Fuzzy-search on Content-Addressable Filesystem with Threshold Cryptosystem

Aufar Gilbran - 13513015
Informatics / Computer Science
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
aufargilbran@gmail.com

***Abstract*—This paper aims to provide a unique approach to create a applicative usage of threshold cryptosystem, specifically secret sharing, on content-addressable filesystem. Through some careful file structure modification, secret sharing cryptosystem can be used to do fuzzy-search on these filesystems which files is addressed by its full content. Fuzzy-searching allow us to get content of the file by its partial content, which significantly increase the usability of this type of filesystem to be used by human.**

Keywords*—threshold cryptosystem, secret sharing, content-addressable filesystem

I. Introduction

Filesystems are the core structure responsible for managing files created by a user. Filesystem help us to read, write, delete and update the file by receiving what we wanted the to be and update the block of data located in the disk accordingly. There's no computer system that doesn't use filesystem because most user want to have some data to be persistent, thus filesystem is essential. Since disk access is very slow, a good filesystem needs to manage operations done by user efficiently, that is with as minimum disk access as possible.

There are many types of filesystem, some are specific to the device/medium and some are created for efficient disk access but all of those filesystems are created to meet the user demand. The most important user demand is functionality. User want filesystems that provides functionality that supports the operation that the user wanted. We could have a filesystem that implement all known functionality that any user demand, but any user wouldn't want to use a monolithic filesystem because it would take too much time to run user operations. Because of that, filesystems are created for a specific cause in mind.

One of the many filesystem types that are known are the content-addressable filesystem. Content-addressable filesystem manage user files by mapping between the digest of the file and its location on the disk. Thus, to locate a file in the filesystem, the user just need to provide its content digest to the filesystem and the filesystem look at the mapping and then return the located file to the user. This is done in a very quick manner because there's no need to traverse a directory structure like general usage filesystem. The problem is, a content-addressable are limited in functionality because we need to keep track on all the digest of a file content. Of course, it is the system that keep track of the mapping between file digest and file location, but what if the user need a specific file with only some of its content?

There is a technique that is commonly used to search by partial content called fuzzy-searching. Fuzzy-searching let us find our desired string with some of its non-overlapping substring. This technique is useful for human because most of the time we cannot remember a full, exact string that we desired but it is easier for to remember that the string we are searching have some keywords. The goal of this technique is to retrieve our string with keywords that we do remember. This technique is useful for the kind of functionality that we need in a content-addressable filesystem to make usage of the filesystem easier.

The next problem is how to verify the file with some of its partial content when what we have is only the file content digest. To solve this problem, we need to do some modification to how a file is structured in normal content-addressable filesystem. We also need to be able to verify a string with some other partial string. This could be done with the threshold cryptosystem.

In this paper, we will discuss the content-addressable filesystem, how it was structured and used, and how we can add fuzzy-searching functionality with threshold cryptosystem (specifically secret sharing) that can be useful on this type of filesystem.

II. Fundamental Knowledge and Theorem

The following knowledges and theorems are essential to implement fuzzy-searching functionality on a content-addressable filesystem.

A. Content-addressable Filesystem

Knowledge of filesystem is the key to impelement a new functionality to a filesystem. Specifically, knowledge of content-addressable filesystem is crucial to understand how to integrate the functionality into our filesystem. But first, before we can implement a new functionality to a filesystem, we need to each term that may be required to understand the implementation of the new functionality.

File is a form of abstraction used for storing information, which is available to a computer program and is usually based on some kind of persistent storage. A file need to be persistent in the sense that it remains available for other programs to use after the program that created it has finished executing.

Filesystem is used to control how data is stored and retrieved. Without a filesystem, information placed in a storage area would be one large body of data with no way to tell where one piece of information stops and the next begins. By separating the data into blocks of information and giving each block a specific name, the information is separated and can be easily identified.

