The Compressed Storage in Time Memory Tradeoff Attack

Lijun Zhang[1][2]
Science and Technology on Communication Security Laboratory, Chengdu, 610041,China
E-mail: achong731@sohu.com

Qingbing Ji
Science and Technology on Communication Security Laboratory, Chengdu, 610041,China
E-mail: jqbdxy@163.com

Fei Yu
Science and Technology on Communication Security Laboratory, Chengdu, 610041,China
E-mail: feiyu1980@gmail.com

Yu Zhou
Science and Technology on Communication Security Laboratory, Chengdu, 610041,China
E-mail: yuzhou1982@gmail.com

Time memory tradeoff (TMTO) attack is an important and practical cryptanalytic technique for inverting random functions. Actually, it is almost the only sufficiently effective method to recover keys or passwords of an encryption system in a time constrained attack environment. In this paper, we investigate the table storage problem in the pre-computation phase that is an essential consideration in the TMTO attack. After exploiting the structure and lookup operation of the table, we propose several compression ways to ensure the table elements storage more efficient, which brings a lot of benefit for the online search phase in the attack. Furthermore, we provide explicit experiment results to demonstrate the efficiency improvement derived from the application of these compression strategies.

CENet2015
12-13 September 2015
Shanghai, China

[1]Speaker
[2]Correspongding Author

1. Introduction

Exhaustive search is the universal technique to solve cryptanalytic problems by searching the expected key in the candidate space. However, this kind of brute force attack is usually unacceptable in practice on the aspects of consumed time or memory since every new attack instance requires to restart the process of encryption or lookup in an enormous dictionary containing all pairs of key and ciphertext. The first breakthrough is the time memory tradeoff (TMTO) method proposed by Hellman in 1980 [1], which is applied to crack DES. Later, it was improved by Rivest [2] and Oechslin [3] with the idea of distinguished points and rainbow table respectively. Recently, Hong and Moon [4] presented an efficiency comparison of all known cryptanalytic tradeoff algorithms. It is worth mentioning that TMTO attack is very practical by making use of pre-computation tables which need to be constructed only once. For example, this attack is implemented in many platforms such as FPGA and GPU to retrieve key of cipher LILI-128 [5] and login passwords of Unix [6] or Windows system [3]. Compared with the traditional lookup attack, the significant advance of TMTO is the replacement of pairs (key, ciphertext) in the table by chains of keys or passwords which not only saves a lot of storage but also enables search process much faster.

Table storage compression is an inevitable consideration in the TMTO attack. Hong [4] gave some suggestion to compress the storage and Barkan [7] derived the theoretical bound for storage size while later Saran [8] improved the result by choosing the parameters elaborately. More recently, Broek [9] studied the storage compression when attacking stream ciphers. However, these improvements for table compression are not comprehensive and incomplete.

In this paper, we present a completed analysis of table storage in TMTO attack including various strategies for point choice, chain construction, table format and so on. These strategies can be generally useful for any kind of tradeoffs such as classic tradeoff, distinguished points and rainbow table. Besides, we also give concrete experiment results to demonstrate the application of these storage optimizations in a real attack.

2. Table Construction and Storage in TMTO Attack

The idea of TMTO is to save the memory at the cost of cryptanalytic time. The TMTO attack contains two phases. The first phase called pre-computation is to construct attacking table and the second phase is online searching when the ciphertext is known. Take the key recovery attack as an example, we briefly explain the principle of this attack. Given the fixed plaintext P_0 and corresponding ciphertext C_0, we expect to find the key k such that $C_0=E_k(P_0)$. In the pre-computation phase, choose m starting points from key space K, i.e., $SP_1, \dots, SP_m$ *in* K and iteratively apply the encryption E and reduction function R: C--->K which generates a chain of keys. Denote $f(\mathrm{k})=R(E_k(P_0))$, we finally have the following chains:

$$SP_i= k_{i1}\text{-->}k_{i2}\text{-->}k_{i3}\text{-->}k_{i4}\text{-->}\dots\text{-->}k_{it}=EP_i,\ (i=1,\ \dots,\ m) \qquad (2.1)$$

where k_{ij} is the j-th iteration of f on SP_i, EP_i is the endpoint of the i-th chain and t is the chain length. Note that only pairs (SP_i, EP_i) are stored in a table and every chain can be regenerated from SP_i, and f. The function f is the same for this table in the original tradeoff (called Hellman table). Later, in order to increase the success rate, different f is adopted in columns of the table (called rainbow table). For simplicity, in this paper we do not distinguish these different kinds of table since we concentrate on the storage of table.

