Secret Sharing and Splitting

Laurence Grant Brian Fleming

Notre Dame, IN - December 16, 2002

Abstract

From nuclear weapons to governments to class projects, the problem of mistrust in any working relationship requires a secure system to prevent access, power, or information from being compromised by a single member, or a small group of members. While our project does not claim unbreakable security for high-level government secrets, the concept of Secret Sharing or Secret Splitting would allow a feasible solution for small-scale situations where the information put in each parties' keys is the only way to gain access the secret key.

1 Keywords

applied cryptography, cryptology, XOR, secret sharing, secret splitting, key management

2 Introduction

In the situation that a large amount of power, or sensitive information, is in the hands of a group of people, there is the possibility (or even probability) that it is undesirable for any one member to wield direct access to the key necessary. The solution approached in our project was to create an n number of distinct keys, one for each party, where only a certain combination of the keys would allow access, ensuring that one, or even a coalition of, members would not be able to hijack access and compromise the data and its power.

3 Implementation

Our program is actually an implementation of two different methods of distributing a share of the secret key equally among several different parties. The first way, **Secret Splitting**, gives each person 1 piece of the key, and it requires all the pieces to reconstruct the secret key[SC96]. The second way, **Secret Sharing**, gives each person 1 piece of the key, but it only requires a certain number of the total pieces to reconstruct the secret key[SC96]. Each has its place in real world applications, and each can be properly implemented to best fit the situation.

3.1 Secret Splitting

The first mode which does the splitting of the secret key, is the secret splitting mode. This mode takes the number of pieces, X, which are required, and the key, K, to generate X equally important pieces. To generate X pieces of your key, you create $X-1$ random numbers. After $P_1, P_2, P_3, \ldots, P_{X-1}$, random numbers are created using a secure random number generator (this prevents someone from trying to duplicate the splitting procedure), you then operate on these pieces.

$$P_X = P_1 \oplus P_2 \oplus P_3 \oplus \ldots \oplus P_{X-1} \oplus K$$

Now to the users you give the P_X pieces, and you destroy K. While from this method it may seem as though the final piece is the only one with any value relating to the key itself, this is not the case. With the XOR function ($\oplus$) each piece is inherently important in the reconstruction of the key, if any bits in any of the pieces are changed, then the key is not recoverable. To reassemble the key, you do a very similar operation.

$$K = P_1 \oplus P_2 \oplus P_3 \oplus \ldots \oplus P_{X-1} \oplus P_X$$

Some argue that the length of these random numbers should be the same length as the key. This is to prevent two or more users from determining the size of the key by looking for similarities. But this result is the same as if the pieces weren't the same size as the key, so it provides no extra security.

3.2 Secret Sharing

The second mode of splitting in our program is known as **Secret Sharing**. This also breaks up the secret number, but now, there is a way to reconstruct

the number without all the pieces. An example of this in the real world might be something like nuclear launch device, which any three of 5 generals can launch. This way no single general who goes crazy can launch, but it does allow for a pair of generals to not have to be at the main launching base at all times. This is known as a *threshold scheme*, more specifically a $(m, n) - threshold\ scheme$. For each secret number you make N pieces, known as *shadows*, and it only requires M of these shadows to reassemble the secret number.

The specific version of this **Secret Sharing** which we are doing is known as the **LaGrange Interpolating Polynomial Scheme** which was originally presented by Adi Shamir[SH79]. You first choose a prime, P, which is larger than both the possible number of shadows, and the largest possible secret (in our program P is set at 1051). This P is also always made public. In our program we allow the user to create $(2, 2), (2, 3), \ldots, (2, 7)$ pairs. We generate a random number, A which is to be kept secret, and is discarded after the operation.

$$F(X) \equiv Ax + K \pmod{P}$$

You can pick any value for X which is less than $P - 1$, and you use the previous equation to generate N shadow pairs;

$$(X_1, F(X)_1), (X_2, F(X)_2), \ldots, (X_N, F(X)_N)$$

To reassemble this you only need two pairs of shadows, you then plug them into the above equation, and are left with two unknowns, A and K. If you have less than two equations, you can gain no extra information, because this only reduces the possible solution space to an infinite line of solutions. And any more than two shadows is redundant.

Due to the complexity of solving the matrix in the finite field $\mathbf{Z_P}$ which results from these operations, we only implemented a $(2, N)$ system, with $P = 1051$. The reduction of this operation to a linear function is outlined in the following equations. Assume you know the two pairs (B, C) and (D, E) with B and D being the X value of the equation $AX + M \equiv F(X)$, and C and E being the $F(X)$ values, so we must look for K, the secret key.

$$\begin{bmatrix} B & 1 \\ D & 1 \end{bmatrix} \begin{bmatrix} A \\ K \end{bmatrix} = \begin{bmatrix} C \\ E \end{bmatrix} \quad \text{Now: } R_2 = R_2 - R_1 \left(\frac{D}{B}\right)$$

$$\begin{bmatrix} B & 1 \\ 0 & 1 - \left(\frac{D}{B}\right) \end{bmatrix} \begin{bmatrix} A \\ K \end{bmatrix} = \begin{bmatrix} C \\ E - \left(\frac{CD}{B}\right) \end{bmatrix} \quad \text{Now: } R_2 = \frac{R_2}{1 - \left(\frac{D}{B}\right)}$$

$$\begin{bmatrix} B & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} A \\ K \end{bmatrix} = \begin{bmatrix} C \\ \frac{E-\frac{CD}{B}}{1-\frac{D}{B}} \end{bmatrix} \quad \text{Which means:}$$

$$K = \frac{E - \frac{CD}{B}}{1 - \frac{D}{B}}$$

This equation looks simple, but then one must remember we are working in a finite field, so multiplication, addition, and subtraction are fairly easy, but division requires computing the inverse of the denominator in the size of your prime (in our case, 1051). Luckily, because we have chosen our own B and D's we can precompute the inverses to save time while running the program. This reduces the number of inverses we have to find from 3 to 1.