There are many different kinds of filesystems. Each one has different structure and logic, properties of speed, flexibility, security, size and more. Some filesystems have been designed to be used for specific applications. For this paper, we only need to understand the content-addressable filesystem.

Content-addressable filesystem is based on content-addressable storage also referred to as associative storage, which mechanism is for storing information that can be retrieved based on its content, not its storage location. It is typically used for high-speed storage and retrieval of fixed content. Content-addressable filesystem add one more level of abstraction from content-addressable storage to a filesystem, thus the operations return a digest of the file which can be mapped into its storage location.

Knowledge of content-addressable filesystem is

B. *Shamir's Secret Sharing Threshold Cryptosystem*

Threshold Cryptosystem is a cryptosystem such that, in order to decrypt an encrypted message in the system, several key holders (more than some threshold number) must cooperate in the decryption process. The message is encrypted with a public key and then the corresponding private key is distributed among the participating parties.

Let n be the number of key holders. The threshold cryptosystem is called *(k,n)*-threshold, if at least t of these key holders can efficiently decrypt the ciphertext, while less than k of these key holders will not be able to get any useful information. Obviously it is possible to define *(k,n)*-threshold signature scheme, where at least k parties are required for creating a signature. Threshold versions of encryption schemes can be built for many public encryption schemes. The goal of these schemes is to be more secure as the original scheme.

Shamir's Secret Sharing is an algorithm in cryptography created by Adi Shamir, creator of RSA. It is a form of secret sharing, where a secret is divided into different shares, giving each participant its own unique share. Some of the shares or all of them are needed in order to reconstruct the secret.

The following is the algorithms to distribute the secret to each of key holders:

- Suppose we want to use a *(k,n)*-threshold scheme to share our secret S.
- Choose random prime P number and $k-1$ positive integers $a_1, a_2, \ldots, a_{k-1}$, with $a_i < P$. Let $a_0 = S$. Build the polynomial $f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{k-1}x^{k-1}$.
- To give the *i-th* participant their share, we need to compute $(n_i, f(n_i))$, with n_i is a unique integer assigned to the *i-th* participant. It should be different with n_j for $i \neq j$. The computation result it the share for the *i-th* participant

To resconstruct the secret, we can use any subset of k of key holder's share and find the coefficient of the polynomial using polynomial interpolation. The secret is then the constant term a_0.

C. *Fuzzy-search*

In string theorem, fuzzy searching is the technique to find desired strings provided a pattern with approximations (rather than exactly equal). The closeness of a match is measured in terms of the number of operations necessary to convert the string into an exact match. This number is called the edit distance between the string and the pattern.

However, in this paper we only use fuzzy-search as a concept and functionality. That is, the user can provide some patterns and the algorithm then returns the string desired. In fact, we will not use the standard measurement of closeness to find the exact match and instead use Shamir's Secret Sharing decryption process to validate the correctness.

III. Proposed Design

For fuzzy-searching functionality become available in our filesystem, we need to have our own design of the filesystem. This design should describe the following points:

- File structure
- File operations
- Filesystem operations

For simplicity, we also need an assumption that holds true for every operation and structure in the filesystem without loss of generality. This makes sure that the explanation for the design is a simple as possible but general enough to be used at different type of machines and devices.

Let us call the new designed filesystem as AGFS with the assumption that every file on AGFS is a plaintext file and filesystem operations are operating on byte level. AGFS will have standard architecture and operations like other content-addressable filesystem but with some modifications on file structure and operations addition. These modifications and addition are required to keep the needed data to verify fuzzy-searching closeness to its corresponding data and also define operations interface between user program and the filesystem.

We also assume that the file “cannot” be changed. To change the file, instead of updating the current blocks we delete the old file and write new file. This mechanism is used because when we change the file on a content-addressable filesystem, we need to update all the header of the file. This makes the usual update operations pointless and we can just discard update operations to keep things simple.

