IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Optimizing an Encryption File System On Android through Blowfish Algorithm

Miss Kirti . P. Lokhande
ME(CSE) 3rd Sem
G.H. Raisoni College of Engineering,Amravati

Dr. Mahip M. Bartere
Asst Prof. (CSE)
G.H. Raisoni College of Engineering,Amravati

Abstract— Recently smart gadgets including smart phones and tablets are gaining popularity. These devices contain Personal Identifiable Information. An attacker can compromise a smart phone and gain full control of it by connecting another computing device to it using the USB physical link. Moreover, by simply capturing the smart phones physically, adversaries have access to confidential or even classified data if the owners are the government officials or military personnel. In this papert, here discuss a secure and encrypted file system on Android operating system and optimize the performance using certified encryption algorithm BlowFish provided in OPENSSL libraries. This paper presents EncFS which is a FUSE (File system in USErspace) based file-system offering encryption file system to protect the removable and persistent storage on heterogeneous smart gadget devices running the Android platform. In this papet, the data at rest including physical partition on the device and removable storage card is encrypted using user provided password. The encrypted file system is mounted only after successful password verification with user at system boot up.

Index Terms— Smart handheld devices, Full disk encryption, Encrypted filesystem, *Blowfish, Encryption, Decryption*.

1 INTRODUCTION

Technology trends in both hardware and software have driven the hardware industry towards smaller, faster and more capable mobile hand-held devices that can support a wider-range of functionality and open source operating systems. Mobile hand-held devices are popularly called smart gadgets (e.g. smart phones, tablets, e-book readers). The smart gadget life-cycle has evolved drastically in recent years. These new generations of the smart gadget devices such as the iPhone and Google Android devices are powerful enough to accomplish most of the tasks that previously required a personal computer. However, smart gadgets have to come a long way in terms of security. Organizations have come to realize that these commercially available smart gadgets will soon have to serve as an integral part of their operations. This requires a level of security that allows for security of data at-rest and on the move to support secure communications. Major challenges in providing file system encryption in mobile devices are:

1. A major obstacle is that there is a serious lack of National Institute of Standards (NIST) approved encryption algorithms on these commercially available smart gadgets.
2. On smart gadgets where resources, like the battery, are very limited, it is important to keep a low footprint on such solutions.
3. The expectation for each individual application to support encryption runs into the key management problem: other applications in the system can potentially gain access to the key and render the encryption useless.
4. This system has to be ubiquitous and integrate into the ecosystem of smart gadgets with minimal maintenance and installation cost.

Therefore, there is a need for a practical approach to build common security libraries that operate at the operating system level and provide strong encryption. In this paper, the focus is on analyzing the performance for persistent storage protection using encryption on smart gadget devices. The paper uses EncFS which is a FUSE (File system in USErspace) based encryption file system and uses certified cryptographic algorithms to store encrypted versions of every file in a source directory. The volume key is decrypted using a password supplied by the user. This is different from full-disk encryption software because the protected data is mounted in memory at a specified mount point in the file system. Since file I/O operations on the mount point eventually hit the encrypted copy of the file on the underlying file system, various performance optimizations can be possible by adjusting the file system parameters such as block size, buffer size. Finally, we discuss the limitations of file system encryption and demonstrate that it is feasible on smart gadget devices with a reasonable performance overhead. Additionally, this paper leverages advantages provided by underlying operating system:

1. Use of NIST validated cryptographic libraries which are not implemented in kernel-space.
2. The proposed implementation can be extended to different hardware with negligible effort.
3. This paper needs to focus on data encryption without having to deal with other aspects of file system design. Underlying file systems like ext3 and yaffs2 already have strong support for handling data-corruption and journaling.

2 LITERATURE SURVEY

In 2012 Zhaohui Wang, Rahul Murmuria, Angelos Stavrou [1] has presented Implementing and Optimizing an Encryption File system on Android ,a novel FUSE (Filesystem in USErspace) encryption file system to protect the removable and persistent storage on heterogeneous smart gadget devices running the Android platform. The proposed file system leverages NIST certified cryptographic algorithms to encrypt the data at-rest. We present an analysis of the security and performance trade-offs in a wide-range of usage and load scenarios. Using existing known micro benchmarks in devices using encryption without any optimization, we show that encrypted operations can incur negligible overhead for read operations and up to twenty (20) times overhead for write operations for I/O intensive programs. In addition, we quantified the database transaction performance and we observed a 50% operation time slowdown on average when using encryption. We further explore generic and device specific optimizations and gain 10% to 60% performance for different operations reducing the initial cost of encryption. Finally, we show that our approach is easy to install and configure across all Android platforms including mobile phones, tablets, and small notebooks without any user perceivable delay for most of the regular Android applications [1].