To find a multiplicative inverse in a finite field is not an easy task. First we must Make sure the number, which we are trying to find an inverse does have an inverse in the field we are working. To check this we must make sure the gcd of the number and the field size is one. This is the reason we work in a prime field, to ensure all elements in the field have multiplicative inverses.

To find the multiplicative inverse of any number, n, in a prime field of size p, we can let $n^{-1} = n^{p-2}$. So now when we need two divide a number, a by p, we can reduce this by multiplying a by n^{-1}[ST97]. But to do this in the program it requires a loop with $p - 2$ operations. This is the major reason we precompute the inverses of our B's and D's. And as the size of our p would rise we would need to find a more efficient way of computing this inverse. There are ways which exist, but to implement them would require more work than time it saves when we are working in a field of size 1051.

Our program's limit of size of prime and a $(2, N)$ size system is by no means the limit of this idea. For example, to extend it to a $(5, N)$ domain, given the key, K, one w ould need to generate random numbers A, B, C, D.

$$F(X) \equiv Ax^4 + Bx^3 + Cx^2 + Dx + K \pmod{P}$$

To generate N shadows, you would just need to provide N different X's into this equation, and output N pairs of $(X, F(X))$. Then, to reassemble, you plug in the five $(X, F(X))$ pairs, and then you solve the 5 x 5 matrix for $A, B, C, D, and\ K$ with K being your secret key.

4 Possible Exploits

While the secret splitting algorithms we have implemented are impossible to break if all parties only have access to their own keys, if this trust is broken, at any stage of the process, then the level of security all but disappears. The following hypothetical example is one which shows how one party, with complete knowledge, can use information to cheat one person in particular, this could be extended to complete knowledge cheating everyone, or partial knowledge cheating some of the participants.

We begin by meeting our participants, Adam, Brent, Chet, David, and Eve. While none of these parties trust each other, and thus the need for the secret splitting algorithm, there exists an even higher level of animosity between Adam and Eve (something about the Apple Corporation). The code which they are breaking up is one which allows all parties to successfully receive the next set of information in a fight to win a government contract. The code has several variations, one which allows all parties to move on, others which allow certain parties to continue, and many which end all parties' involvement in the contract. If Eve knows that the number given to them by the government allows all parties to continue, including Adam, and she knows which code removes Adam from the race, and Eve has access to all the codes, then Eve has power over the whole system. All Eve has to do is to put all the other people's pieces the other people got, excluding her own, into the "generator", and to replace her number, the secret number which will end Adam's chance at the contract. The number which these 5 codes generate Eve can now use as her code, and when all parties re-convene, she can successfully exact revenge upon Adam.

Another possible way to break our program's Secret Sharing protocol lies in the weakness of the random generator function. Currently the generator is seeded with a combination of the key and the number of pieces. If one were to have access to a few of the pieces, and the program, he would be able to generate all the possible combinations of keys and number of pieces in a **brute force attack** . Because you are not actually working with the system for which the key was designed, there are no repercussions involved with testing all possible options. This weakness could be removed by improving both the seeding of the random number generator and the number generator itself.

5 Summary

We feel that our program met all of our original goals. It successfully splits the key into 2 - 7 pieces, and the user can choose whether to require all pieces to reassemble, or just 2 pieces are needed. It operates in a GUI environment which works equally well for Windows or UNIX environments. The speed of the program is appropriate for the difficulty of operations which it is performing.

6 The Future of SCB

While SCB did meet its original goals of implementing both secret sharing and secret splitting, there is almost unlimited room for growth. The most obvious extension would be to allow larger number to be split into as many pieces as the user required. Also we could allow for any (M, N) combinations (for up to a reasonable size M).

Another possible option would be to allow a third party to enter the secret key remotely and only display the pieces to the group members. Also this could be set up so the third party enters the key, and N e-mail addresses, and have the program automatically send the pieces to the users.

A third extension would be to add a file encryption module right into the program itself. Then the group would give the program a file location, the number of pieces needed, and the type of splitting. Then the program would encrypt the file with a secret key which no one would ever see. The program would next take this key, split it properly, and display the pieces to the group members. Then to reassemble, the group would gather all the pieces, and the encrypted file (which they could all safely keep a copy of themselves, because no one party could unlock it), and let the program output the original file.

7 Thanks

We'd like to first thank our families. We'd also like to thank Professor Izaguirre, Bertrand Meyer, Seymour Cray, Fr. Sam Peters, Fr. Sorin and our fellow classmates who were with us through the long nights in Fitzy.

8 Credits

8.1 Brian "George" Fleming

Brian is a junior in computer science at the University of Notre Dame and hails from the great state of Maryland. His interests involve computers because of their purity, but he appreciates such things as the aesthetic beauty inherent in cleaning lawn mowers. Regardless of future plans, he firmly believes that rock and roll is, in fact, here to stay.

8.2 Laurence "LT" Grant

LT is a junior math major from the small town of Newark Valley, NY who now is a Screaming Otter of Sorin College. He hopes to go on and get his Ph.D. in math, probably specializing in Cryptography. In his spare time, LT is a video game and sport junkie.

References

[SC96] Bruce Schneier, *Applied Cryptography*, New York, NY, John Wiley & Sons, Inc. 1996.

[SH79] Adi Shamir, *How to Share a Secret*, Communications of the ACM **24** (1979), n. 11, (1979), 612-613.

[ST97] Saul Stahl, *Introductory Modern Algebra*, New York, NY, John Wiley & Sons, Inc., 1997.