Before we look at each filesystem architecture, we need to define how user can provide the pattern for fuzzy-searching. We use a variant of fuzzy-search, that is we do not provide a single pattern but multiple patterns. We will call the possible pattern for fuzzy-searching as *tag* for the rest of the paper. Filesystem users will provide tags to locate file in fuzzy-search manner. These tags are generated from the content of the file that is split by each words. Each unique word then become a tag for fuzzy-searching.

The following are modifications from standard content-addressable filesystem made to support the fuzzy-searching functionality.

A. *File Structure*

File structure in a filesystem commonly have file entry, file header, and file content sections (or blocks). These section defines and keep information required for the filesystem to process file operations applied to the file and pointers to locate related file sections.

In AGFS, the file structure is defined as follows:

- File Entry

  The file entry is a block in the upper level of storage area that function as an entry point of a file. This entry point should have a pointer to the file header location on disk and the location of the file content on disk. This block include informations that can be used by the filesystem to determine what can be done on the file by user.

  For AGFS, the file entry should keep additional informations required to construct the secret that is distributed using Shamir's Secret Sharing scheme. The secret is the file header explained later. These information should be capable to do a mapping between fuzzy-search tag that is possibly provided by the user to the actual share linked to the pattern.

  The mapping is done by using one property of Shamir's Secret Sharing scheme, that is, to construct the secret on *(k,n)*-threshold scheme, only *k* subset out of *n* is required. These means that the shares is actually interchangeable. Suppose that we have $h_1, h_2,$ and h_3 with h_i is the *i-th* participant. If the 3rd participant were to suddenly bail out of the cooperation, we can still proceed the decryption process with 4th participant or 5th participant share. This property means that each tag can be assigned with whatever shares currently available. Thus, populating the mapping table becomes trivial task.

- File Header

  The file header is one or more blocks that keeps informations about the files, so the filesystem know what to do to process the user request operation for file. In standard content-addressable storage, the file header section also define the file content digest which usually used to point to the file location in the storage. Usually, on content-addressable filesystem and its derivatives, the file content digest is not used as the digest already points to the file entry which have the content's location on the storage. In our new filesystem, we will exploit the digest and exploit its convention to add information required by our filesystem.

  For AGFS, the file header section is used to actually validate the closeness of the file with tags provided by user. We added a new field to the header that is a *magic string* unique to our filesystem. Magic string is a string that exists solely for the purpose of verifying. In our cause, we want to know wether or not the secret construction actually contains the magic string.

  File header in AGFS act as a secret that is distributed by the Shamir's Secret Sharing scheme. This can be done as follows:

  ◦ The file header has magic string that besides act as a verification string, it also act as a pad to make sure the total file header size is 64-bit.

  ◦ The file header (including file content digest and file type) is used as a secret and then distributed using Shamir's Secret Sharing scheme to get the shares. The scheme used is proportional to the number of tags available. So if there's *n* tag and we only require at least *k* tag to construct the secret, we use the (*k*,*n)*-threshold scheme to distribute the share.

  Since the file header also includes the file content digest, each file with the different content or different file type will have different secret. Thus, the shares will be different if the file is different. This implies that different share will not be able to construct the original file header and the filesystem will decide that the file is not a match since the magic string will not be the same.

  At the creation of the file, the filesystem need to create the mapping table between the shares and fuzzy-search tags locatable from file entry section. Implementation details of mapping table creating is explained later in this paper.

B. *File Operations*

The only change we need to introduce to the file operations are the offset of the file header. Since there is the newly added magic string *before* the standard file header, we need to make sure that every file operations start operate with additional *B* bit offset, with B is the magic string size.

C. *Filesystem Operations*

File operations are defined as an interface between user programs and the filesystem. When user program need to do a specific operation on the file, it calls the filesystem operation defined in the filesystem with the interface proper parameters to modify the file.