In 2012 Pratap Chnadra Mandal has presented Superiority of Blowfish Algorithm, containing Information Security has been very important issue in data communication. Any loss or threat to information can prove to be great loss to the organization. Encryption technique plays a main role in information security systems. This paper provides a fair comparison between four most common and used symmetric key algorithms: DES, 3DES, AES and Blowfish. A comparison has been made on the basis of these parameters: rounds block size, key size, and encryption / decryption time, CPU process time in the form of throughput and power consumption. These results show that blowfish is better than other algorithm [2].

In 2012 M. Anand Kumar and Dr.S.Karthikeyan has presented Investigating the Efficiency of Blowfish and Rejindael (AES) Algorithms, The growth rate of the internet exceeds than any other technology which is measured by users and bandwidth. Internet has been growing at a rapid rate since its conception, on a curve geometric and sometimes exponential. Today, the Internet is moving exponentially in three different directions such as size, processing power, and software sophistication making it the fastest growing technology humankind has ever created. With the rapid growth of internet, there is need to protect the sensitive data from unauthorized access. Cryptography plays a vital role in the field of network security. Currently many encryption algorithms are available to secure the data but these algorithms consume lot of computing resources such as battery and CPU time. This paper mainly focuses on two commonly used symmetric encryption algorithms such as Blowfish and Rejindael. These algorithms are compared and performance is evaluated. Experimental results are given to demonstrate the performance of these algorithms [4].

In 2011 Jawahar Thakur1 , Nagesh Kumar has presented DES, AES and Blowfish: Symmetric Key Cryptography Algorithms Simulation Based Performance Analysis, Security is the most challenging aspects in the internet and network applications. Internet and networks applications are growing very fast, so the importance and the value of the exchanged data over the internet or other media types are increasing. Hence the search for the best solution to offer the necessary protection against the data intruders' attacks along with providing these services in time is one of the most interesting subjects in the security related communities. Cryptography is the one of the main categories of computer security that converts information from its normal form into an unreadable form. The two main characteristics that identify and differentiate one encryption algorithm from another are its ability to secure the protected data against attacks and its speed and efficiency in doing so. This paper provides a fair comparison between three most common symmetric key cryptography algorithms: DES, AES, and Blowfish. Since main concern here is the performance of algorithms under different settings, the presented comparison takes into consideration the behavior and the performance of the algorithm when different data loads are used. The comparison is made on the basis of these parameters: speed, block size, and key size. Simulation program is implemented using Java programming

[5].

In 2010 Aditya Rajgarhia , Ashish Gehani Proposed Performance and Extension of User Space File Systems , Several efforts have been made over the years for developing file systems in user space. Many of these efforts have failed to make a significant impact as measured by their use in production systems. Recently, however, user space file systems have seen a strong resurgence. FUSE is a popular framework that allows file systems to be developed in user space while offering ease of use and flexibility and discuss the evolution of user space file systems with an emphasis on FUSE, and measure its performance using a variety of test cases. We also discuss the feasibility of developing file systems in high-level programming languages, by using as an example Java bindings for FUSE that we have developed. Our benchmarks show that FUSE offers adequate performance for several kinds of workloads [6].

3 INTRODUCTION TO CRYPTOGRAPHY

An encryption algorithm plays an important role in securing the data in storing or transferring it. The encryption algorithms are categorized into Symmetric (secret) and Asymmetric (public) keys encryption. In Symmetric key encryption or secret key encryption, only one key is used for both encryption and decryption of data. Example: Data encryption standard (DES), Triple DES, Advanced Encryption Standard (AES) and Blowfish Encryption Algorithm. In asymmetric key encryption or public key encryption uses two keys, one for encryption and other for decryption. Example: RSA