In the attack phase, given a target ciphertext C', we try to search the key k in the table which is corresponding to C'. It works as follows: first compute $Y=R(C')$ and compare Y with EP_i, if there is a match, then regenerate that chain from starting point and the right key k is expected at position just before endpoint in that matched chain. Otherwise, compute Y<---f(Y) recursively until we find a match. Naturally, if no match is found in this process eventually, the attack will fail.

From the principle of TMTO, we could see that the table is pivotal in the attack. Since the table consists of a large number of pairs (SP_i, EP_i), it is important to perform a good data structure. In fact, the excellent storage is not only helpful for saving the memory space but also benefits search efficiency in the online attack phase. The storage is related to choice of starting and ending points, the construction way of chains and the data organization of entire table. In the following sections, we will study and present optimized strategies for these aspects of storage.

3. The Compression Strategy of Points and Chains

The table is constructed from starting point and the points of starting and ending are basic component of a tradeoff table, so it is necessary to research on the point choice for optimized storage.

3.1 The Compression of Points

The only requirement of starting point is that it must come from key space. Theoretically, it can be chosen randomly when a chain is constructed. If the the size of key space is N, the stored bits of a starting point is $logN$. However, we could make use of consecutive numbers to represent the starting points which require less storage. For a table of m chains (the value of m can be derived from expected success rate by a formula [3], the starting point is number from 1 to m. Therefore the storage is reduced from $logN$ to $logm$.

For binary key recovery, it is natural to express the candidate keys in consecutive numbers and the above strategy can be applied conveniently. However, for the case of password recovery, since the password is usually the combination of printable characters, it requires to encode these combinations to consecutive numbers. For example, all the passwords of numerics (0, 1, 2, ..., 9) with length 1 to 6 can be encoded by consecutive number from 1 to $10^1+10^2+...+10^6=1111110$. Contrarily, given a number n in the range [1, 1111110], it is not difficult to translate n back to a real password. This can be done by first deciding the length of password and then deriving it according to order of array [0, 1, 2, ..., 9].

Endpoint is the other part of stored pairs. Generally, after many iterations of function *f,* the endpoints are random values in the key space. So the storage of an endpoint is also $logN$ bits. Rivest suggested the distinguished point method to reduce the endpoint storage [2]. The distinguished point is a point satisfying a certain criterion, for example its first 8 bits are zeroes. A chain is stopped only when a distinguished point is reached. Now the endpoint storage is less as these zeroes of 8 bits can be discarded.

In practice, after all pairs of (SP_i, EP_i) are written in a table when the chains are finished, these pairs will be sorted according to the order of endpoints. Then we could use another technique to reduce the storage size of endpoints. As we know, the consecutive numbers have the same several prefix bytes which can be removed as an index, so only the rest part of the number needs to be stored. This strategy is explained in Fig.1. In this figure, the same prefix (3 bits) of adjacent numbers is extracted and the whole storage can be considerably reduced when the value of chain number m is large.

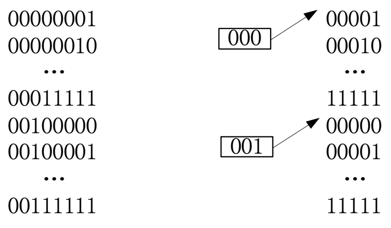

Figure 1: compressed storage of consecutive numbers

3.2 The Compression of Chains

The chain construction and regeneration is the most time-consuming operation in TMTO attack. The main shortcoming of distinguished point is that the chain length is not constant, so it is uncertain when to stop the iteration of function f and increase the attack time. But the rainbow table does not have the advantage of less storage of endpoints as distinguished point. Here we recommend the combination technique of distinguished and rainbow table. The operation of this kind of chain construction is as follows: for table chains of fixed length t, choose t iteration functions f_i, i=1, ..., t. Apply f_1 on the starting point SP_j until the first distinguished point DT_1 is found, then apply f_1 on point DT_1 until the second distinguished point DT_2 is found and so forth. A chain is ended at t-th distinguished point DT_t and the pair (SP_j, DT_t) is stored in the table.

