Self Encrypting Data

David Irvine, email: david.irvine@maidsafe.net, LifeStuff: David

maidsafe.net limited (registered in Scotland Sc 297540)

September, 2010 ***Abstract*—This paper presents a system of encryption that requires no user intervention or passwords. The resultant data item then has to be saved or stored somewhere as in all methods. The encryption here is aimed at creating cipher-text (encrypted) objects that are extremely strong and closer to perfect in terms of reversibility, as opposed to known encryption ciphers available today. This paper focuses on symmetric encryption, but does not introduce a new cipher. Instead the paper describes a method of enhancing the use of this technology to produce highly secure data and, to do so in many situations and implementations.**

***Index Terms*—security, freedom, privacy, encryption**

CONTENTS

I. INTRODUCTION

ENCRYPTION has been a goal of man since before the times of the Romans, and Caesar Ciphers (simple replacement ciphers) through Enigma machine ciphers to modern day complex matrix manipulations are present in abundance in computer enhanced algorithms. This paper describes a way to use such algorithms in addition to direct encryption that clearly shows significant improvements in our use of encryption.

A. The Issues Addressed by this Paper

The issue with today's encryption of data is in just that; we encrypt data, as a whole. This reduces the potential set of possible inputs, i.e. if we are chasing somebody's bank balance, we may expect the output to be roughly the size of a bank statement, and guess what? It is! Furthermore, the security of a whole piece of data encrypted with a single algorithm depends upon just that single algorithm not getting broken. Almost all encryption ciphers appear to reduce in effectiveness as we understand the mathematics better and create more powerful computers. One may believe then, that the answer is to encrypt bits of files. However, this would require many passwords or algorithms; like putting more locks on a door to make it more secure, it gives people a headache to think that they may have to remember multiple passwords for each file. This is unlikely to be a successful manoeuvre.

B. Conventions Used

This paper does not require all the operations listed below, but lists these for example implementations, which are outlined later.

Hash = Hash function such as SHA, MD5, etc. We assume a cryptographically secure algorithm.

Symm = Symmetrical encryption such as AES, 3DES, etc.

PBKDF2 = Password-Based Key Derivation Function or similar

C. Symmetric Encryption

This paper will use AES as an example to cover all symmetric encryption algorithms (to some extent) and therefore will use a key and initialisation vector and plain-text input data. There will be no mention of MAC or similar additions to the algorithms in the hope that the reader would not attempt to implement a poorly stated or incorrect algorithm. The primary role in, interestingly, not encryption, it is to produce difficult to guess uncompress-able output. There may be alternative methods of producing difficult to guess uncompress-able output, these are not considered here.

Proposition 1. *Difficult to guess and uncompress-able output equates to random results based on random input data and* *random, unrelated algorithm inputs (plain text, key and iv in the case of modern symmetric cyphers).*

D. Cryptographically Secure Hash

Fact 2. *The ideal cryptographic hash function has four main or significant properties:*

1) it is easy (but not necessarily quick) to compute the hash value for any given message
2) it is infeasible to generate a message that has a given hash
3) it is infeasible to modify a message without changing the hash
4) it is infeasible to find two different messages with the same hash

Conjecture 3. *A cryptographically secure hash which is a one way function will create output that has a uniform distribution and can be computed in polynomial time. The output should be in fact random, although can be affected by size of input. Given a sufficiently large input the output will be random (within limits). The size of input required is dependent on the strength of the hash functions employed. In essence output can be considered evenly distributed and random. The limits of this randomness are not presented in this paper. It is assumed a sufficiently secure algorithm acting on a significantly large input will produce randomness that is acceptable for this conjecture.*

A hash function can be thought of as a digital fingerprint. Just as a fingerprint of a person is supposed to be unique, then a digital hash function is also supposedly unique. We have all heard of two people with identical fingerprints (but perhaps have never met any!) and in the digital world it can be possible to get two pieces of data with the same hash result. This is referred to as a collision and reduces the security of the hash algorithm. The more secure the algorithm, then the likelihood of a collision is reduced. It is very similar to taking more points of reference on an actual fingerprint to reduce collisions in that area of science also. This is an area where both systems share a similarity in the increasing complexity of measurement and recording of data points of reference.

In cryptographically secure hashing, the data is analysed and a fixed length key called the hash of the data is produced. Again similarly to human fingerprinting a hash cannot reveal data just as a fingerprint cannot reveal a person (i.e. you cannot recreate the person from the print) and you cannot recreate the data from the hash.