4 BLOWFISH ALGORITHM

Blowfish is a variable-length key block cipher. It does not meet all the requirements for a new cryptographic standard. It is only suitable for applications where the key does not change often, like a communications link or an automatic file encryptor. It is significantly faster than DES when implemented on 32-bit microprocessors with large data caches, such as the Pentium and the PowerPC. Blowfish is one of the fastest block ciphers in general use, except when changing keys. Each new key requires pre-processing equivalent to encrypting about 4 kilobytes of text, which is very slow compared to other block ciphers. This prevents its use in certain applications, but is not a problem in others, such as SplashID. In an application, it's actually a benefit especially the password-hashing method used in OpenBSD uses an algorithm derived from Blowfish that makes use of the slow key schedule. Blowfish is not subject to any patents and is therefore freely available for anyone to use. This has contributed to its popularity in cryptographic software.

4.1 Blowfish Encryption algorithm

Blowfish was designed in 1993 by Bruce Scheier as a fast, alternative to existing encryption algorithms. Blowfish is a symmetric block encryption algorithm designed in consideration with

1. Fast: it encrypts data on large 32-bit microprocessors at a rate of 26 clock cycles per byte.
2. Compact: it can run in less than 5K of memory.
3. Simple: it uses addition, XOR, lookup table with 32-bit operands.
4. Secure: the key length is variable, it can be in the range of 32~448 bits: default 128 bits key length.
5. It is suitable for applications where the key does not change often, like communication link or an automatic file encryptor.
6. Unpatented and royalty-free.

4.2 Description of the Algorithm

Blowfish is a variable-length key, 64-bit block cipher. The algorithm consists of two parts: a key expansion part and a data- encryption part. Key expansion converts a key of at most 448 bits into several sub key arrays totaling 4168 bytes. Data encryption occurs via a 16-round Feistel network. Each round consists of a key-dependent permutation, and a key- and data-dependent substitution. All operations are XORs and additions on 32-bit words. The only additional operations are four indexed array data lookups per round.

Sub keys: Blowfish uses a large number of sub keys. These keys must be pre-computed before any data encryption or decryption.

1. The P-array consists of 18 32-bit sub keys: P1, P2,..., P18.
2. There are four 32-bit S-boxes with 256 entries each:

S1,0, S1,1,..., S1,255;

S2,0, S2,1,..,, S2,255;

S3,0, S3,1,..., S3,255;

S4,0, S4,1,..,, S4,255.

Need of digits of pi(∏) : The use of digits of pi as a initial sub key table for two reasons:

1. It is random sequence not related to algorithm.

2. It could be either stored as part of the algorithm or derived when needed.

Generating the Sub keys:

The sub keys are calculated using the Blowfish algorithm. The exact method is as follows:

1. Initialize first the P-array and then the four S-boxes, in order, with a fixed string. This string consists of the hexadecimal digits of pi (less the initial 3). For example:

P1 = 0x243f6a88

P2 = 0x85a308d3

P3 = 0x13198a2e

P4 = 0x03707344

2. XOR P1 with the first 32 bits of the key, XOR P2 with the second 32-bits of the key, and so on for all bits of the key (possibly up to P14). Repeatedly cycle through the key bits until the entire P-array has been XORed with key bits. (For every short key, there is at least one equivalent longer key; for example, if A is a 64-bit key, then AA, AAA, etc., are equivalent keys.)
3. Encrypt the all-zero string with the Blowfish algorithm, using the subkeys described in steps (1) and (2).
4. Replace P1 and P2 with the output of step (3).
5. Encrypt the output of step (3) using the Blowfish algorithm with the modified sub keys.
6. Replace P3 and P4 with the output of step (5).
7. Continue the process, replacing all entries of the P-array, and then all four S-boxes in order, with the output of the continuously-changing Blowfish algorithm. In total, 521 iterations are required to generate all required sub keys. Applications can store the sub keys rather than execute this derivation process multiple times.

