IntelliSec – The 1st International Workshop on Intelligent Security Systems
11-24th November 2009, Bucharest, Romania

RAMP SECRET SHARING AND SECURE INFORMATION STORAGE

CSIRMAZ, Laszlo
Central European University, Renyi Institute, Budapest

Abstract: *In our age information is an important asset. We need systems in which costumers can securely store information, ensi\uring that is persists, contunuously available, and is kept confidental. We discuss how such a systems might work, what are the ingredients, and enlist some interesting unsolved problems.*
Key words: *Secret sharing, ramp scheme, information dispersal, distributed encryption*

1. INTRODUCTION

Information is an important asset. Companies, individuals might have tremendous problem when losing valuable data. Data recovery could cost a whole fortune. Thus storing copies of the most valuable data chunk in several remote places is an everyday solution. Commercial IT companies offer data storage services with high level guarantees. Using such services seems to be a good solution. Nevertheless, there are some basic requirements such and arrangements must meet:

1) Diversity: never rely on a single service. If you use a single storage server, you lose your data if that server crashes of becomes unavailable. Rather you should use a multitude of servers.

2) Security: never trust any third party. Never store your valuable data without encryption, and especially don't let any third party store it where you have no control over who sees the data.

3) Cost-effectiveness: minimize your cost by minimizing the amount of data you store. Arrange the data so that each external server stores the minimal possible amount of data.

We discuss how such a service can work, what are the ingredients, and how to combine them. We enlist some interesting problems as well. For an introduction for an existing system, see Wylie *et al* (2000).

2. BASIC INGREDIENTS

Distributing information and later recovering it from some of the share is the topic of **secret sharing**. It has a huge literature, see, e.g., Blakley (1979) .Shamir (1979), Krawczyk (1993). In the simplest case, discovered by Shamir (1979), the secret is split into n pieces so that the secret can be recovered form any k of them. In this case the secret is an element of a finite field, and the *shares* are the computed as the value of a secret random polynomial at certain public values. During the *reconstruction* phase the polynomial is recovered via interpolation from the available shares, and the secret is computed as the value of the polynomial at zero.

Such threshold schemes come very handy when using information storage. We split the information into n parts to be stored at n different companies. The data can be recovered from any k of the shares, thus even only k of the servers are available, the data still can be recovered.

Furthermore, the above threshold secret scheme has the following security advantage: even if k-1 of the servers collude and put together their shares, they have *no information on the secret*. This strong guarantee comes at a certain cost, namely all shares must be at least as long as the secret itself. In the threshold case, fortunately, this length is the minimal theoretically possible.

Using Shamir's threshold secret sharing scheme our first and second goal is achieved, but the third is not. If we want to recover the data from k shares, then theoretically the share can be as short as $1/k$ of the length of the data. For example, simply split the data into k pieces bitwise, namely the first chunk contains the first, k+1, 2k+2, etc bits, the second chunk the 2, k+2, 2k+2, etc, and the k-th chunk contains the k-th, 2k, 3k, etc bits. It is clear that each chunk contains 1/k of the original one, and the original data can be recovered from them.

In this way we have cost-effectiveness, but we have lost both diversity *and* security. It is even not clear how can we reach diversity: we want to recover the data from any k shares, not only from a particular k shares.

Fortunately Shamir's original idea works here as well. Suppose the data is not the value of the polynomial at zero, rather the polynomial itself. If the secret data is the k coefficients of a degree k-1 polynomial, then it can be recovered from arbitrary k values it takes by using Lagrange interpolation.

Thus we have both diversity and efficiency. What is still missing is security. Using the above method, each share, or even a combination of several shares reveal a lot of information about the secret. Is there anything we can do?

Using the ideas from Krawczyk (1993) before distributing the data we use a strong encryption and the encrypted data is distributed among the data storage servers. Even if all of them collude, they can recover the encrypted data only, which has no use without knowing the encryption key.

To recover our data we do need the secret key, thus it becomes a very valuable and important data, thus it should be stored securely at several places. However, as the key is at most several thousand bits long, opposed to the data, which could be several tera or petabytes, effectiveness is much less an issue for the key.

Summing up: the proposed secure information storage consists of the following parts:

1. a secure encryption system which using a secret key K can safely encrypt the whole data.
2. a *perfect* secret sharing scheme which distributes the secret key and leaks no information
3. a *ramp* secret sharing scheme which distributes the data, this scheme might leak out data as long as the encryption scheme is safe.

In the next section we look in more detail these ingredients and their interplay.

