A Direction to Avoid Re-encryption in Cryptographic File Sharing

Lanxiang Chen, Dan Feng, Lingfang Zeng, and Yu Zhang

School of Computer, Huazhong University of Science and Technology,
Key Laboratory of Data Storage System, Ministry of Education, Wuhan, China
lxiangchen@gmail.com, dfeng@hust.edu.cn

Abstract. Almost all cryptographic file sharing systems need re-encryption when the sharing was revoked. These systems differ from each other only in the timing of re-encryption. As re-encryption is an expensive operation, it is significant to avoid re-encryption. The purpose of this paper is to advise a direction to avoid re-encryption and facilitate file sharing in cryptographic file sharing system. A Black-box model is set up to achieve this objective. In the model, FPGA or ASIC chips are used to act as the black-box as they have been extensively researched and applied in cryptography. Some applications of FPGA and ASIC in cryptography are detailed in this paper. Their feasibility to be functioned as the black-box is discussed. Also a software implementation on FPGA is attached with tested and effective performance.

Keywords: cryptographic file system, FPGA, ASIC, access control.

1 Introduction

It goes without saying the importance of information sharing. The rapid development of computer networks brings convenience for information sharing. As the computer network is open and pervasive, there are many security threats. How to securely share information? How to make you (adversary) can't 'see' the information even if you get the related data? One important way is to encrypt information before publishing. Only the authorized user can access the information. With this in mind, it gives birth to a set of cryptographic file sharing system.

Blaze's CFS [1] is the first cryptographic file system. But it was designed as a secure local file system, so it lacks features for sharing encrypted files among different users. The only way to share a protected file is to directly hand out keys for protected directories to other users. Compared to CFS, TCFS [2] makes file encryption transparent to users, provides data integrity, and enables file sharing between users of a group (UNIX group). There are similar systems such as SiRiUS [3] and SNAD [4]. All these systems need owner to provide file keys to share their files to others. Once owners want to revoke some users or the users leave their groups, owners have to re-encrypt their files as the keys have been exposed to the revoked users and distribute the new keys to the other un-revoked users. According to seven months of AFS protection server logs obtained from MIT, there were 29,203 individual revocations of

K. Li et al. (Eds.): NPC 2007, LNCS 4672, pp. 375–383, 2007.

users from 2,916 different access control lists (counting the number of times a single user was deleted from an ACL) [5]. Revocation will introduce masses of expensive cryptographic computation and key distribution to these systems.

To reduce the impact of aggressive re-encryption cost on performance, Cepheus [6] proposes the concept of lazy revocation which delays re-encryption to next file update. Plutus [5] exploits this concept. There are three types of re-encryption schemes when revocation:

1) Aggressive re-encryption – immediately re-encrypt data with a new key after a revocation.
2) Lazy re-encryption – delay re-encryption of the file to the next time it is updated or read. This saves encryption work for rarely-accessed files, but leaves data vulnerable longer.
3) Periodic re-encryption – change keys and re-encrypt data periodically to limit the window of vulnerability.

The difference of the three types of re-encryption is just the timing of re-encryption. All the above systems need re-encryption when revocation. There are systems that don't need re-encryption when revocation, such as NCryptfs [7]. NCryptfs file system is a stackable file system designed to provide kernel-level encryption services. Its file key is stored in the kernel memory, the user and the owner have to access files from the same machine. Since the encryption keys are always stored in the kernel's memory, it is never revealed to other users. Therefore, revocation of users does not require re-encryption. But the user has to (remember and) supply to the owner a hash of his password for every directory he wishes to access and the owner's machine must be online. Therefore, NCryptfs is quite inconvenient to use for distributed file sharing. As the key is pinned in memory, it is also vulnerable to attacks.

As discussed above, either system needs re-encryption or it needs not re-encryption but is inconvenient to file sharing. How to avoid re-encryption at the same time having convenient file sharing? In this paper a new direction is presented. Section 2 introduces the Black-box model. Section 3 will explain the technology which will be used to build the model. We will analysis feasibility in section 4 and conclude in section 5.

2 Model

To share files with others distributed everywhere, it is inevitable to transfer file data over network. Once data is on the wire, the best way to secure data is encryption. User can access the encrypted file only if he can decrypt the file: he must have the file key or some other special tools to complete. If the user has file key, when file owner doesn't hope him to access the file or the user leaves the group that is authorized to access the file, the file owner has to re-encrypt the file as the file key is exposed to the revoked user. How to share file with others without revealing key to them? The possible way is to access the shared file as NCryptfs, file key just only exists in owner's kernel memory. All users who want to access the file have to supply to the owner a hash of his password for every directory he wishes to access. First, it is quite

inconvenient to use for distributed file sharing. Also, once owner is compromised, all users can't access files. Thirdly, as the key is pinned in memory, it is also vulnerable to attacks.