$$SP_j= k_{j1}\text{-->}k_{j2}\text{-->}DT_1\text{-->}...\text{-->}DT_2\text{-->}...\text{-->}DT_t=EP_j. \quad (3.1)$$

It is worth mentioning that in the chain construction, we need to set a threshold of iteration times for function f_i since the chain length is uncertain when a distinguished point is found. Once the threshold value is reached, we simply abandon this chain and start with a new starting point.

4. The Compression Strategy of Entire Table

The strategies described so far aim at reducing the storage size of table. However, the final strategy of compressed storage we will discuss is attempting to improve the search efficiency. The basic idea of facilitating faster lookups in a table is to sort the endpoints of the stored point pairs. But when we identify the target point with stored endpoints, its efficiency is based on the chain number m and the comparison become slower as the m is larger.

A better alternative is to adopt hash table which is a widely known technique of data structure. If we want to record many words in a table, the idea of hash table is to determine the position of every word in a table in advance by a position function G. For a word w, its storage position is $G(w)$. In practice, we usually set an array of one dimension to store these words, and the value $G(w)$ is exactly the corresponding subscript of array. It is advantageous to search a given word w' in a hash table since the existence of w' can be identified by finding out whether the record of position $G(w')$ in the table is empty or not.

For TMTO attack, we would store a lot of point pairs (SP_i, EP_i) in the pre-computation table. As is mentioned before, in the phase of online searching, the output of iteration function f will be compared with the stored endpoints EP_i and if they match, that chain will be regenerated from the corresponding starting point SP_i. So it is favorable to store these point pairs in order of endpoints in a hash table. Meanwhile, it should be careful to choose a proper function G because of the large amount of stored point pairs.

5. Experiment Results of Compressed Storage

Now we present a real TMTO attack against A5/1 encryption in GSM telephony system. In this example, the application of storage compression strategy will be demonstrated detailedly.

GSM is the most popular mobile communication system in the world the users of which exceeds 4 billion. It employs A5 series encryption to protect user's privacy and provides service of phone call, text message and internet surfing. A5 series encryption contains three kinds of algorithms, i.e., A5/1, A5/2 and A5/3. A5/1 is the original algorithm used in Europe and America while A5/2 is the deliberately weakened version for global export. A5/3 is developed publicly and stronger but not yet being used in practice. Both A5/1 and A5/2 are designed secretly but later were discovered by reverse engineering [10]. A5/2 was proved to extremely weak and can be broken instantly [11]. Therefore many researchers are interested in cryptanalysis of A5/1. There were several academic breaks attempts [12, 13], but they did not work practically until the rainbow table attack was published [14]. We will explain how to apply compression strategy of table storage in this real attack.

5.1 The Brief Description of A5/1 Algorithm

A5/1 is a stream cipher based on a combination of three linear feedback shift registers with irregular clocking. The Feedback polynomials of three registers are $x^{19}+x^{18}+x^{17}+x^{14}+1$, $x^{22}+x^{21}+1$ and $x^{23}+x^{22}+x^{21}+x^{8}+1$ respectively. These registers are clocked in a stop/go fashion using a majority rule associated with a clocking bit (8, 10, 10). At each cycle, the clocking bit of all three registers is examined and the majority bit is determined. A register is clocked if the clocking bit agrees with the majority bit as is in Fig.2.

