs mkfs.cramfs dir-root-files /dev/loop0 mount -t cramfs /dev/loop0 mnt

Both cramfs and ext2 file systems, encrypted by Dm-crypt, follow benchmark (c) in Fig. 2. We then created Dm-crypt ext2 file systems on the pre-allocated file image-fs.ext2 using these instructions: losetup /dev/loop0 image-fs.ext2 cryptsetup -c aes -y create image-fs.ext2 /dev/loop0 mkfs.ext2 /dev/mapper/image-fs.ext2 mount /dev/mapper/image-fs.ext2 mnt

We then created Dm-crypt cramfs file systems on the preallocated file image-fs.cramfs using these instructions: losetup /dev/loop0 image-fs.cramfs cryptsetup -c aes -y create image-fs.cramfs /dev/loop0 mkfs.cramfs dir-root-files /dev/mapper/image-fs.cramfs mount /dev/mapper/image-fs.cramfs mnt

6. Test Beds

Consumer electronics devices cover a wide-ranging field of electronics with a broad spectrum of processor speed, memory type, memory size, and data. We ran our tests using single user and multiuser modes on two workstations, 430 MHzand 2.4 GHz. All tests took place on a 13.6GB disk and 128MB Flash Memory Key. To minimize the impact of cache and other I/O operations, all tests were performed on a cold cache, achieved by unmounting, and remounting the file systems, removing modules, and detaching the loop device between iterations. The tests were located on a dedicated partition in the outer sectors of the disk and in a flash memory key. We documented the

command elapsed time (real time), the time involved in the test kernel processes (sys time), and the elapsed time spent encrypting/decrypting (crypto elapsed time). We ran all the tests several times.

7. Performance Comparison

Security benefits of an encrypted file system necessitate encryption overhead time that makes operation performance slower from 47% for the twofish cipher algorithm to 113% for the cast6 cipher algorithm, as shown in Fig. 4. This happened when writing a 50 MB file to an encrypted ext2 file system, which was created on a 128 MB key flash by Cryptoloop. We also used different CryptoAPI cipher algorithms.

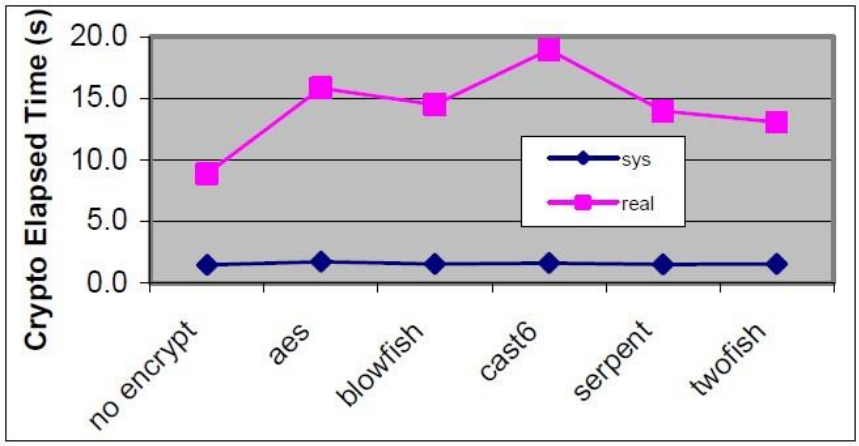

Fig. 4. Write 50 MB file into 128MB key flash encrypted ext2 file system.

The most important commands related to the file system creation and usage are mkfs, mount, read, and write. Table 1 shows the cryptographic part of the performed commands on an encrypted 32 MB jffs2 file system using Cryptoloop and Dm-crypt.

TABLE I ENCRYPTED JFFS2 FS COMMAND'S PERFORMANCE

Tested command	mkfs	mount	write	read	mount	write	read	mount
		0 files	1 file	1 file	2 files	1 file	1 file	2 files
Files size (MB)	32		1	1	2	10	10	11
Cryptoloop (s)	11.32	0.22	2.03	0.21	0.68	20.27	4.45	4.41
sys (s)	5.85	0.02	1.20	0.19	0.04	11.83	4.24	0.17
dm-crypt (s)	13.10	0.21	1.38	0.24	0.63	13.99	2.15	3.86
sys (s)	9.93	0.03	2.23	0.09	0.06	22.33	0.85	0.20

The encryption and decryption elapsed time level of commands is an important metric for analysis of encrypted file system performance. Based on the proposed timing method we compared and contrasted the file system performance of encryption by Cryptoloop and Dm-crypt cramfs, jffs2, and ext2 file systems, using Linux Crypto API cipher algorithms aes, blowfish, cast6, serpent, and twofish that are all licensed free for any use. For all cipher algorithms we used 256-bit key size.