3. ENCRYPTION SYSTEM

To ensure maximal security, an encryption scheme should be used which allows *random access* to the data. Very probably not all data will be required at once, and modifying data at particular location will be necessary. Such a system could be, e.g., AES with a key which is derived from the master key K and the actual location of the block to be

encrypted/decrypted inside the whole data. Such method is used when encrypting the content of a hard drive. A particularly useful method to "tweak" a block cipher for this purpose was designed by Phillip Rogaway in 2003, see Rogaway (2004), and is called XEX mode. A variation, which uses two different keys rather than a single one was named XTS and was approved as the IEEE 1619 standard for cryptographic protection of data on block-oriented storage devices in December 2007.

The encryption works as follows. The $E_K(m)$ is a secure block cipher using secret key K. The data is split into chunks and each chunk will be accessed incrementally. In practice, the chunk size can vary from 16k to a few megabytes. Let N denote some physical address identifying a chunk, and let i be the sequence number of the next block (thus i=0 for the first block, i=2 for the next, etc.). Then first create a *chunk key* L as follows:

$$L = E_{K2}(N)$$

where *K2* is the so-called *secondary key*. This L has the same length as the block size of the block cipher. Using L new keys are generated for each block. For the first block this is the same as L, for each subsequent block the previous key is multiplied by a primitive element of the finite field of which L is a member. Denoting this primitive element by a, the *block* key of the i-th block is

$$\Delta = a^i L$$

Encrypting the content M of the i-th block is done as

$$C = E_{K1}(M+\Delta)+\Delta$$

where the addition is bitwise modulo 2, and *K1* is the primary encryption key. Generating the next block key from the previous one is a fast and efficient, and is negligible compared to the execution of the encription itself. In fact, it is a mere multiplication by a constant (predetermined) value over a finite field. This multiplication can even be speed up by choosing an appropriate base. In Rogaway (2004) paper it is shown that if E is resistent to chosen ciphertect attack (CCA-secure), then so is this scheme. Thus if we have confidence in the underlying block cipher, then this encryption scheme which makes random access possible within the stored data is secure as well.

4. SECRET SHARING SCHEMES

As discussed above, secret sharing scheme is used both for storing the master key and the data itself. In both cases the simplest, and the most popular arrangement is Shamir's threshold scheme: any k shares out of n determine the secret.

The *security* of the two schemes should be different. While the data - being encrypted - is not sensitive, the *key* is sensitive. Thus, we should use the strongest possibility, namely any k-1 of the shares should give no information on the secret value at all.

As indicated in the introduction, Shamir's secret sharing is based on polynomial interpolation. In the stronger case suppose that the secret to be distributed is an element of the finite field **F**. Label all units who will receive some share by different elements of the field, for simplicity we indicate these elements as small integers. To distribute the secret $s \in$ **F**, choose a random polynomial of degree k-1:

$$p(x)= a_{k-1} x^{k-1} + a_{k-2} x^{k-2} + \ldots + a_1 x + a_0$$

in a way that the value of the polynomial at 0 is the secret itself. Unit i will receive the share $p(i)$.

As any degree k-1 polynomial is determined uniquely by the k values it takes at k different places, thus $p(x)$ can be determined uniquely by the shares of k units. But can it be done efficiently?

The key observation is the so-called *Lagrange interpolation theorem.* Suppose the polynomial is given at places $b_1, b_2, \ldots, b_k$. Then it is easy to come up with a polynomial of degree $k-1$ which takes 1 at, say, b_1, and zero at all other places. As it is zero at $k-1$ places and it has degree $k-1$, it should be of the form

$$p_1(x) = c\,(x-b_2)\,(x-b_3)\ldots(x-b_k)$$

The constant c should be chosen so that $p_1(x)$ have value 1 at b_1 i.e. c is just the reciprocal value of

$$(b_1-b_2)\,(b_1-b_3)\ldots(b_1-b_k).$$

We can define the polynomials $p_2(x), \ldots p_k(x)$ similarly. Now the polynomial

$$p(x) = v_1 p_1(x) + v_2 p_2(x) + \ldots + v_k p_k(x)$$

takes the value v_1 at b_1, v_2 at b_2,, etc, and v_k at b_k., thus we have recovered the randomly selected polynomial. To get the secret, one has to replace 0 here to get

$$\text{secret} = v_1 p_1(0) + v_2 p_2(0) + \ldots + v_k p_k(0).$$

As can be seen, there is no need t compute the polynomials $p_i(x)$ only the value they take at zero. Knowing all the b_i values beforehand, these values can be precomputed. Recovering the secret is thus a simple linear combination of values in the field **F**.

Recovering the secret will be done (hopefully) rarely, and it is only as a safety device. Don't forget: your data is worth a single penny only if you know the secret key to recover the data. Storing the master key (or keys) by secret sharing makes key loss a negligible possibility. Secret sharing also ensures that to recover the key at least k shares are necessary, thus if at most $k-1$ of the external companies collude, they can get no information on the key. Choosing the value of k appropriately is a matter of trust which should be balanced between the possibility of data theft and data loss.