Is there a scheme to avoid re-encryption while having convenient file sharing?

We already know the requirements. It is to avoid re-encryption while sharing file with others conveniently. To avoid re-encryption, the only way is not to reveal file key to others while they can use the file key to decrypt the file. So the file key should be encrypted using their private key. User who can access file can get encrypted file and corresponding encrypted key. There are also other requirements such as the scheme is easy to configure, portable and low cost etc. A black box is supposed here. The encrypted file and encrypted key are the inputs of the black-box. And it outputs the decrypted file which we wish to get. Figure 1 is the illustration. The Black-box should be implemented utilizing existing technology and have preferable performance and is transparent to users.

Fig. 1. Black-box model with inputs of encrypted key and ciphertext and it outputs cleartext

The Black-box may be implemented in many ways. But as Field Programmable Gate Array (FPGA) and Application-Specific Integrated Circuit (ASIC) technologies have been extensively researched and applied in Cryptography, we think they are the most suitable to act as Black-box. Along with avoiding re-encryption, hardware-based cryptographic solutions can provide significant security and performance improvements over software solutions.

3 The Application of VLSI in Cryptography

Very Large Scale Integration (VLSI) circuit is the field which involves packing more and more logic devices into smaller and smaller areas. Obeying Moore's law, the capability of an integration circuit has increased exponentially over the years, in terms of computation power, utilization of available area, yield. People can now put diverse functionality into the integration circuit, opening up new frontiers. Examples are embedded systems, where intelligent devices are put inside everyday objects, and ubiquitous computing.

The application of VLSI has been extensively researched in cryptography. These researches can be categorized into two types: the ASIC implementations and the FPGA implementations. The ASIC implementations have the advantage of fully optimized structure and thus resulted in smaller circuit area, higher speed of operation, and lower power consumption. But the design and implementation of ASIC is complex and time consuming and the cost is very high. The ASIC circuit can not be modified once it has been implemented. So it can not be adopted to often changed

environment. Most of these designs were carried out on reconfigurable platforms. The reconfigurable platforms make use of the FPGA technology which combined the high speed of specialized hardware architecture and the agility of the software platform. And also the FPGA implementations cost much less than the ASIC.

Existing cryptographic algorithms utilizing ASIC and FPGA cover various fields, like AES [8], DES, SHA, HMAC and RSA [23]. For different design objectives and requirement, there are many design alternatives [9] as follows:

- Low area, low bandwidth designs, and high area, high bandwidth designs,
- Iterated architectures (frequent feedback), and fully unrolled pipelined architectures (zero feedback),
- Designs where part of the logic is executed using pre-computed SRAM operations, and designs where no pre-computed tables of SRAM are used,
- Designs with pre-computed key/round material, and designs with runtime generation of key/round material with the data to be encoded,
- Designs supporting dynamic selection of variable key sizes, and designs supporting a singular fixed key size,
- Designs supporting generic block-cipher encryption / decryption, and designs supporting full-duplex encryption and decryption paths,
- A wide range of target hardware chipsets and architectures,
- A broad range of power consumption objectives.

From the first ASIC implementation [10] of AES, there are serials of related implementation schemes [11, 12]. The best performance implementation of AES-ECB 128-bit on ASIC is Hodjat's [13]. It uses 473K gates with 606MHz clock frequency, and its highest speed is 77.6Gbps. The best performance implementation of AES-ECB 192-bit and 256-bit on ASIC is North Pole Engineering's [14]. It uses 26K gates with 323MHz clock frequency, and its highest speed is 41.3Gbps. The best performance implementation of AES-FEEDBACK on ASIC is Morioko's [15]. It uses 168K gates with 909MHz clock frequency and its highest speed is 11.6Gbps.

References [16-19] are some of the early implementations of the Rijndael algorithm before it was accepted as the Advanced Encryption Standard on FPGA. There are also serials of implementation schemes [20, 21].The best performance implementation of AES-ECB 128-bit on FPGA is Fu's [21]. It uses 17887 slices with 212.5MHz clock frequency, and its highest speed is 27.1Gbps. The best performance implementation of AES-ECB 192-bit and 256-bit on FPGA is North Pole Engineering's [14]. It uses 5840 slices with 100MHz clock frequency, and its highest speed is 12.8Gbps. The best performance implementation of AES-FEEDBACK on FPGA is Helion Tech's [22]. It uses 447 slices with 219MHz clock frequency and its highest speed is 25.48Gbps.

