Time-Memory Tradeoffs

François-Xavier Standaert & Jean-Jacques Quisquater

UCL Crypto Group
Laboratoire de Microélectronique
Université catholique de Louvain
Place du Levant, 3, B-1348 Louvain-La-Neuve, Belgium
fstandae;jjq@uclouvain.be

Many searching problems allow time-memory tradeoffs. That is, if there are K possible solutions to search over, the time-memory tradeoff allows the solution to be found with high probability, in T operations (time) with M words of memory, provided the time-memory product $T \times M$ is larger than K. Cryptanalytic attacks based on exhaustive key search are the typical context where time-memory tradeoffs are applicable.

Due to large key sizes, exhaustive key search usually needs unrealistic computing powers and corresponds to a situation where $T = K$ and $M = 1$. However, if the same attack has to be carried out numerous times, it may be possible to execute the exhaustive search in advance and store all the results in a memory. Once this precomputation is done, the attack could be performed almost instantaneously, although in practice, the method is not realistic because of the huge amount of memory needed: $T = 1$, $M = K$. The aim of a time-memory tradeoff is to mount an attack that has a lower online processing complexity than exhaustive key search and lower memory complexity than a table lookup, neglecting the precomputations (hence, it only makes sense if the attack has to be performed multiple times). The method can be used to invert any one-way function and was originally presented by Hellman in [3].

1 The original method

Let $E_K(\mathrm{X}) : 2^n \times 2^k \rightarrow 2^n$ denote an encryption function of a n-bit plaintext X under a k-bit secret key K. The time-memory tradeoff method needs to define a function g that maps ciphertexts to keys: $g : 2^n \rightarrow 2^k$. If $n > k$, g it is a simple reduction function that drops some bits from the ciphertexts (e.g. in the DES, $n = 64$, $k = 56$). If $n < k$, g adds some constant bits. Then we define

$$f(K) = g(E_K(P)), \tag{1}$$

where P is a fixed chosen plaintext. Computing $f(K)$ is almost as simple as encrypting, but computing K from $f(K)$ is equivalent to cryptanalysis. The time-memory tradeoff method is composed of a precomputation task and an online attack that we describe as follows.

Precomputation task: The cryptanalyst first chooses m different start points: SP_1, SP_2,..., SP_m from the key space. Then he computes encryption chains where $X_{i,0} = SP_i$ and $X_{i,j+1} = f(X_{i,j})$, for $0 \leq j < t$:

$$
\begin{aligned}
&X_{1,0} \xrightarrow{f} X_{1,1} \xrightarrow{f} X_{1,2} \xrightarrow{f} .... \xrightarrow{f} X_{1,t} \\
&X_{2,0} \xrightarrow{f} X_{2,1} \xrightarrow{f} X_{2,2} \xrightarrow{f} .... \xrightarrow{f} X_{2,t} \\
&X_{3,0} \xrightarrow{f} X_{3,1} \xrightarrow{f} X_{3,2} \xrightarrow{f} .... \xrightarrow{f} X_{3,t} \\
&\qquad\qquad\qquad\qquad .... \\
&X_{m,0} \xrightarrow{f} X_{m,1} \xrightarrow{f} X_{m,2} \xrightarrow{f} .... \xrightarrow{f} X_{m,t}
\end{aligned} \tag{2}
$$

To reduce the memory requirements, the cryptanalyst only stores start and end points ($SP_i = X_{i,0}$, $EP_i = X_{i,t}$) and sorts the $\{SP_i, EP_i\}_{i=1}^{m}$ on the end points. The sorted table is stored as the result of this precomputation.

Online attack: Now we assume that someone has chosen a key K and the cryptanalyst intercepts or is provided with $C = E_K(P)$. Then he can apply the function g to obtain $Y = g(C) = f(K)$ and follow the algorithm:

Algorithm 1 Online attack

1. If $Y = EP_i$, then either $K = X_{i,t-1}$ or EP_i has more than one inverse image. We refer to this latter event as a false alarm. If $Y = EP_i$, the cryptanalyst therefore computes $X_{i,t-1}$, by reconstructing the chain from the start points, and checks if it is the key, for example by seeing if it deciphers C into P.
2. If Y is not an end point or a false alarm occurred, the cryptanalyst computes $Y = f(Y)$ and restarts step 1.