Jffs2 is a compressed file system that directly reads and writes to flash and scans all nodes at mount time. We assumed that files and folders that are to be encrypted by the file system require about 32 MB. To study the effects of encryption, we analyze jffs2 with small files (about 3% of the 32 MB file system size) and big files (about 30% of the 32 MB file system size). Dm-crypt and Cryptoloop show similar trends on small file sizes.

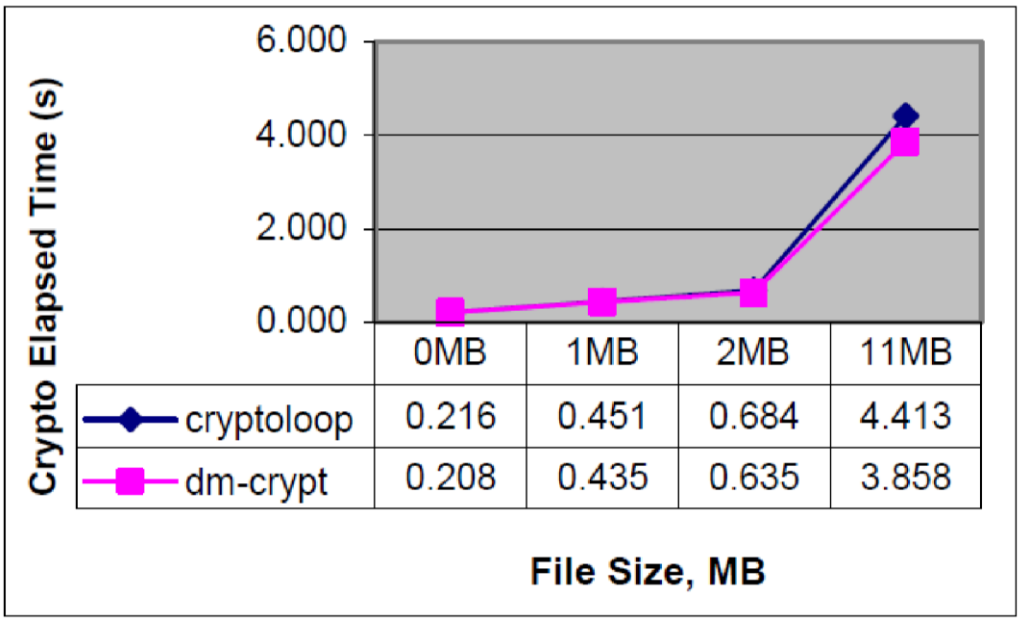

Fig. 5. Mounting (mount) encrypted jffs3 file system with different file sizes performance.

As the file size increases, Dm-crypt spends less crypto time than Cryptoloop (Table 1, Fig. 5). As you can see, each cipher algorithm requires a different crypto time (Fig. 6).

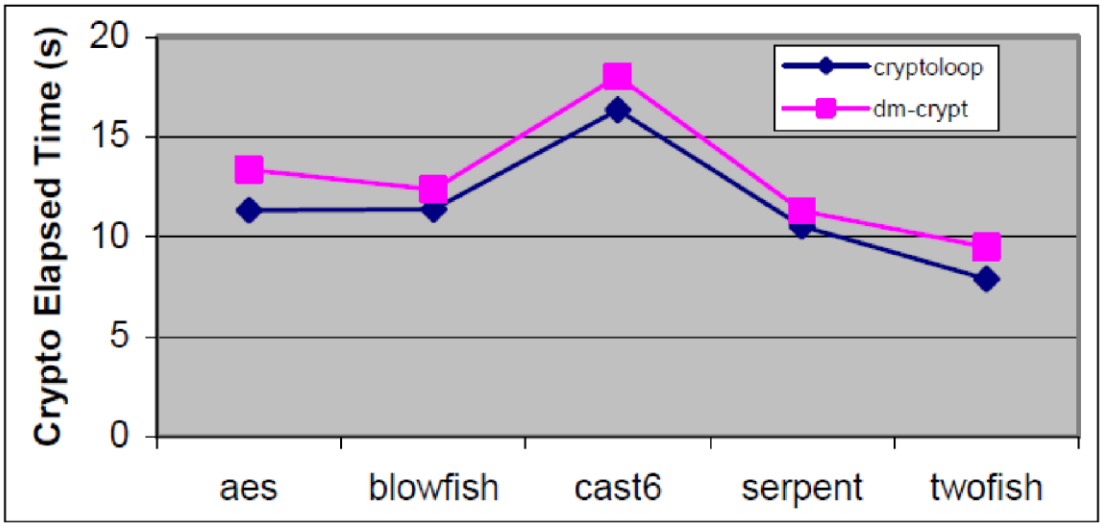

Fig. 6. Creation (mkfs) encrypted 32MB jffs2 file system using Cryptoloop and Dm-crypt and different ciphers.