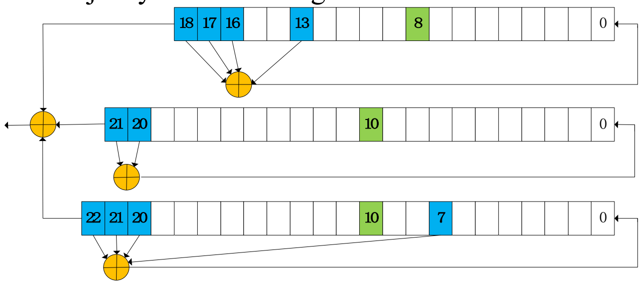

Figure 2: structure of A5/1 stream cipher

A5/1 cipher involves two parameters: 64 bits session key *Kc* and 22 bits frame number *Fn* where *Kc* is the key we want to recover in the attack.

5.2 Table Parameters and Original Storage

The aim of A5/1 cryptanalysis is to recover the secret session key *Kc*. On the assumption that success rate of recovery is over 90%, the parameters of TMTO attack for A5/1 cipher are as follows:

- the number of pre-computation tables is r=40.
- the number of chains in every table is m=8.662*10^9.
- the number of iteration function f is t=8.
- the number of zero bits set in distinguished point is d=12.

Under this configuration, since both starting point and endpoint are 64 bits and every point pairs need 264/8=16 bytes, the original storage of total 40 tables is:

$$S_{orig}=m*r*16\ B=5163\ GB=5.04\ TB \tag{5.1}$$

5.3 Compressed Storage and Comparison

Now we improve the storage of A5/1 tables by applying the compression strategy introduced before.

(1) **Point compression**. As is explained before, we can use consecutive numbers from 1 to m to represent the starting points instead of random values of 64 bits. However, we can make a further improvement by using a pseudorandom generator g whose output is a 64-bit number but input is much shorter and this input is stored as starting point. In our case of A5/1, the length of input is 34 bits. So we save 30 bits for every starting point at a little computation cost of g. For endpoints, the storage size can be reduced to 52 bits since the chosen 12-bit zeros in distinguished point. Now every point pairs only need 86 bits, i.e., 11 bytes and the total storage of total 40 tables is

$$S_{point}=m*r*11\ B=3550\ GB=3.46\ TB \tag{5.2}$$

(2) **Table compression**. There are several steps to finish table compression. Firstly, when the pre-computation of table is finished, it is sorted in the order of endpoints and pairs with the same endpoints are discarded. After this, the chain number is reduced to almost m'=6.010^9. Now the sorted storage of total 40 tables is

$$S_{sort}=m'*r*11\ B=2457\ GB=2.40\ TB \tag{5.3}$$

If these 40 tables are stored as files in disk, then the data access time is a bottleneck in the practical attack. Therefore we should make the storage structure more efficient for searching phase, first we can use the idea of consecutive number mentioned before. The point pairs will be directly written to disk blocks instead of being stored in binary files. In each block, only the first pair is stored entirely and endpoints in other pairs is stored by the balance with the first endpoint since neighboring values just differ in several bits. For example, let the pairs in a block are (SP_i, EP_i), i=1, ..., n, then from i=2, the pair (SP_i, EP_i), is stored as (SP_i, F_i) where F_i= EP_i -EP_1 . For fast searching, we can apply the hash table strategy. After the data of pairs are written completely in a block, every endpoint of first pair in this block is copied to another file called index file *F*. This means we choose the position function *G* as $G((SP_i, EP_i))=EP_i$. Now, given an endpoint *EP* to be matched, its block position is quickly identified by loading the index file *F* and endpoints can be compared only in one block, which accelerates the matching process greatly. This kind of table is called index table and can be regarded as a degenerated form of hash table. The compression of table storage is described in Fig.3.

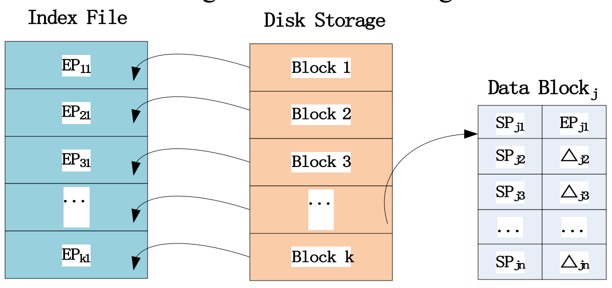

Figure 3: compression storage of entire table

In this way, a complete point pair can be encoded with an average length of around 54 bits and the total size of table is

$$S_{final}=m'*r*54/8\ B=1508\ GB=1.47\ TB \tag{5.4}$$

The compression effect of every strategy applied is compared in table 1.