4.3 Blowfish Encryption

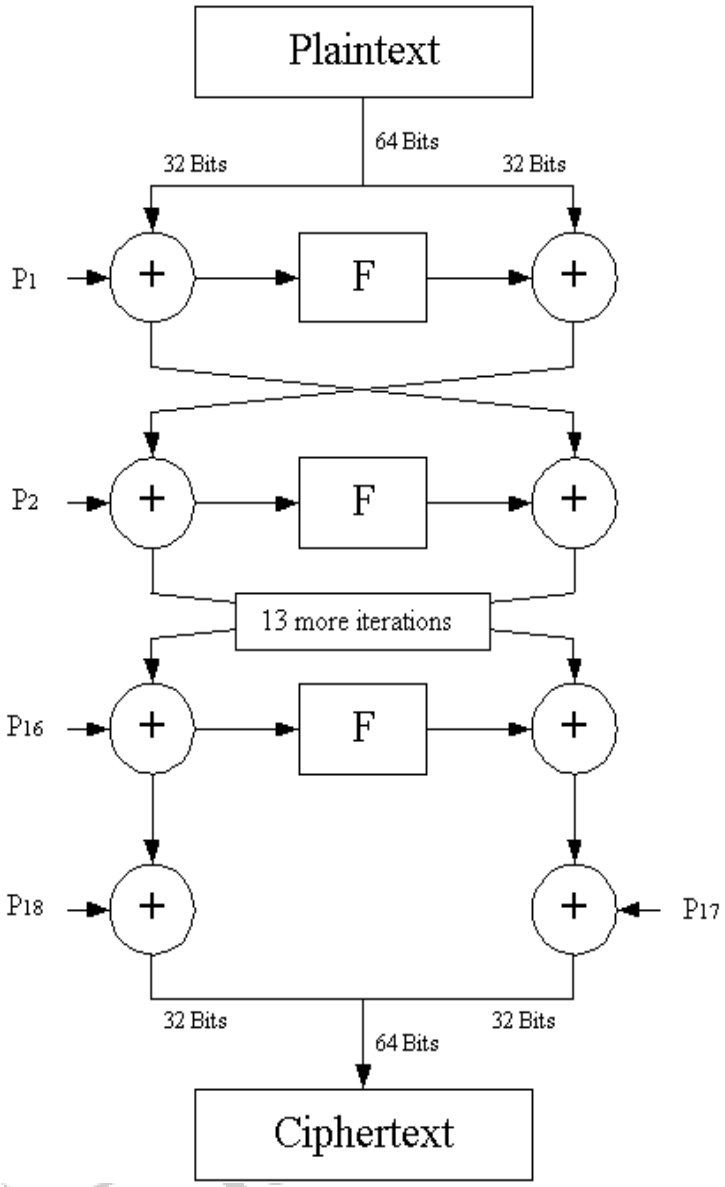


Fig 1 : Blowfish Encryption

Blowfish is a Feistel network consisting of 16 rounds. The input is a 64-bit data element, x.

Divide x into two 32-bit halves: xL,xR

For i = 1 to16:

xL = xL XOR Pi

xR = F(xL) XOR xR

Swap xL and xR

Next i

Swap xL and xR(Undo thelast swap.)