Early hash algorithms such as MD4, MD5 and even early SHA are considered broken, in the sense that they simply allow too many collisions to occur. Hence larger descriptors (keylengths) and more efficient algorithms are almost always required.[1]

II. Implementation

A. Overview

1) Split into several chunks (C$_n$).
2) Take hash of each chunk (H$_{c_n}$).
3) In case of AES or similar cypher, use [keysize] (C$_{n-1}$) as the key, use [next bytes iv size](C$_{n-1}$) as the IV. (for AES 0 to 32 == key and 32 to 48 == iv)
4) Create obfuscation chunk (OBFC$_n$) by concatenating the hashes of other chunks ([unused part of]C$_{n-1}$ C$_{n-2}$and C$_n$).
5) Run encryption cypher or similar reversible method on (C$_n$), to produce (C$_{random}$).
6) Now data is considered to be randomised and of the same length as input data.
7) OBFC$_n$ is also random output, but of a length less than the input data.
8) Now we take OBFC$_n$$(repeated)$ XOR C_{random} to produce our output data.
9) Rename each with the hash of the new content and save these hashes.

Definition 4. One Time Pad as defined by Shannon[2], [5] is regarded as the only cryptosystem with theoretically perfect secrecy. It posses the following 3 items that define it:

1) Pads cannot be reused.
2) For a Shannon implementation as opposed to earlier cyclic pads, the pad must be as long as the message to be encrypted. i.e. a pad must be non repeating. This is the enhancement that takes this system a step further than the Vernam cypher [3], [4]
3) The pad must contain only random data.

Fact 5. *In this paper on the NSA web site, the Vernam cypher appears as the* unbreakable cypher *for on line tty (teleprinter) encryption, although XOR in this case is called modulo 2 division (simply different wording for the same mathematical operation). This paper is found here http://www.nsa.gov/about/_files/ cryptologic_heritage/publications/misc/tsec_kw26.pdf The title of this paper is "Securing Record Communications: The TSEC/KW-26".*

Conjecture 6. *As the Shannon system suggests a one time use random pad that is longer than the data to be encrypted is required for a true one time pad. In this paper we have used a symmetric encryption cypher (AES as example, with CFB) to introduce what can be described as randomness to the data itself. If this is truly random then it's the perfect pad in it's own right. We have also created an Obfuscation pad, which almost creates a pad that is usable as a OTP, however it fails to answer 2 above, i.e. it repeats as it's shorted than the data to be encrypted.*

Proposition 7. *We propose given the above definition and conjecture that the data itself be considered the pad and the obfuscation chunk is now repeating data (which is allowed by the definition of the Shannon Pad), although this is rather large amount of repeating data, it is also repeating random data. We propose this be considered as a form of one time pad. Added to that we also propose the actions taken on the data ton include randomness as well as pad randomness may in fact take the whole concept of OTP, just a little further,*

[1] you can see where the problem exists as the logical conclusion is a key longer than any known piece of uncompress-able data, to ensure no collisions

making a guess significantly more difficult.

B. File Chunking

Definition 8. $f_c \equiv$ file content; $f_m \equiv$ file metadata; $fh \equiv H(f_c) \mathit{or} fh \equiv H(H(c_1) + H(C_2) + ...H(C_{n-1}))$.

1) Take the size of the file ($f.size()$) and calculate number n of chunks[2]
2) Create chunks of 256KB (settable) in length and hash these chunks.
3) Take hash of each chunk, hash these and create a structure, which we will refer to as a data map.

The chunks are created with fixed size to ensure the set required to recreate the file is as almost as large as the number of available chunks in any data store.This data map is mapped to file metadata through fh

C. Encryption Step

In the encryption stage, we require two separate non deterministic pieces of data, the encryption key (or password) and the Initialisation Vector (IV). To ensure all data encrypts to the same end result we determine the IV from what can be considered non deterministic data[3], that being the hash of one of the chunks.

Definition 9. Encrypt with key and IV is shown as $Enc_{[key][IV]}(data)$in the following example. It is assumed the key and the IV for chunk n are derived from separate portions of the hash of chunk $n - 1$. In the case of AES for instance the first 32bytes of this hash are the Key and the next 16 bytes may be presumed to be the IV. Therefore these items are selected from random data, although the randomness can be deterministic (if we can guess the output of an algorithm such as AES, by guessing the input parameters, i.e. brute force) on the case of a one way function such as a cryptographic hash (as discussed).

Example 10. $Enc_{[H(c_{n-1[first32bytes]})][H(c_{n-1[32-48bytes]})]}(c_{xn}) \equiv c_{xen}$

Therefore we have the final stage of the data represented as chunks of highly obfuscated chunks. We than take the hash of each chunk again $H(c_{xen})$ and rename each chunk with the hash of its content.

D. Obfuscation Step

In the obfuscation step, we pollute each chunk with data from other chunks.

Example 11. For c_n, create an identically-sized data chunk by repeatedly rehashing the hash of chunk n + 2 and appending the result, i.e. $H(c_{n-2}) + H(H(c_{n+2})) + H(H(H(c_{n+2}))) + ...$

This is called the XOR chunk n (c_{XORn}) and is unsurprisingly XORed ($\bigoplus$) with chunk n.[4]

Example 12. $c_{XOR1} \bigoplus c_1 \equiv c_{x1}$, $c_{XOR2} \bigoplus c_2 \equiv c_{x2}$, etc.