Compression Level	Original	Point	Chain	Table
Storage Size (TB)	S_{orig}(5.04)	S_{point}(3.46)	S_{sort}(2.40)	S_{final} (1.47)
Compression Rate	1	68.6%	47.6%	29.2%

Table 1: experiment results of compression effect

6. Conclusion

In this paper, we studied the compressed storage of pre-computation table, which is an essential problem in the TMTO attack. We proposed several compression strategies of different levels to make both the table storage and searching process more efficient. At last, we gave a concrete example in a real cryptanalytic attack to demonstrate the effect of compressed table storage. These compression strategies are generally helpful for TMTO attack and can be more effective as the table size is larger.

References

[1] M. Hellman. *A cryptanalytic time-memory trade off* [J]. IEEE Transactions on Information Theory, IT-26(4):401–406, 1980.

[2] D. Denning. *Cryptography and Data Security* [M], pp. 98-100. Addison-Wesley, , Massachusetts, USA, 1982, Boston.

[3] P. Oechslin. *Making a faster cryptanalytic time-memory trade-off* [C]. CRYPTO 2003, LNCS 2729, pp. 617-630, Springer-Verlag, Berlin , 2003.

[4] J. Hong, S. Moon, *A comparison of cryptanalytic tradeoff algorithms* [J]. To appear in J. Cryptology. Published online on July 2012, available at http://dx.doi.org/10.1007/s00145-012-9128-3.

[5] M. O. Saarinen. *A time-memory tradeoff attack against LILI-128* [C]. In Fast Software Encryption, vol. 2365, p. 231-236, Leuven, Belgium, February 2001.

[6] N. Mentens, L. Batina, BartPreneel and I. Verbauwhede,*Time-Memory Trade-Off Attack on FPGA Platforms: UNIX Password Cracking* [C], LNCS 3985, pp.323–334, Springer-Verlag, Berlin , 2006.

[7] E. Barkan, E. Biham, A. Shamir, *Rigorous bounds on cryptanalytic time/memory tradeoffs* [C]. In Advances in Cryptology—CRYPTO 2006, LNCS 4117, pp. 1-21, Springer-Verlag, Berlin , 2006.

[8] N. Saran, A. Doganaksoy, *Choosing parameters to achieve a higher success rate for Hellman time memory trade off attack* [C]. In 2009 International Conference on Availability, Reliability and Security, (IEEE, 2009), pp. 504–509, Fukuoka, 2009.

[9] F. Broek, E. Poll. *A Comparison of Time-Memory Trade-Off Attacks on Stream Ciphers* [C]. AFRICACRYPT 2013, LNCS 7918, Springer-Verlag, pp. 406-423, Berlin, 2013.

[10] M. Briceno, I. Goldberg, D. Wagner, *A pedagogical implementation of the GSM A5/1 and A5/2 "voice privacy" encryption algorithms*, http://cryptome.org/gsm-a512.htm, 2001.

[11] B. Elad, B. Eli, K. Nathan, *Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication* [J], Journal of Cryptology 21(3): 392-429, 2010.

[12] A. Biryukov, A. Shamir, and D. Wagner, *Real time cryptanalysis of A5/1 on a PC* [C]. Proccedings of Eighth International Workshop Fast Software Encryption, pp.1-18, New York, 2001.

[13] S. Kumar, C. Paar, J. Pelzl, *Breaking ciphers with COPACOBANA a cost-optimized parallel code breaker* [C]. LNCS 4249, pp.101-118, Springer-Verlag, Berlin, 2006.

[14] K. Nohl, *Attacking phone privacy. BlackHat2010,* USA, 2010, available at http://www.blackhat.com/ html/ bh-us-10/bh-us-10-briefings.html