xR = xR XOR P17

xL = xL XOR P18

Recombine xL and xR

Function F

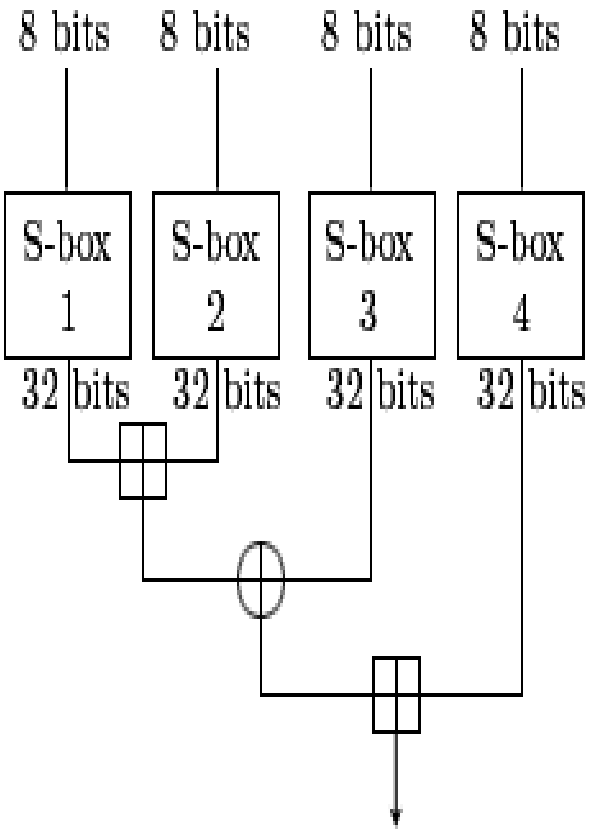


Fig 3: Function F

Divide xL into four eight-bit quarters: a, b,c, and d

F(xL) = ((S1,a + S2,b mod 232) XOR S3,c) + S4,d mod 232

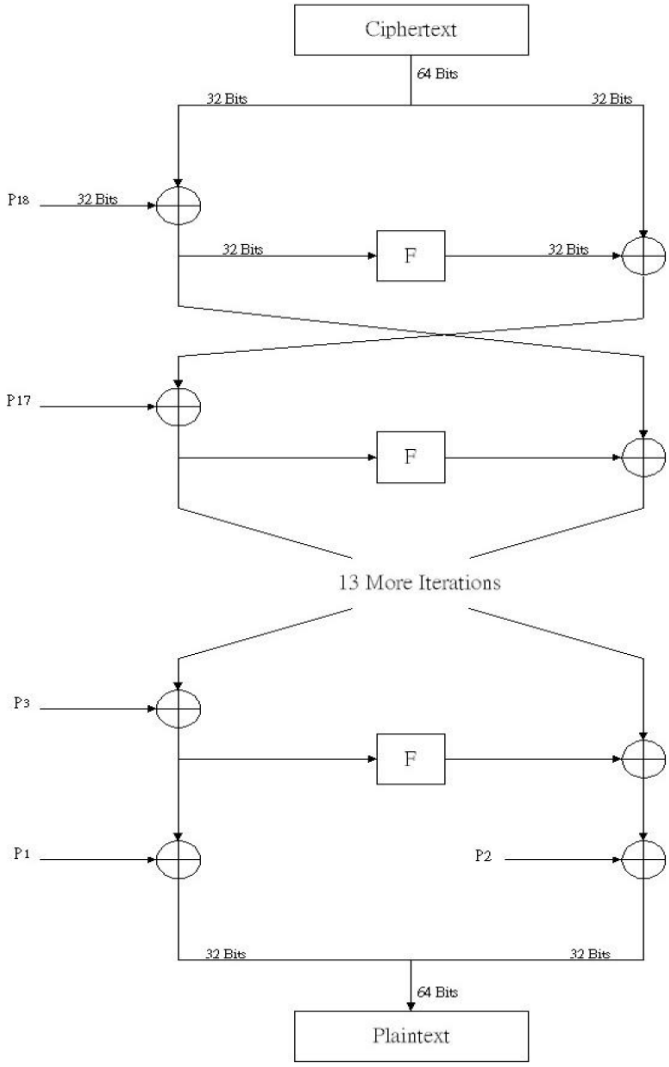


Fig 4: Blowfish decryption

4.4 Blowfish Decryption

Decryption is exactly the same as encryption, except that P1, P2,..., P18 are used in the reverse order. Implementations of Blowfish that require the fastest speeds should unroll the loop and ensure that all Sub keys are stored in cache.