Some filesystem operations in AGFS needs to be modified to accommodate the fuzzy-searching functionality. The modification takes place on CREATE, UPDATE, and OPEN operations with the modification as follows:

- CREATE

  File creation is provided by the CREATE operation on filesystem. In standard content-addressable filesystem, the CREATE operation will do the following:

  - Check to see if the file already exists by checking the digest existence.
  - Allocate Space on the disk for the file.
  - Insert a pointer to the first block of the file entry.

  In AGFS, we add more steps to add mapping table between fuzzy-searching tags and secret shares distributed by Shamir's Secret Sharing scheme with file header as the secret. The additional steps are as follow:

  - Create a list of fuzzy-searching tags created by splitting the file content by words.
  - Create a list of shares distributed by Shamir's Secret Sharing scheme.
  - For each tag in tag list, assign to any share on the share list that is not assigned yet as the corresponding share value for the tag.
  - Write a pointer on the file entry section to the first tag-share mapping.

  After these operations, the file entry section will have a pointer to list of tag-share mapping.

- UPDATE

  As mentioned before, the update operation are to be deprecated and replaced by the DELETE and CREATE operation. Although we say replaced, to make the filesystem to be compatible with current technologies, we only change the logic of UPDATE operations to call CREATE and DELETE. The UPDATE operation will work as follow:

  - Keep the file before update in the memory.
  - Apply the update to the file in memory.
  - Create new file with its content from the file in memory.
  - If the new file is created successfully, delete the old file.

  This mechanism make sure that the tags and content digest are always updated atomically.

- OPEN

  Because we added a magic string *before* the digest, we need to slightly change the OPEN operation to read the message digest *B* bits *after* the pointed location provided by file entry section with *B* is the size of the magic string.

Other than these operations, filesystem operations are the same with standard content-addressable filesystem.

There is additional operation that is needed for the fuzzy-search to be working as intended. We will call operation as SEARCH. The operation is working as follows:

- SEARCH

  Fuzzy-finding is provided by this operation. The operation take list of tags as its parameter, and then reconstruct the secret for each file header in the storage. If the secret is valid, that is the magic string is identifiable, then the file entry is a match.

Aside from these modifications, we use the standard architecture from content-addressable filesystem for our filesystem. The overall architecture of the new filesystem are illustrated by Figure 1.

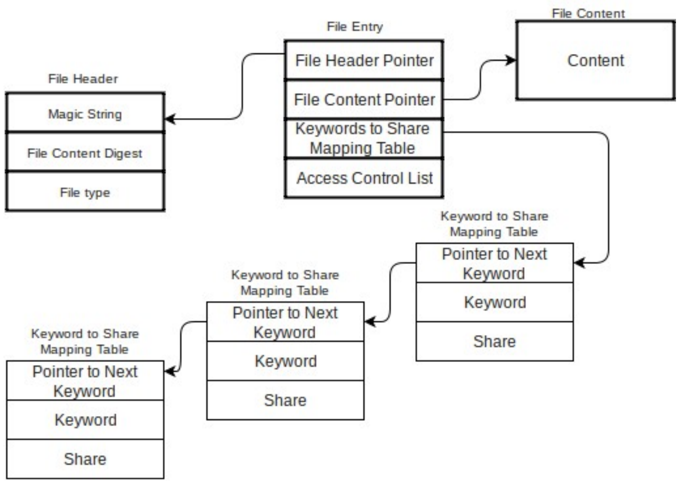

Figure 1. Overall filesystem and file architecture

IV. Experiment and Analysis

Experimentation on AGFS will done mostly on the fuzzy-searching functionality that is added on top of standard content-addressable filesystem.

The experiment will be done in these scenarios:

- The created a new file on AGFS and then fuzzy search the new file using some of its content
- Fuzzy-search with wrong share, but the correct ones still above the threshold
- Fuzzy-search when there is two or more files

For these experiment, we will assume that the magic string is *7c60* in hexadecimal representation. Also, the experiments will be done with *(2, n)*-threshold scheme, such that it is required to have at least 2 tags to get a match.