Table 1 and Fig. 5 show that jffs2 takes more time to mount a file system with bigger files since it scans all nodes at mount time. As for cramfs, the contents are decrypted as needed, and the mount time is almost independent of file size in the file system. These results can also be seen in Table 2 which shows crypto elapsed time for commands on 32MB encrypted cramfs FS using Cryptoloop and Dm-crypt.

This table shows that Cryptoloop performs faster creation and mounting of cramfs file systems than Dm-crypt and lags a little on reading both small and big files.

TABLE IIENCRYPTED CRAMFS FS COMMAND'S PERFORMANCE

Tested command	mkfs	mount	mkfs	mount	mkfs	mount	read	read
	1 file	1 file	2 files	2 files	2 files	2 files	1 file	1 file
Files size (MB)	32	1	32	2	32	11	1	10
Cryptoloop (s)	0.194	0.003	0.388	0.003	2.128	0.004	0.216	2.181
sys (s)	0.105	0.005	0.294	0.006	1.461	0.005	0.083	0.822
dm-crypt (s)	0.220	0.016	0.419	0.016	2.274	0.016	0.208	2.089
sys (s)	0.390	0.007	0.768	0.005	2.948	0.007	0.930	0.910

For benchmarking Dm-crypt and Cryptoloop, we used our second2.4 GHz processor test bed. We certainly see some different results due to the higher performance machine in Table 3 which shows crypto elapsed time for commands on 32MB encrypted cramfs and ext2 file systems, using Cryptoloop and Dm-crypt. We also tested crypto elapsed time for reading and writing of 1 MB file.

For most of the operations, Cryptoloop, still outperforms Dm-crypt, and shows similar performance regarding each algorithm. The biggest difference between cramfs and ext2 is the time it takes to make the file system. The cramfs is clearly much faster than ext2.

TABLE III ENCRYPTED FS COMMAND'S PERFORMANCE

Tested Commands	Ciphers				
	aes	blowfish	cast6	serpent	twofish
Ext2					
Dm-Crypt: make (s)	0.0244	0.0311	0.0882	0.0622	0.0549
Cryptoloop: make (s)	0.0258	0.0301	0.0935	0.0618	0.0547
Dm-Crypt: write (s)	0.0017	0.0021	0.0063	0.0041	0.0039
Cryptoloop: write (s)	0.0016	0.0020	0.0058	0.0040	0.0044
Dm-Crypt: read (s)	0.0153	0.0195	0.0550	0.0414	0.0298
Cryptolop: read (s)	0.0144	0.0176	0.0548	0.0406	0.0352
Cramfs					
Dm-Crypt: make (s)	0.0144	0.0181	0.0519	0.0360	0.0347
Cryptoloop: make (s)	0.0145	0.0184	0.0522	0.0358	0.0347
Dm-Crypt: read (s)	0.0161	0.0200	0.0553	0.0418	0.0310
Cryptoloop: read (s)	0.0150	0.0184	0.0547	0.0414	0.0306

8. Conclusion

The contributions of this work are that we generalized benchmarks for Linux cryptographic file systems and the performance of Crypto API cipher algorithms. We showed the prototype of the implemented benchmarks to create the desired cryptographic file systems of your choice (jffs2, cramfs, and ext2) and mount it to the location of your choice (dedicated partition, file on another file system, CD, USB memory stick). We developed a timer module

for timing investigation of cryptographic activities for the Linux Cryptoloop and Dm-Crypt methods and performed a comprehensive comparison of cipher algorithms (aes, blowfish, cast6, serpent, and twofish) and their usage in the ext2, cramfs, and jffs2 cryptographic file systems, measuring more specific aspects of encryption/decryption performance in CE devices. These results can be used by a system designer to estimate the crypto-timing consumption of encrypted cramfs, jffs2, and ext2 file systems.