Implementations of RSA on ASIC refer to [24, 25] and implementations on FPGA can refer to [26, 27]. The best performance implementation of RSA is the implementation of McIvor [28].

Due to space limitations, the process of other related algorithms implementation won't be discussed.

4 Feasibility

Using FPGA and ASIC to act as the Black-box, they must satisfy following characteristics.

- User doesn't know the private key which is used to encrypt the file key. Usually, user should know their private key. As we can't expose the file key to user and the file key is encrypted with the private key. So each user's Black-box can generate public-private pair for user and submit the public part to user. As only Black-box knows user's private key, it can sign for user,
- User can't change Black-box data flow. Once the Black-box is distributed to users, they can't modify the Black-box,
- Convenience to file sharing. We will illustrate a FPGA scheme to show how to share files conveniently,
- Performance and cost. The Black-box should have preferable performance and acceptable cost.

As the ASIC circuit can not be modified once it has been implemented and FPGA is through configure file to set work pattern. The configure file is binary file and there isn't way to compile the file in reverse. How FPGA is organized is not known by anyone else except the designer. So they both satisfy characteristic two. As detailed above, the performance of ASIC and FPGA are excellent.

4.1 An FPGA Scheme

Figure 2 shows a simple architecture of the FPGA scheme. The files are stored to storage device by owners. File owner establishes ACL (Access Control List) according to local policy in which the owner defines who can access the file and uses whose public key to encrypt the file key. The encrypted file key is inserted to ACL and stored in storage device. The client who wants to access the file, he fetches file data and ACL from storage device. The ACL is signed by owner then client first verify the ACL. There is timestamp in ACL to avoid the client reuses the ACL. Client knows whether he can access the file from the ACL. He can fetch corresponding file key encrypted using his public key. Figure 3 illustrates the data flow between user and storage device.