A. File creation and fuzzy search on AGFS

This scenario assume that standard content-addressable filesystem operation are working as intended. Thus, we only need to check if the mapping actually works.

The following table describe the experiment result (bold is input and normal is output:

File 1			
File Header (Hexadecimal Representation)			
7c600ac6b5f3b72f			
File Content			
fuzzy searching is awesome			
Mapping Table			
At 0x0000	**At 0x0010**	**At 0x0100**	**At 0x0110**
Next: 0x0010	Next: 0x0100	Next: 0x0110	Next: NULL
Tag: fuzzy	Tag: searching	Tag: is	Tag: awesome
Share: 801404bd12a4e49bc6f5e	Share: 80283121f4acb506d1acd	Share: 803c225ae6a43ac22c2bc	Share: 80418a09e8adc62d2f0f6
Fuzzy-search			
Tags: fuzzy, is File mapped shares: [801404bd12a4e49bc6f5e, 803c225ae6a43ac22c2bc] Secret: 7c600ac6b5f3b72f			

Table 1. File creation and fuzzy-search experiment result

As seen from the experiment result in Table 1, the file operations CREATE created a linked list mapping with each element in the list map a tag to a share and vice-versa. It also correctly construct the secret in which the filesystem can verifiy the existence of the magic string (*7c60)*.

From the experiment result, we know that the mapping table size will be proportional to the amount of unique words in the file content. This is a waste of space since basically we need *O(n)* more space for *n* words file. This may or may not pose a problem, depending on the system specific needs.

The fuzzy search also has horrible performance since all the tags need to be matched against all possible tags listed in the file mapping tables. The complexity to match a file is *O(WT)*, with *W* is the number of words in the file and *T* is the number of tags the fuzzy-search provides.

B. Fuzzy-search with the wrong share

For this scenario, we will use the last experiment table, but with different fuzzy-searching parameters given to the filesystem.

File 1			
File Header (Hexadecimal Representation)			
7c600ac6b5f3b72f			
File Content			
fuzzy searching is awesome			
Mapping Table			
At 0x0000	**At 0x0010**	**At 0x0100**	**At 0x0110**
Next: 0x0010	Next: 0x0100	Next: 0x0110	Next: NULL
Tag: fuzzy	Tag: searching	Tag: is	Tag: awesome
Share: 801404bd12a4e49bc6f5e	Share: 80283121f4acb506d1acd	Share: 803c225ae6a43ac22c2bc	Share: 80418a09e8adc62d2f0f6
Fuzzy-search			
Tags: fuzzy, are, awesome File mapped shares: [801404bd12a4e49bc6f5e, 00000000000000000000, 80418a09e8adc62d2f0f6] Secret: 004bd12a4e49bc6f5e **Tags: fuzzy, searching, are** File mapped shares: [801404bd12a4e49bc6f5e, 80283121f4acb506d1acd, 00000000000000000000] Secret: 7c600ac6b5f3b72f			

Table 2. Fuzzy-search with wrong share included result

The experiment result from Table 2 shows us that if there's a wrong share included the algorithm correctness is determine whether there is enough correct shares before the wrong share. This undeterministic behavior is very bad for a filesystem. Thus, it is better to just let the match fail when there is a tag that does not exist in the mapping table.

C. Fuzzy-search when there is two or more files

For this scenario, because a different file will unlikely to have the same mapping table (which will increase the likelihood of wrong share), we assume that the filesystem will let the match fail when there is a tag that does not exist in the mapping table. The filesystem will inform the user program with exception.

We will use the previous file in the filesystem and create a new file to test this. The only difference of these file will be the content of these two files, which makes the content digest on the file header to be different too.

