Analysis of secret functions with application to computer cryptography

by INGEMAR INGEMARSSON
Linköping University
Linköping, Sweden

ABSTRACT

In computer cryptography we cannot avoid that data and the corresponding encrypted data can be read by an outside observer. The information contained in these observations may be used to decrypt parts of encrypted data or ultimately to identify the key in the cryptographic transformation. In this paper we have analyzed this situation using the concepts of information theory. The result shows that in most cases it is theoretically possible for an outside observer to identify the key after very few observations. As this must be avoided we have to rely on computational complexity in the process of deriving the key. This is achieved by using one-way functions which are practically impossible to invert.

INTRODUCTION

Computer cryptography differs from communication cryptography in two respects: (i) A particular set of data is used more than once and by several users. (ii) Data are processed by the computer. These two differences impose restrictions on the type of cryptologic transformations suitable for use in an electronic data processing (EDP) system. In this paper we will focus our attention on the problems caused by property (i) above.

Communication cryptography was analyzed by Shannon 1949. His model includes an information source (a stochastic process) with known statistics. The information from the source is encrypted and then observed by an outside observer. The goal of the observer is to derive the original information and/or identify the encryption transformation. Our approach is somewhat different. When several users have access to the same data, the encryption transformation for the particular set of data is preferably fixed, at least for some time. In an EDP system it is also realistic to assume that a set of data and the corresponding cryptogram is known to an outside observer. The observer may use this knowledge to facilitate decryption of other cryptograms, encrypted with the same transformation. He may also ultimately be able to identify the encryption transformation. We want to investigate his chances to work in these directions. A related problem, from the legitimate user's point of view, is: How do we avoid that an outside observer gets the ability to decrypt stored cryptograms? Our approach originates in information theory. The encryption or decryption transformation (or algorithm) may be regarded as a set of known functions of the input variable. The function is chosen by the key which is unknown. From the beginning an outside observer does not know which function is actually realized. To him each function has the same probability.

When he observes the input and the corresponding output he receives some information about the actual function. The number of possible functions is decreased. To what extent does this help the observer if he wants to estimate the output corresponding to another input? We may formulate the question more precisely this way: Before the observation the uncertainty, measured as entropy, is $H_0$ bits. If no information as how to estimate other outputs is conveyed by the first observation, then the uncertainty remains unchanged. The entropy connected with the second observation is still $H_0$. It may sound as a good design objective to keep the uncertainty (the entropy) unchanged after several observations. Unfortunately this is limited by one of the main results of this paper.

$$\sum_{i=0}^{N} H_i = \log M$$

Here $H_i$ is the entropy after $i$ observations and $M$ is the number of encryption or decryption transformations, i.e., the number of keys. As is seen from equation (1) the requirement that the entropy should remain constant must be limited to a finite (and perhaps low!) number of observations. After that the entropy is zero, i.e., the observer knows exactly which function is actually chosen, i.e., he knows the key!

If we want to avoid this fallacy the sequence of entropies

$$\left\{ H_i \right\}_{i=0}^{\infty}$$

125

From the collection of the Computer History Museum (www.computerhistory.org)
must decrease. This means that the difficulty of the observer to estimate outputs for given inputs is decreasing with the number of observations! Obviously we have to compromise to obtain reasonable protection of the key and reasonable low chance to estimate the outputs.

However, when typical figures are put in the equations the result is most unsatisfying for the designer. If the model is accurate, it is too easy to break the system. The way out of this dilemma is of course to build a system for which this model cannot be used. The main point here is that we have not taken into account the computational problem involved in estimating the output for a given input starting from the knowledge from several observations. Hence the theory shows that we have to rely on computational complexity when designing computer cryptographic systems. The complexity requirement is preferably formulated in terms of one-way functions. Such a function is easy to compute but its inverse is not computable in a reasonable amount of time.

SECRET FUNCTIONS

We want to implement a secret function, i.e., a function which is not completely known to an outside observer. Such is the case for example in cryptology and in access control systems. The input variable x is supposed to be discrete and takes only a limited number of values. The output variable y is a function of x.

\[ y = f_\alpha(x) \]  
\[ \alpha = 1, \ldots, M \]  

Starting from scratch, an outside observer does not know \( \alpha \), but knows the set of functions \( f_\alpha \):  
\[ \{f_\alpha(x)\}_{\alpha=1}^M \]  

The problem is now: How does the knowledge of a number of pairs \((x,y)\), satisfying (2), affect the uncertainty about the parameter \( \alpha \)? The range of possible \( \alpha \) obviously cannot increase if the observer gets to know one more pair \((x,y)\). The problem of the designer of the secret function seems to be maximizing the remaining number of possible \( \alpha \). We will see that this is not good advice. H leads to low uncertainty about \( y \), given \( x \). A better formulation of the problem is therefore: How does the knowledge of a number of pairs \((x,y)\), satisfying (2), affect the uncertainty of \( y \) given \( x \)?