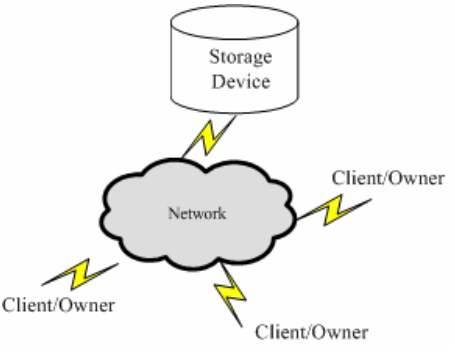

Fig. 2. Storage architecture

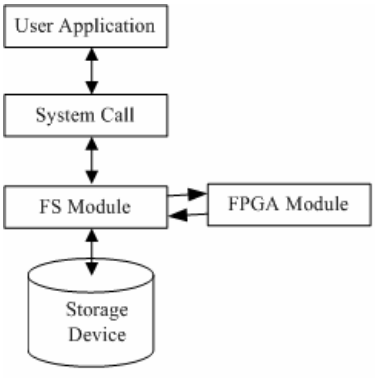

Fig. 3. Data flow between user and storage device

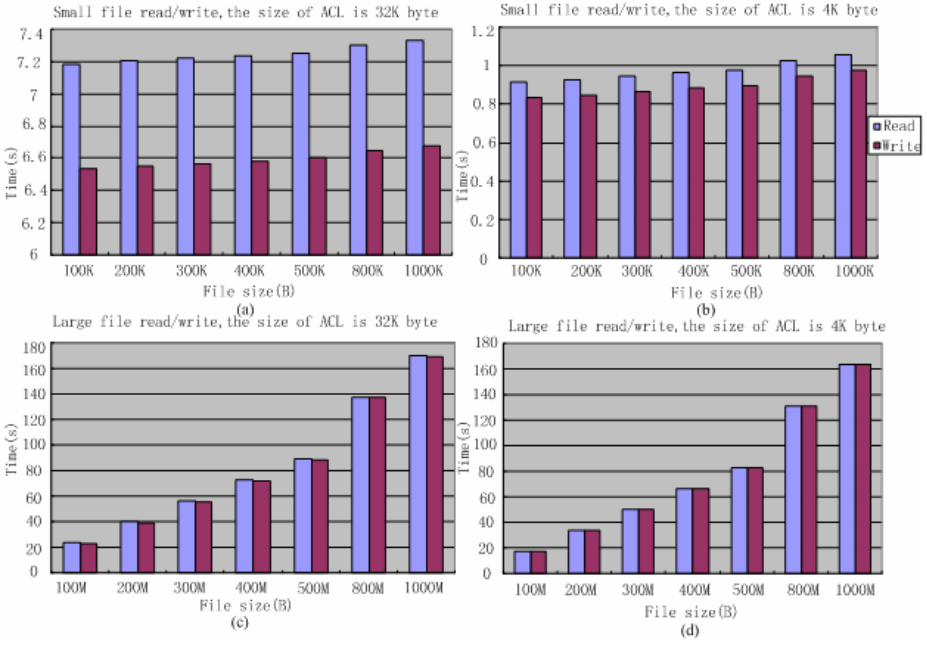

Fig. 4. (a)Small file read/write with a 32K bytes ACL. (b) Small file read/write with a 4K bytes ACL. (c) Large file read/write with a 32K bytes ACL. (d) Large file read/write with a 4K bytes ACL5 Conclusion and Future Work.

Client application program requests system call to read file from storage device. File system module verifies the ACL and fetches corresponding file key, then sends the encrypted file key and file to FPGA module which decrypts file key and encrypted file. A write request first causes system to create file key and establish ACL. Client system encrypts the file using the file key and requires FPGA module to sign the ACL, then attaches the signed ACL to encrypted file and sends to storage device. It

only needs to provide file and file key for user and the rest work is transparently completed by FPGA module. As user doesn't know file key, it needs not to re-encryption when revoking the user. And this way authentication server or other center server can be eliminated, so it is well suitable to distributed file sharing.

4.2 Experimental Results

In this section, a software solution on XC4VLX200 is implemented. The read/write costs on FPGA are tested in Figure 4. The AES in 128-bit ECB mode and 1024-bit RSA are used and the size of ACL is limited to 32K bytes. The cost includes verification and decryption for a file read, correspondingly signature and encryption for a file write. The costs of small file read/write and large file read/write with small size ACL and large size ACL respectively are illustrated. Figure 4 (a) and (b) demonstrate the costs of small file read/write, and Figure 4 (c) and (d) demonstrate the costs of large file read/write. It indicates that read operation is slower than write and the size of ACL affects the costs of read/write greatly. It results from the operation of RSA which is time-consuming operation.

5 Conclusion and Future Work

In this paper, a direction is proposed which can avoid re-encryption when revocation in cryptographic file sharing system. As re-encryption is an expensive operation, it is significant to avoid re-encryption. We set up a Black-box model which is used to avoid re-encryption. In the model, FPGA or ASIC chips are used to act as the Black-box. The application of FPGA and ASIC in cryptography is detailed and their feasibility to function as the Black-box is discussed. We demonstrate the feasibility through a software implementation on FPGA with tested and effective performance.

The software implementation on FPGA just testifies that it is feasible, however it can't reflect the impact of avoiding re-encryption on performance. And the performance of software implementation on FPGA is far worse than hardware implementation. So we can't compare the performance of our scheme with other cryptographic file system described above. The future work is to evaluate the impact of re-encryption on performance quantitatively and compare the performance of our scheme using hardware implementation on FPGA with other cryptographic file system.

Acknowledgments. This work was supported by the National Basic Research Program of China (973 Program) under Grant No. 2004CB318201, and the National Science Foundation of China under Grant No. 60603048.

References

1. Blaze, M.: A Cryptographic File System for Unix. In: First ACM Conference on Communications and Computing Security, Fairfax, VA, November 1993 (1993)
2. Cattaneo, G., Catuogno, L., Persiano, P., Sorbo, A.D.: Design and implementation of a transparent cryptographic file system for UNIX. In: FREENIX Track: 2001 USENIX Annual Technical Conference (2001)