Once master key is available, we can start recovering our data. In this case – being the data encrypted – there is no need for high secrecy. In this case we can focus on *efficiency*.

Once again, we consider the scenario when we want to recover our data from any k of the external units. It is clear (but a strict proof can be quite involved) that we cannot do better than to store at least $1/k$ of the data at each unit. The question is: can we do it in a way so that we can recover the whole data from any k of them? The answer is *yes*, and the method used is the *ramp secret sharing* indicated in the title of this paper.

In a *ramp* sharing we do not require that unqualified sets should have no information on the secret value (which is a very important property when the encryption key is distributed), rather we focus on efficiency, i.e., store the smallest possible amount of material.

We shall use Shamir's original idea with a little twist. Namely, in this case the distributed value will be the *whole polynomial*, and not its value at a certain point. As we have seen above, the polynomial can be recovered from any k values it takes somewhere. The method is as follows.

Choose the finite field **F** to have size, say 2^{128} which means that each element of **F** is a 128-bit binary number. Take the next k elements of the data stream, and consider them as the coefficients of the polynomial

$$p(x)= a_{k-1} x^{k-1} + a_{k-2} x^{k-2} + \ldots + a_1 x + a_0$$

Compute the value of this polynomial at 1, 2, ..., $N-1$, and N, where N is the number of units we want to distribute the data among. Then send these values to the corresponding units. Observe, that for each block of k 128-bit number, we send one 128-bit number to each unit, thus each of them will receive only $1/k$ part of our data.

To recover the data we connect to k units with labels say $b_1, b_2, \ldots, b_k$. We can recover any block of length k. We request the data from which was generated from this particular block. Say, we receive the 128-bit numbers $v_1, v_2, \ldots, v_k$. Using these values we can recover the polynomial $p(x)$ as

$$p(x) = v_1 p_1(x) + v_2 p_2(x) + \ldots + v_k p_k(x)$$

and the coefficients of $p(x)$ give us the numbers in the requsted block.

Observe, again, that similarly the case above, we can precompute the polynomials $p_i(x)$ thus recovering the coefficients of $p(x)$ requires computing k linear combinations of the returned values.

5. IMPROVEMENTS

The method outlined in the above uses the specially tailored Shamir's secret sharing. There are other efficient secret sharing schemes which can be fit to a more refined requirement. This happens, e.g., when we have more storage servers available, and might weight them. Very probably they do not offer the same level of services, they have different prices, and have different reliability. Or simply we trust one service better than others. Thus we might need to store more data on one server than the other. How can we balance the distribution?

There is an extensive literature to that type of questions in case of *perfect secret sharing*, see, e.g., Csirmaz (2008) for a comprehensive overview of the topic. For *ramp schemes* we know very little. The main unsolved question can be formulated as follows:

Given n servers and we enlist those subsets which should recover the secret data. Also, each server has a price tag which tells the storage cost per unit data on that server. Find an algorithm which minimizes the total cost of distributed data storage.

As additional requirements, we can prescribe certain *untrusted* coalitions, i.e. coalitions of servers which should not be able to recover the secret.

It is clear, that if k servers should recover the secret, then one of them should store at least $1/k$ part of the secret. In Shamir's scheme this lower bound is attained, in fact every server stores just as much as $1/k$ part of the total data. If one server offers half price on storage, is there any way we could put more load on that particular server without giving it extra power as well?

5. CONCLUSION

We have outlined how secure distributed data storage should work. It uses data encryption and secret sharing. The latter one is used for two purposes: first to distribute the secret key used during encryption and decryption, second to distribute the encrypted data. While we have a good understanding how secret sharing works when the secret key is distributed, and unqualified coalitions should not gain information on the data, our knowledge is less vague for the other case. The simplest secret sharing, namely Shamir's threshold scheme works well and is theoretically the best possible, but it is not scalable, and cannot handle different prizing models.

6. REFERENCES

Blakley, G.R (1979), *Safeguarding cryptography keys*, Proc. NCC AFIPS;, pp 313-317

Csirmaz, L (2008) *Secret sharing schemes: Solved and unsolved problems Available from:* http://www.renyi.hu/~csirmaz/talk9-.pdf

Shamir, A (1979) *How to share a secret*, Comm of ACM, vol 22(1979) no 11, pp 612-613

Krawczyk, H (1993) *Secret sharing made short*, CRYPTO'93, LNCS 773 (1993) pp 136-146

Rogaway, P (2004) *Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC*, ASIACRYIPT '04, LNCS 3329 (2004) pp 16-31

Wylie, J.J, Bigrigg, M.W, Strunk, J.D, Ganger, G.R, Killccote, H, Khoisa, P.K(2000): S*urvivable Information Storage Systems*,Computer-Los Alamitos,