4.5 Flow chart of Blowfish algorithm

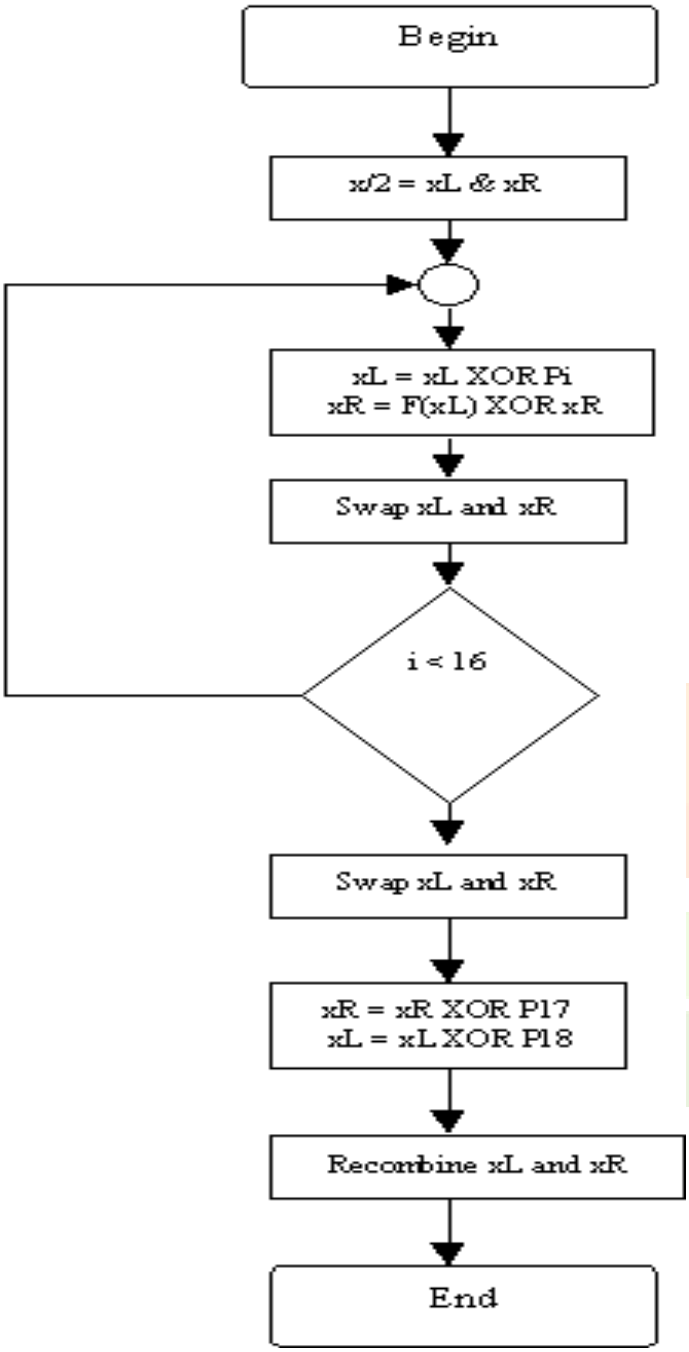
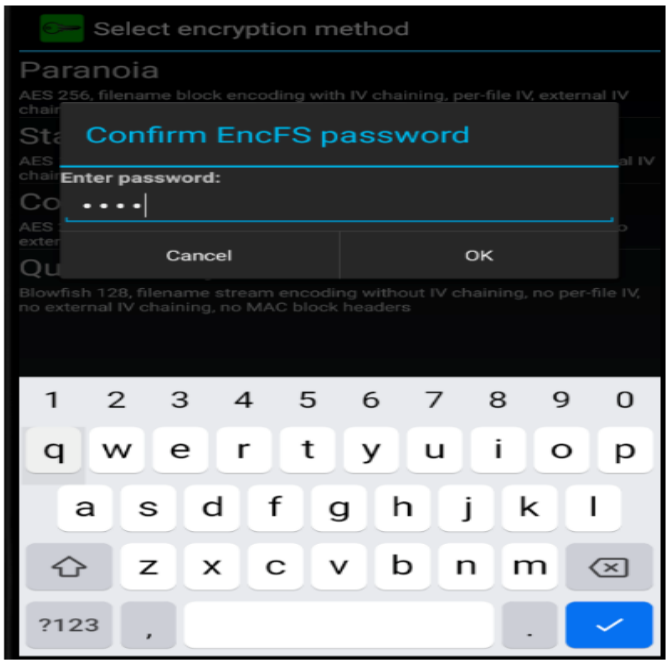