File 1
File Header (Hexadecimal Representation)

7c600ac6b5f3b72f			
File Content			
fuzzy searching is awesome			
Mapping Table			
At 0x0000	**At 0x0010**	**At 0x0100**	**At 0x0110**
Next: 0x0010	Next: 0x0100	Next: 0x0110	Next: NULL
Tag: fuzzy	Tag: searching	Tag: is	Tag: awesome
Share: 801404bd12a 4e49bc6f5e	Share: 80283121f4ac b506d1acd	Share: 803c225ae6a 43ac22c2bc	Share: 80418a09e8a dc62d2f0f6
Fuzzy-search			
Tags: fuzzy, is, awesome File mapped shares: [801404bd12a4e49bc6f5e, 803c225ae6a43ac22c2bc, 80418a09e8adc62d2f0f6] Secret: 7c600ac6b5f3b72f **Tags: awesome, cryptosystem** File mapped shares: [801404bd12a4e49bc6f5e, 0000000000000000000] Secret: failed match exception by filesystem			
File 2			
File Header (Hexadecimal Representation)			
7c60c3fbe3d14c2f			
File Content			
awesome threshold cryptosystem			
Mapping Table			
At 0x0000	**At 0x0010**	**At 0x0100**	
Next: 0x0010	Next: 0x0100	Next: NULL	
Tag: awesome	Tag: threshold	Tag: cryptosystem	
Share: 8011e3b67aeeabcc f036a	Share: 8023ff26e19d95de dd2a5	Share: 80320b56974c802f 39de0	
Fuzzy-search			
Tags: fuzzy, is, awesome File mapped shares: 0000000000000000000, 0000000000000000000 80320b56974c802f39de0] Secret: failed match exception by filesystem **Tags: awesome, cryptosystem** File mapped shares: [8011e3b67aeeabccf036a, 80320b56974c802f39de0] Secret: 7c60c3fbe3d14c2f			

Table 3. Fuzzy-search when there is two or more files experiment result

Experiment result from Table 3 shows us that the fuzzy-search feature working as intended when there are multiple files. Each of the file's file header (the secret) can be constructed back from the tag and if the tag is wrong, the file will not be matched.

The experiment also show us how space-inefficient the AGFS architecture is. For each file, AGFS will create the mapping for corresponding to the file, even if the tag is exists in other files. This mapping table will effectively reduce the amount of effective disk space available to the user by 3 to 4 times the original size, which is undesirable.

The fuzzy-search on multiple files also bring performance hit, since AGFS need to scan through all file entry in the storage. This implies that a fuzzy-search is proportional not only to the number of words in the file, but also the number of files in the storage. The complexity for doing a fuzzy-search is $O(NWT)$, with N is the number of files, W is the number of words in the file, and T is the number of tags the fuzzy-search provides.

V. Conclusions

Cryptography has always been about securing informations and knowledge. In this paper, we see that cryptography can be beneficial in fields other than security. The possibility of a previously impossible feat that cryptography can achieve make the author hopes for more applications of cryptography that is beneficial in fields other than security.

The use of cryptography for filesystem addressing is a completely possible feat, in the sense that correctness of the functionality is achieved. It still lack of efficiency because there is no prior mature study to this application of cryptography. The author hope for further study, improving filesystem architecture to be more efficient but as effetive as AGFS.

Acknowledgment

I thank Allah S.W.T. for health and strength so I can finish this paper, Mr. Rinaldi Munir for his teaching in Cryptography course during this semester, and also my friends and fellow students in Informatics Institut Teknologi Bandung for their supports and assistance.

References

[1] Grosshans, Daniel. *File Systems: Design and Implementation*. Englewood Cliffs, NJ: Prentice-Hall, 1986. Print.

[2] Shamir, Adi *(1979), "How to share a secret", Communications of the ACM **22** (11): 612–613, doi:10.1145/359168.359176*

Originality Statement

I hereby declare that this paper is my own writing, not an adaptation, nor translation of other's paper and not plagiarism.

Bandung, 19th May 2016

[illegible]

Aufar Gilbran

13513015