Remark that the cryptanalyst needs to access the table lookup every time a new Y is computed. If all $m \times t$ elements of the table (removing the first column that cannot be reached) were different, the probability of success PS would be $\frac{m \times t}{2^k}$. The actual probability of success depends on how the precomputed chains cover the key space. Unfortunately, there is a chance that chains starting at different keys collide and merge. The larger a table, the higher the probability that a new chain merges with a previous one. Each merge reduces the number of distinct keys that are actually covered by the table. If f is a random function, then the probability of success is bounded by:

$$PS_{table} \geq \frac{1}{N} \sum_{i=1}^{m} \sum_{j=0}^{t-1} (1 - \frac{it}{N})^{j+1}. \tag{3}$$

Equation 3 indicates that, for a fixed value of N, there is not much to be gained by increasing m or t beyond the point at which $mt^2 = N$. To obtain a high probability of success, a more efficient method is to generate multiple tables using a different function g for each table. The probability of success with r tables is:

$$PS_{tot} \geq 1 - (1 - PS_{table})^r. \tag{4}$$

Chains of different tables can collide, but not merge since the function g is different for every table.

2 Distinguished points and rainbow tables

The idea of using distinguished points (DPs) in time-memory tradeoffs is due to Rivest in [4]. If $\{0,1\}^k$ is the key space, a DP property of order d is usually defined as an easily checked property that holds for 2^{k-d} different elements of $\{0,1\}^k$, e.g. having d bits of the key equal to zero. In a time-memory tradeoff using DPs, the start and end points of the precomputed chains fulfill a DP property. As a consequence, the chains have variable length but detectable extreme points. This greatly reduces the number of table lookups during the online attack from t to 1.

A remarkable property of the DP method is that merges can be easily detected and, therefore, can possibly be rejected during the precomputation in order to build perfect tables [5]. The major drawback of DPs is that they introduce variable chain lengths and they are more difficult to analyze [6]. DP methods can also be used to detect collisions (e.g. of hash function) as suggested in [8, 9].

An alternative solution to reduce the number of table lookups is to use the rainbow tables presented in [7]. That is, to use a different function g for each point in a chain:

$$\begin{array}{c} X_{0,0} \xrightarrow{f_1} X_{0,1} \xrightarrow{f_2} X_{0,2} \xrightarrow{f_3} \ldots\ldots. \xrightarrow{f_t} X_{0,t} \\ X_{1,0} \xrightarrow{f_1} X_{1,1} \xrightarrow{f_2} X_{1,2} \xrightarrow{f_3} \ldots\ldots. \xrightarrow{f_t} X_{1,t} \\ X_{2,0} \xrightarrow{f_1} X_{2,1} \xrightarrow{f_2} X_{2,2} \xrightarrow{f_3} \ldots\ldots. \xrightarrow{f_t} X_{2,t} \\ \ldots\ldots. \\ X_{m,0} \xrightarrow{f_1} X_{m,1} \xrightarrow{f_2} X_{m,2} \xrightarrow{f_3} \ldots\ldots. \xrightarrow{f_t} X_{m,t} \end{array} \quad (5)$$

Two rainbow chains can only merge if they collide at the same position. Other collisions do not trigger a merge. As a consequence, rainbow tables are an elegant alternative to perform a time-memory tradeoff. As further reading, [1] provides a careful analysis of different cryptanalytic time-memory tradeoffs and discusses the technique of checkpoints that can be used to improve the detection of false alarms. Another analysis of cryptanalytic time-memory tradeoffs is provided in [2].

References

1. G. Avoine, P. Junod, P. Oeschlin, *Characterization and Improvement of Time-Memory Trade-Off Based on Perfect Tables*, ACM Transactions on Information and System Security, vol 11, num 4, July 2008.
2. E. Barkan, E. Biham, A. Shamir, *Rigorous Bounds on Cryptanalytic Time/Memory Tradeoffs*, in the proceedings of CRYPTO 2006, LNCS, vol 4117, pp 1-21.
3. M. Hellman, *A Cryptanalytic Time-Memory Tradeoff*, IEEE Transactions on Information Theory, vol 26, pp 401-406, 1980.
4. D. Denning, *Cryptography and Data Security*, pp 100, Addison-Wesley, 1982.
5. J. Borst, *Block Ciphers: Design, Analysis and Side-Channel Analysis*, Phd Thesis, Departement of Electrical Engineering, Katholieke Universiteit Leuven, 2001.
6. F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, J.-D. Legat, *A Time-Memory Tradeoff Using Distinguished Points: New Analysis and FPGA Results*, in the proceedings of CHES 2002, LNCS, vol 2523, pp 593-609.

7. P. Oechslin, *Making a faster Cryptanalytic Time-Memory Trade-Off*, in the proceedings of CRYPTO 2003, LNCS, vol 2729, pp 617-630.
8. J.J. Quisquater, J.P. Delescaille, *How easy is collision search? Application to DES*, in the proceedings of EUROCRYPT'89, LNCS, vol 434, pp. 429-434.
9. P.C. van Oorschot, M.J. Wiener, *Parallel collision search with cryptanalytic applications*, Journal of Cryptology, 12(1), pp 1-28, Winter 1999.