E. Data Map

In the previous sections, we described the process of self encrypting data. However, it did leave an important question unanswered. How do we reverse this process to retrieve the plain-text from the cipher-text chunks? The answer is data maps.

In the II-A steps 1, 3 & 7 we collected important data. This data alone is enough to reverse the encryption process and this is stored in a structure we refer to as a data map. This is described in the following table.

$fh = H(H(c_1) + H(C_2) + ...H(C_{n-1}))$[5]	
$H(c_1)$	$H(c_{xe1})$
$H(c_2)$	$H(c_{xe2})$
...	...
$H(c_n)$	$H(c_{xen})$

Data Map structure

With this structure the names of all the chunks are in the right hand column and all passwords and IV's (which are derived from the original chunk hashes) are stored in the left hand column. The file hash in the top row identifies the data element and acts as the unique key for this file. Reversing the process is now obvious.

1) Retrieve the chunks listed in right hand column.
2) Create each XOR chunk again.
3) Reverse the obfuscation stage.
4) Decrypt each result.
5) Concatenate the results.

This is the complete encrypt / decrypt process for each file.

F. Data Atlas or Recursive Data Maps

The data maps (dm) from multiple files can be concatenated into a new structure we call the Data Atlas (da). Therefore $dm_1 + dm_2 + \ldots dm_c \equiv da$. This data atlas is itself now a large piece of data and is fed into the self-encryption process once more. This produces a single data map and more chunks. These chunks are stored somewhere and the single remaining data map is the key to all the data.

III. Future works

To provide effectiveness the algorithms presented in this paper will require the addition of a secure mechanism to protect the data map of the data atlas. This will be furthered in an example of self authenticating system that will use this as entry to a system.

In addition the information should be looked after a network or system that is secured itself. This would require a very secure network or perhaps even the advancement of a self managing self healing network. This will be presented in a future paper on such a system.

[2] Number of chunks is a setting and depends on implementation, you may wish a max number of chunks, or maximum chunk size, this decision and code is left to the reader.

[3] This is an area of debate as to whether this is non deterministic data, in this case the argument is that the only way to determine the data is to have the original data in the first place, therefore there is no need to determine keys as it would be fruitless. This is somewhat of a philosophical debate and likely to be the topic of a few furled eyebrows over a few drams in a few bars for a few years to come.

[4] In this case we have selected XOR to represent a logical operation to obfuscate the data, this is not restrictive in any way and may be replaced by other obfuscation methods.

IV. Conclusions

This process allows for multiple data elements to be encrypted in a very powerful fashion. Indeed there may be some debate as to whether the encryption or obfuscation stages cannot be left out (well, at least one of them). It is decided this is not a bottleneck in such a system, as data can be processed at speeds in excess of current networking capabilities in many cases. This is open to further research for differing situations though.

An important issue here is that all data is encrypted using no user information or input. This means that if the container for all the chunks is a single container then duplicate files will produce the exact same chunks and the storage system can automatically remove duplicate information. It is estimated the savings in such a system would be greater than 95%.

Also interesting is the fact that the encryption may be seen as a "step too far"; nevertheless it does indicate that any break in an encryption cipher will not reveal any data to an attacker. This is a valuable and important point.

It is hoped the research in this field will continue and measures of number of chunks versus data map size, etc. would reveal interesting scope for optimisations and improvements.

Compression has been missed out from the steps in this paper and can be simply added to the process of hash/encryption of each chunk. This further improves efficiency, particularly with regard to improving data de-duplication results.

References

[1] David Irvine, maidsafe: A new networking paradigm, david.irvine@maidsafe.net

[2] Shannon, Claude (1949). "Communication Theory of Secrecy Systems". Bell System Technical Journal 28 (4): 656–715.

[3] Gilbert S. Vernam, "Cipher Printing Telegraph Systems For Secret Wire and Radio Telegraphic Communications", Journal of the IEEE, Vol 55, pp109–115 (1926).

[4] Gilbert S. Vernam, "Automatic Telegraph Switching System Plan 55-A", AIEE Transactions on Communication and Electronics, May 1958, p. 239. Also in Western Union Technical Review Vol 12 No 2, April 1958, p. 37.

[5] C.E. Shannon, "Communication Theory of Secrecy Systems," Bell System Technical Journal, Vol. 28, No. 4 (October 1949), pp. 656–715.

David Irvine is a Scottish Engineer and innovator who has spent the last 12 years researching ways to make computers function in a more efficient manner.

He is an Inventor listed on more than 20 patent submissions and was Designer of one of the World's largest private networks (Saudi Aramco, over $300M). He is an experienced Project Manager and has been involved in start up businesses since 1995 and has provided business consultancy to corporates and SMEs in many sectors.

He has presented technology at Google (Seattle), British Computer Society (Christmas Lecture) and many others.

He has spent many years as a lifeboat Helmsman and is a keen sailor when time permits.