TIME-MEMORY TRADE-OFF USING DISTINGUISHED POINTS

BÜLENT YILMAZ

July 05, 2007

1 INTRODUCTION

Consider the following situation. Suppose, you know that in some future time, you will be given a problem chosen from a fixed public problem set, the answer to which belongs to a fixed public answer set. The problems in the problem set cannot be solved directly, but given any element of the answer set, it is easy to identify which problem it is an answer to.

There are two extreme ways one can deal with this situation. When you are given the problem, you could try every possible choice until you find the correct answer. This would be time consuming. Another method would be to do this time consuming operation before the problem is given, at your leisure, storing the answer-problem pairs in a table. You will be able produce the correct answer just by looking in the table, when the specific target problem is given, but this is at the cost of pre-computation time and memory for storing the table.

In 1980 Martin Hellman described a cryptanalytic time-memory trade-off which reduces the time of cryptanalysis by using precalculated data stored in memory. This technique was improved by Rivest before 1982 with the introduction of distinguished points which drastically reduces the number of memory lookups during cryptanalysis.

The time-memory trade-off is a probabilistic method. Success is not guaranteed and the success rate depends on the time and memory allocated for cryptanalysis.

This project explains the main aspects of TMTO using Distinguished Points and an implementation of it onto a reduced form of SHA-1.

2 ONE-WAY FUNCTIONS

A one-way function is a function that is easy to compute but hard to invert. Easy to compute means that some algorithm can compute the function in polynomial time. Hard to invert means that no probabilistic polynomial-time algorithm can compute a pre-image of $f(x)$ with better than negligible probability, when x is chosen at random. A trapdoor function is a one-way function that can be easily inverted only if one knows a secret piece of information, known as the trapdoor.

By inversion of a one way function, we mean a process which finds a pre-image of any point in the range of $f(x)$, provided that it exists. And in some

situations, the point has more than one pre-image. Then, we have to take the problem of inversion in two categories.

Let $f(x)$ be a function from A to B.

- Problem Type 1. Given $f(x_0) = y$, $x_0 \in A$, $y \in B$, find $x \in A$ such that $f(x) = y$.
- Problem Type 2. Given $f(x_0) = y$, $x_0 \in A$, $y \in B$, find x_0.

Definition: A function graph of a function $f : A \longrightarrow A$ is a directed graph whose nodes are the elements of X and whose edges are the ordered pairs $(x, f(x))$ for all $x \in X$.

Let f and X be as given in the definition. If we start from a point $x_0 \in X$ and apply f iteratively, we get the following sequence $\{x_0, x_1 = f(x_0), x_2 = f(x_1), \ldots, x_n = f(x_{n-1})\}$. Since the set X is finite, after some iterations, we will encounter a point which already appears, causing a cycle.

3 TMTO

3.1 Hellman's Method

The most basic pre-computation technique is to compute and store the hashes of all the passwords in the keyspace ordered by hash, so that cryptanalysis is almost instantaneous. However, this requires storage equal to the keyspace size. Hellman showed how to decrease storage requirements at the expense of attack time in the general context of cryptanalyzing a cipher.

Given a fixed plaintext P, define a mapping f on the keyspace K as $f(k) = R(E_k(P))$ where E is the *encryption function* and R is a *reduction function* which maps ciphertexts to keys. By iterated applications of f, we create a chain of keys. The crucial observation is that by storing only the first and last elements of a chain, we can determine if the key corresponding to a given ciphertext belongs to that chain (and also find the key) in time $O(t)$ where t is the length of the chain.

Creating a chain works as follows. Given a starting point k_0, we compute $k_1 = f(k_0), k_2 = f(k_1), \ldots, k_t = f(k_{t-1})$. The keys k_0 and k_t are stored, while the rest are thrown away. When given a ciphertext C to cryptanalyze, we recover the key by computing $k = R(C)$ and $k_i = f^{i-1}(k)$ for $i = 1, 2, \ldots$. Observe that if the key k belongs to the chain, i.e., $k = k_i$ for some i, then

$f^{t-i}(k) = k_t$. Thus, after at most t applications of f, we can determine whether or not the chain contains k. Further $i-1$ applications of f suffice to compute k_{i-1} from k_0. Since k_{i-1} satisfies the property that $C = E_{k_{i-1}}(P)$, it is the key we are looking for.

This does not work with certainty because different chains may merge. Therefore, we need to use multiple tables with a different reduction function for each table, with many chains in each table. The storage requirement as well as the cryptanalysis time for this algorithm are $O(|K|^{2/3})$, where K is the keyspace.

