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 focussed on creating cypher-text (encrypted) objects that are extremely strong and closer to perfect in terms of reversibility as opposed to known encryption cyphers available today. This paper focusses on symmetric encryption, but does not attempt to create a new cypher, instead focusses on enhancing modern 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

LIST OF TABLES

LIST OF ALGORITHMS

I. INTRODUCTION

ENCRYPTION has been a goal of man since before the times of the Romans and Ceaser cyphers (simple replacement cyphers) through Enigma machine cyphers to modern day complex matrix manipulations present in abundance in computer enhanced algorithms. This paper takes such algorithms and adds some features 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 immediately the potential set of possible inputs, i.e. if we are chasing somebodies bank balance, we may expect the output to be roughly the size of a bank statement and guess what? it is! In another convention or proposition, a whole piece of data encrypted with a single algorithm would only require the algorithm to be broken. Almost all encryption cyphers appear to be reduced in effectiveness as we understand the mathematics better and have more powerful computers, this is unlikely to reduce in the future. One may believe then the answer is encrypt bits of files, although this would require many passwords or algorithms, like putting more locks on a door to make it more secure, it does give people a headache to think 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 or MD5 etc.

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

PBKDF2 = Password-Based Key Derivation Function or similar

C. Symmetric encryption

This paper will use symmetric encryption as a term to cover all 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.

D. Cryptographically secure hash

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 (or two people having the same fingerprint) 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 is produced, this key is called the hash of the data. Again similarly with 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, otherwise it would be a miraculous compression scheme indeed.

Early hash algorithms are considered broken, such as md4, md5 and even early SHA schemes, this is a bad representation as they are not broken, they simply can allow too many collisions and larger descriptors (key lengths) and more efficient algorithms are almost always required.

Therefore a hash is merely a best attempt to distinguish a piece of data by a fixed length string representation of the contents of the data and of only that data[1], it is really as simple as that.

II. Implementation

A. Overview

1) Take file and get hash of this.
2) Split into several chunks of various sizes.
3) Take hash of each chunk.
4) Create another chunk by concatenating the hash of the file.
5) XOR (or an equivalent logical operation) the chunks together as in a one time pad.
6) Take the hash from (3) and use the hash of chunk +1 as the key, use the hash of chunk +2 as the IV (yes this will force each file to the exact same output).
7) Rename each with the hash of the new content and save these hashes.

For multiple files this is a recursive process which will in itself create a large file

B. File Chunking

Definition 1.
$f_c \equiv file\ content,\ f_m \equiv file\ metadata,\ hf \equiv H(f_c)$

1) Take hf and create a structure, which we will refer to as a data map.
2) Take the size of the file $(f.size())$ and calculate number of chunks n[2]
3) Create chunks by taking the first n bytes of hf and randomise the chunk sizes to remove the possibility of gathering a set of similar chunks and associating these with a piece of data. This process must allow chunks to be of varying sizes in a manner not calculable (this is why the hash is used). Many algorithms can be employed at this stage.

[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

[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.

The chunks are created with random size to ensure the set required to recreate the file is as large as the number of available chunks in any data store.

C. Obfuscation Step

Repeatedly rehash the file hash,

Example 2. for (std::str length=$H(hf)$; length.size() < file.size(); length+=$H(length)$).

This is the XOR ($\bigoplus$) chunk. The process is now a repeated XOR[3]operation.

Example 3. $(XORchunk \bigoplus c_1) \equiv c_{1x}, (c_1 \bigoplus c_2) \equiv c_{2x} \ldots (C_n \bigoplus c_1) \equiv c_{nx}$ (rotational operation)

D. 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[4], that being the hash of one of the chunks.

Definition 4. Encrypt with key and IV is shown as $Enc_{[key][IV]}(data)$in the following example. It is assumed the key size is the hash size and the IV is derived from the first portion of the hash of the appropriate chunk hash.

Example 5. $ENC_{[C_{n+1}][C_{n+2}]}(c_n) \equiv C_{xen}$

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 by the hash of it's content.

E. Data Map

In the previous sections we discovered the process of self encrypting data, however, it did leave several as of yet unanswered questions. How do we reverse this process to retrieve the plain-text form the cypher text chunks. The answer is data maps.

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

fh	
$H(c_1)$	$H(c_{xe1})$
$H(c_2)$	$H(c_{xe2})$
$\ldots$	$\ldots$
$H(c_n)$	$H(c_{xen})$

Data Map structure

[3]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 operations to enhance the operation.

[4]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.

With this structure the names of all the chunks are in the right hand column and all passwords and IV's 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 as described in right hand column.
2) Reverse the encryption stage.
3) Create a the XOR chunk again.
4) Reverse the obfuscation stage.
5) Concatenate the results.

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

F. Data Atlas or recursive data maps

Encrypting multiple files is now a repeated method of a single file. This can be carried out very easily by re-cursing though a directory structure, creating chunks and storing them somewhere and gathering together lots of Data Maps (dm). We can concatenate all these into a new piece of data 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 process once more. This produces a single data map and more chunks. These chunks are stored somewhere and the single remaining data atlas is the key to all the data. This is the data map of the data atlas and is a crucial piece of information which must be secured by encrypting it, storing it safely somewhere or some other mechanism of the readers choice. In the maidsafe network this is the encrypted content of the TMID packet as described in [1].

III. 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 (with the exception of the dm of the da) 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 to far', never the less it does indicate that any break in an encryption cypher 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 verses data map size etc. would reveal differing results for differing situations.

References

[1] David Irvine, maidsafe - a new network paradigm. david.irvine@maidsafe.net

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 / Project Manager 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.