References

Gramm-Leach-Bliley Act, (1999). [Online] Available: https://www.ftc.gov/business-guidance/privacy-security/data-security (July 23, 2023) Careful Connections: Keeping the Internet of Things Secure, (2020). [Online] Available:

https://www.ftc.gov/business-guidance/resources/careful-connections-keeping-internet-things-secure (July 23, 2023)

Sunny A. (2022). A Review on Various Methods of Cryptography for Cyber Security. Journal of Algebraic Statistics, vol. 13, no. 3, 2022, 5016-5024.

Singhal, V., Singh, D., & Gupta, S. K., "Crypto STEGO Techniques to Secure Data Storage Using DES, DCT, Blowfish and LSB Encryption Algorithms", Journal of Algebraic Statistics. 2022, vol. 13, no. 3, p1162-1171.

Pandey, A. & P. Bonde, P. Performance evaluation of various cryptography algorithms along with LSB substitution technique. International Journal of Engineering Research & Technology (IJERT), vol. 2, no. 6, 2013, pp. 866871.

Jammula, M., Vakamulla, V. M., &Kondoju, S. K. (2022). Performance evaluation of lightweight cryptographic algorithms for heterogeneous IoT environment. Journal of Interconnection Networks, 2022 Supplement, vol. 22, 1-21.

Mousavi, S. K., Ghaffari, A., Besharat, S., &Afshari, H. (2021). Security of internet of things based on cryptographic algorithms: a survey. Wireless Networks (10220038), vol. 27, no. 2, 1515-1555.

Shyaa, G. S. & Al-Zubaidie, M. (2023). Utilizing Trusted Lightweight Ciphers to Support Electronic-Commerce Transaction Cryptography. Applied Sciences (2076-3417), Jun2023, vol. 13, no. 12, 7085-7111.

Wright, C., Martino, M., & Zadok, E. NCryptfs: A Secure and Convenient Cryptographic File System. USENIX 2003 Annual Technical Conference, San Antonio, Texas, 197-210.

The GNU/Linux CryptoAPI site: www.kernel.org. Mueller, S. &Vasut, M. CryptoAPI. [Online] Available: https://www.kernel.org/doc/html/latest/crypto/index.html (July 23, 2023) Hoelzer, Ralf. Cryptoloop HOWTO. [Online] Available: http://www.ibiblio.org/pub/Linux/docs/HOWTO/Cryptoloop-HOWTO (July 23, 2023)

Broz, Milan, (2021), Dm-crypt: Linux kernel device-mapper crypto target. [Online] Available: https://gitlab.com/cryptsetup/cryptsetup/-/wikis/DMCrypt (July 23, 2023)

Project ID: 195655. Cryptsetup and LUKS - open-source disk encryption. [Online] Available: https://gitlab.com/cryptsetup/cryptsetup (July 23, 2023)

Notes on filesystem layout. [Online] Available: http://lxr.linux.no/source/fs/cramfs/README (July 23, 2023)

Ahn, S., Hyun, S., Kim, T. & Bahn, H. (2013). A compressed file system manager for flash memory basedconsumer electronics devices. IEEE Transactions on Consumer Electronics, vol. 59, no. 3, 544-549.

Quinlan, Daniel. cramfs tools. [Online] Available: https://sourceforge.net/projects/cramfs/ (July 23, 2023)

Woodhouse, David (2003). JFFS2: The Journalling Flash File System, version 2. [Online] Available: https://www.sourceware.org/jffs2/ (July 23, 2023)

Pan, Y., Hu, Z., Zhang, N., Hu, H., Xia, W., Jiang, Z., Shi, L., & Li, S. (2022). HNFFS: Revisiting the NOR Flash File System. 2022 IEEE 11th Non-Volatile Memory Systems and Applications Symposium (NVMSA), 14-19.

Gookyi, D. A. N., Ryoo, K. (2022). A Lightweight System-On-Chip Based Cryptographic Core for Low-Cost Devices. Sensors (14248220), Apr2022, vol. 22, no. 8, 1-28.

Stallings, W. (2020). Cryptography and Network Security: Principles and Practice. (8th ed.). Pearson (Chapters 3-7).

Corbet, J., Rubini, A., &Kroah-Hartman, G. (2005). Linux Device Drivers. (3rd ed.). O'Reilly (Chapters 7). [Online] Available: https://www.oreilly.com/openbook/linuxdrive3/book/ (July 23, 2023)

Lomako, G. & Frank Delmas, F. Performance of Two Linux Methods for Encrypting Data at the File System Level in CE Devices. In 4th IEEE Consumer Communications and Networking Conference, CCNC 2007, Las Vegas, NV, USA, January 11-13, 2007, IEEE, 2007, 1176-1177.