STRUCTURE OF THE SYSTEM

We may divide the system into a known part, containing the functions (4) and an unknown part containing the parameter \( \alpha \).

S may be visualized as a store or memory, capable to store the M possible values of \( \alpha \). Hence the capacity, \( C \), of the store must be:

\[ C = \log_2 M \text{ bits} \]  

Figure 1 also reflects the structure of the implemented system. The block containing the function \( f(x) \) is supposed to be known, while \( S \) is a secure memory. \( \alpha \) is regarded as the outcome of stochastic variable \( A \) with

\[ P[A=\alpha]=1/M \text{ for } 1 \leq \alpha \leq M \]  

The output is then a stochastic variable \( Y \). \( n \) pairs \((x,y)\), satisfying (2), are known to the observer. The uncertainty in \( Y \) for a given \( x \) is measured as the conditional entropy:

\[ H_n(x) = H(Y|Y_1, \ldots, Y_n) = -\sum_{y} \sum_{Y_1, \ldots, Y_n} P(y,Y_1, \ldots, Y_n) \log_2 P(Y,Y_1, \ldots, Y_n) \]  

Note that this entropy is a function of \( x \) and \( y \), where \( \alpha \) is known for the output variable \( y \). If \( Y \) is conditionally independent on previous observations \( Y_1, \ldots, Y_n \) then

\[ H_n(x) = H(x) \]  

where \( q \) is the number of possible values for the output variable \( y \). If \( Y \) is conditionally independent on previous observations \( Y_1, \ldots, Y_n \) then

\[ H_n(x) = H_n(x) \text{ as } P(y|Y_1, \ldots, Y_n) = P(y) \]  

Note that \( H \) is still a function of \( x \). If we sum \( H_n \) over \( n \) we obtain the following result:

\[ \sum_{n=0}^{N} H_n = -\sum_{y} \sum_{Y_1, \ldots, Y_n} P(y,Y_1, \ldots, Y_n) \log_2 P(y,Y_1, \ldots, Y_n) \]  

If we make \( N \) large enough only one or none of the \( M \) functions will pass through a given set of points \((x,y),(x_1,y_1), \ldots, (x_M,y_M)\). Thus equation (10) reduces to:

\[ \lim_{N \to \infty} \sum_{n=0}^{N} H_n = -\sum_{y} \frac{1}{M} \log_2 \frac{1}{M} = -\log_2 \frac{1}{M} = \log M \]
where we have used the probability in equation (6). Finally we combine equations (5) and (11) to:

\[ \sum_{n=0}^{\infty} H_n = C \tag{12} \]

In words: The sum of the entropies at each observation in a sequence of observations equals the capacity (in bits) of the key space. In practice C is the number of bits in the binary key.

**Example**

In a computer cryptographic system the input, output and key are binary numbers.

In the algorithm proposed by NBS as federal standard, for example, k is 64 bits. (Actually lower because of redundancy in the key.) The output is also a 64-bit word.

Thus the maximum entropy (max \( H_n \) according to (8)) is 64 bits. If \( H_n \) really is 64 bits, then from (12) \( H_0, H_1, \ldots \) is zero. Hence all information about the transformation is given in the first observation. The future outcomes are perfectly predictable.

Now suppose that \( H_n \) is lower than 64 bits. Suppose that we want the system to withstand 8 (eight!) observations before the outside observer can identify the key. Then from (12):

\[ \sum_{n=0}^{7} H_n = 64 \text{ bits} \]

and thus the average of \( H_0, \ldots, H_8 \) is 8 bits. With this low entropy it takes only 256 trials (at most!) to make a correct guess of the output for a given input.

The situation is indeed disappointing, from the theoretical point of view!

**ONE-WAY FUNCTIONS**

We have as yet made no indication as how to compute the estimates of the output or the key from our observations. As we have seen from the above example it may be possible to predict the output and to identify the key after just a few observations. This must be avoided, of course. The only way to stop any effort in this direction is to make it practically impossible to perform the calculations which are theoretically possible.

We refer to Figure 2. The function:

\[ y = f(a, x) \]

is calculated by the encryption unit and must be easy to perform. Also:

\[ x = f^{-1}(a, y) \]

has to be performed by the decryption unit. On the other hand we have to design the system so that the function

\[ a = g(x, y) \]

is practically impossible to compute. This is an example of a one-way function. The algorithm proposed by NBS does indeed have this property. To my knowledge it has successfully resisted every attempt to compute the key, given the input and output.

**REFERENCES**