3. Goh, E.-J., Shacham, H., Modadugu, N., Boneh, D.: SiRiUS: Securing Remote Untrusted Storage. In: Proceedings of the Tenth Network and Distributed Systems Security (NDSS) Symposium, pp. 131–145 (2003)
4. Miller, E.L., Long, D.D.E., Freeman, W.E., Reed, B.C.: Strong security for network-attached storage. In: Proceedings of the 2002 Conference on File and Storage Technologies, Monterey, CA, pp. 1–13 (2002)
5. Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., Fu, K.: Plutus: scalable secure file sharing on untrusted storage. In: USENIX File and Storage Technologies (2003)
6. Fu, K.: Group sharing and random access in cryptographic storage file system, Master's thesis, MIT (1999)
7. Wright, C.P., Martino, M.C., Zadok, E.: Ncryptfs: A secure and convenient cryptographic file system. In: USENIX Annual Technical Conference (2003)
8. National Institute of Standards and Technology (NIST), Advanced Encryption Standard (AES), Federal Information Processing Standards Publications, vol. 197 (2001)
9. Gittins, B., Landman, H., O'Neil, S., Kelson, R.: A Presentation on VEST Hardware Performance, Chip Area Measurements, Power Consumption Estimates and Benchmarking in relation to AES, SHA-256 and SHA-512 (14th November 2005)
10. Verbauwhede, I., Schaumont, P., Kuo, H.: Design and Performance Testing of a 2.29 Gb/s Rijndael Processor. IEEE J. Solid-State Circuits (JSSC 2003), 569–572 (2003)
11. Su, C.-P., Horng, C.-L., Huang, C.-T., Wu, C.-W.: A configurable AES processor for enhanced security. In: ASP-DAC, pp. 361–366 (2005)
12. Hodjat, A., Verbauwhede, I.: Area-Throughput Trade-Offs for Fully Pipelined 30 to 70 Gbits/s AES Processors. IEEE Trans. Computers 55(4), 366–372 (2006)
13. Hodjat, A., Verbauwhede, I.: Speed-area trade-off for 10 to 100 Gbits/s throughput AES processor. In: 2003 IEEE Asilomar Conference on Signals, Systems, and Computers (November 2003), http://www.ee.ucla.edu/ãhodjat/AES/asilomar_paper_alireza.pdf
14. "AES Core", North Pole Engineering, http://www.northpoleengineering.com/aescore.htm
15. Morioka, S., Satoh, A.: A 10 Gbps Full-AES Crypto Design with a Twisted-BDD S-Box Architecture. In: IEEE International Conference on Computer Design (ICCD 2002) (2002)
16. Gaj, K., Chodowiec, P.: Fast Implementation and Fair Comparison of the Final Candidates for Advanced Encryption Standard Using Field Programmable Gate Arrays. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 84–99. Springer, Heidelberg (2001)
17. Ichikawa, T., Kasuya, T., Matsui, M.: Hardware Evaluation of the AES Finalists. In: Proc. Third AES Candidate Conf. (2000)
18. Gaj, K., Chodowiec, P.: Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware. In: Proc. Third Advanced Encryption Standard Candidate Conf (AES3 2000), pp. 40–54 (2000)
19. Fischer, V.: Realization of the Round 2 Candidates Using Altera FPGA. In: Comments Third Advanced Encryption Standard Candidates Conf. (AES3 2000) (2000)
20. Rouvroy, G., Standaert, F.-X., Quisquater, J.-J., Legat, J.-D.: Compact and Efficient Encryption/Decryption Module for FPGA Implementation of AES Rijndael Very Well Suited for Small Embedded Applications. In: ITCC 2004, special session on embedded cryptographic hardware, vol. II, pp. 583–587. IEEE Computer Society, Los Alamitos (2004)
21. Fu, Y., Hao, L., Zhang, X., Yang, R.: ICESS 2005. LNCS, vol. 3820. Springer, Heidelberg (2005)
22. AES Core for FGPA and ASIC, Helion Technology, http://www. heliontech.com/core2.htm
23. Rivest, R.L., Shamir, A., Adleman, L.M.: A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Communications of the ACM 21(2), 120–126 (1978)

24. M.-S., F.J., Kang.: A Novel Systolic VLSI Architecture for Fast RSA Modular Multiplication. In: Proceedings of the 2002 IEEE Asia-Pacific Conference on ASIC 2002, pp. 81-84 (2002)
25. Yeşil, S., İsmailoğlu, N., Tekmen, Ç., Aşkar, M.: Two Fast RSA Implementations Using High-Radix Montgomery Algorithm. In: 2004 IEEE International Symposium on Circuits and Systems, pp. 557–560 (2004)
26. Blum, T., Paar, C.: Montgomery Modular Exponentiation on Reconfigurable Hardware. In: Proceedings 14th IEEE Symposium on Computer Arithmetic, pp. 70–77 (1999)
27. Cilardo, A., Mazzeo, A., Romano, L., Saggese, G.P.: Carry-Save Montgomery Modular Exponentiation on Reconfigurable Hardware. In: Procs of the Design, Automation, and Test in Europe Conference (DATE 2004) (2004)
28. McIvor, C., McLoone, M., McCanny, J.V.: High-Radix Systolic Modular Multiplication on Reconfigurable Hardware. In: IEEE International Conference on Field Programmable Technology, pp. 13–19 (2005)