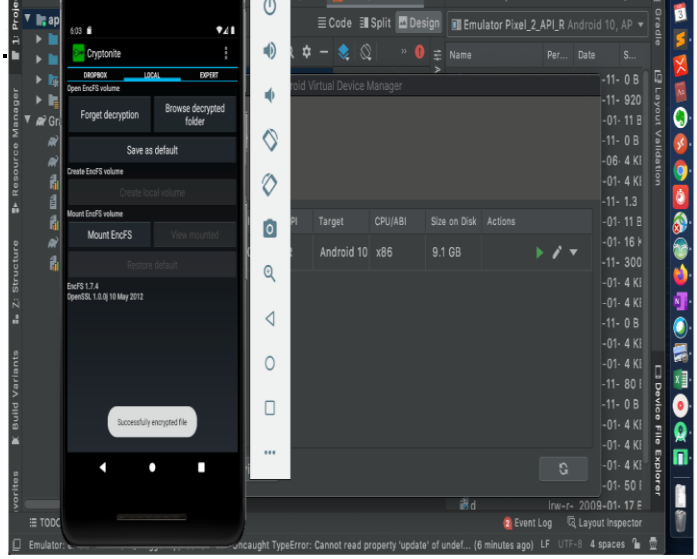
Fig 5: Flowchart of Blowfish Algorithm

5 RESULT

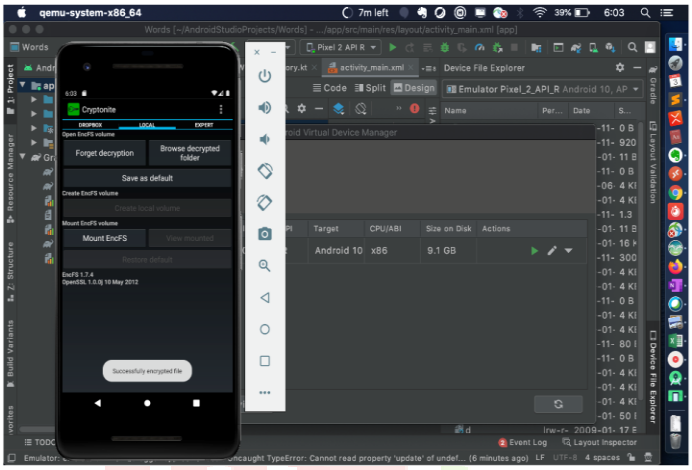
Snapshot 1 Select Target



Snapshot 2 Confirm Password



Snapshot 3 Final Output 1



Snapshot 4 Final Output 1

6 Conclusion

In this paper, presented an implementation of a portable filesystem encryption engine that uses NIST certified cryptographic algorithms for Android mobile devices. We offer a comparative performance analysis of our encryption engine under different operating conditions and for different loads including file and database (DB) operations. By optimizing the file system block-size and I/O mode, we were able to gain 20% to 57% performance. There are various encryption algorithms available like AES, DES, 3DES, blowfish. Therefore, we conclude that our encryption engine is easily portable to any Android device and the overhead due to the encryption scheme is an acceptable trade-off for achieving the confidentiality requirement.

References

[1] Zhaohui Wang, Rahul Murmuria, Angelos Stavrou," Implementing and Optimizing an Encryption File system on Android",2012

[2] Pratap Chnadra Mandal, "Superiority of Blowfish Algorithm", 2012.

[3] M. Anand Kumar and Dr.S.Karthikeyan," Investigating the Efficiency of Blowfish and Rejindael (AES) Algorithms",2012.

[4] Jawahar Thakur1 , Nagesh Kumar," DES, AES and Blowfish: Symmetric Key Cryptography Algorithms Simulation Based Performance Analysis",2011.

[5] Aditya Rajgarhia , Ashish Gehani," Performance and Extension of User Space File Systems",2010.

[6] "Android honeycomb encryption," http://source.android.com/tech/encryption/android crypto implementation.html.

[7] "Whispercore android device encryption," http://whispersys. com/whispercore.html.

[8] "Openssl fips 1402 security policy, version 1.2."

[9] Independent Security Evaluators, "Exploiting android," http://securityevaluators.com/content/case-studies/android/ index.jsp.

[10] S.Pavithra, Mrs. E. Ramadevi "STUDY AND PERFORMANCE ANALYSIS OF CRYPTOGRAPHY ALGORITHMS " International Journal of Advanced Research in Computer Engineering & Technology Volume 1, Issue 5, July 2012 14, pp.82-86

[11] Independent Security Evaluators, "Exploiting android," http://securityevaluators.com/content/case-studies/android/ index.jsp.

[12] J. P. Anderson, "Computer security technology planning study, volume II," Deputy for Command and Management Systems, HQ Electronics Systems Division (AFSC), L. G. Hanscom Field, Bedford, MA, Tech. Rep. ESD-TR-73-51, October 1972.

[13] "Fips pub 1402, security requirements for cryptographic modules." [Online]. Available: http://csrc.nist.gov/publications/ fips/fips1402/fips1402.pdf

[14] "Boost c++ library," http://www.boost.org/. [Online]. Available: http://www.boost.org/.

[15] "Librlog," http://www.arg0.net/rlog.