	Brute Force Attack	Lookup Table
Advantages	No or negligible memory One time, no precomputation	No or negligible time reusability
Disadvantages	Needs enormous amounts of time	Needs enormous amount of storage space precomputation

3.2 Distinguished Points

Rivest suggested an improvement which greatly speeds up the practical performance of the algorithm by decreasing the number of memory accesses during cryptanalysis. This is done using *distinguished points*, which are keys with some special property (such as the first 10 bits being zero). The advantage of requiring that the endpoints of a chain be distinguished points is that during cryptanalysis, a key must be looked up in memory only if it has the distinguished property.

Definition: Let $K \in \{0,1\}^k$ and $d \in \{1,2,3,...,k-1\}$. Then K is a distinguished point (DP) of order d if the DP-property defined beforehand holds for K. Note that using this definition of distinguished point, we do not need to store the fixed bits and reduce the memory requirements of the tradeoff.

The algorithm proposed requires to choose a DP-property of order d and a maximum chain length t. We precompute r tables by choosing r different reduction functions. For each reduction function m different start points (which are distinguished) will be randomly chosen. For each start point a chain will be computed until a DP is encountered or until the chain length is $t+1$. Only start points iterating to a DP in less than t iterations will be stored with the corresponding chain length, the others will be discarded.

Moreover, if the same DP is an end point for different chains, then only the chain of maximal length will be stored. This involves a lower memory complexity than Hellman's tradeoff.

3.3 TMTO Implementation to Reduced SHA-1

The attack is implemented on SHA-1 as an example of the TMTO by using distinguished points. The domain of inputs is 32 letters, the Turkish alphabet and {q,w,x}. The input length is 4, 5 and 6 characters. So, the domain size is $N = 32^4 = 2^{20}$, $N = 32^5 = 2^{25}$, $N = 32^6 = 2^{30}$ respectively. The TMTO method is implemented with classical Hellman method.

	m	t	k	P
20-Bit	128	128	64	2^{20}
	64	128	128	2^{20}
	256	128	32	2^{20}
25-Bit	256	256	512	2^{25}
	256	128	1024	2^{25}
	256	1024	128	2^{25}

I have chosen a Distinguished Point property. Since there are 2^{20} possible input values, I mapped each vector to a 4-character input value starting from 0 to $2^{20}-1$. The constructed inputs are evaluated in the SHA-1 algorithm and the reduction function takes the first 20 bits which enters the next iteration as domain value until the result provides the DP-property. I store the starting point, the end point and the length of the chain as triples in my tables which are sorted by end points.

The offline computation was implemented on a single computer with CPU speed of 2.79 GHz and memory capacity of 1GB.

I have chosen a Distinguished Point property. Since there are 2^{25} possible input values, I mapped each vector to a 5-character input value starting from 0 to $2^{25}-1$. The constructed inputs are evaluated in the SHA-1 algorithm and the reduction function takes the first 25 bits which enters the next iteration as domain value until the result provides the DP-property. I store the starting point, the end point and the length of the chain as triples in my tables which are sorted by end points.

The offline computation was implemented on a single computer with CPU speed of 2.79 GHz and memory capacity of 1GB.

In the online phase 1000 random input values are chosen and their hash values are calculated. The hash values are given as input to the online search algorithm. The computations are carried out on a single computer with CPU speed of 2.79 GHz and memory capacity of 1GB.

	m	t	k	Success Rate	Search Time	False Alarms
20-Bit	128	128	64	$46,3$	$55ms$	108
	64	128	128	$47,1$	$58ms$	95
	256	128	32	$45,7$	$59ms$	110
25-Bit	256	256	512	$43,1$	$67ms$	121
	256	128	1024	$42,6$	$64ms$	125
	256	1024	128	$41,9$	$65ms$	117

Offline Time: Since we have to make a choice for DP-property and calculate an approximation for the average chain length (t), the smaller the average chain length, the less time consumed during offline time.

Memory and Search Time: For mk large we need more memory and since we do not access to the memory if the result is not a distinguished point the online search time is less.

Success Rate: For the the smaller values of k, the success rate is small.

False Alarm: For large m and k values there probability of false alarms decreases.

Compared with exhaustive search, we gain much time in Time-Memory Trade-Off. The only problem is to increase the success rate.

For smaller length of keys it can be used. However for large length of keys it would be hard to store and search so much data.

References

[1] J.Borst, B.Preneel and J.Vandewalle, *A Time-Memory Tradeoff using Distinguished Points,*Technical report ESAT-COSIC Report 98-1, Departement of Electrical Engineering, Katholieke Universiteit Leuven, 1998. ,

[2] M.Hellman, *A Cryptanalytic Time-Memory Tradeoff,* IEEE transactions on Information Theory, Vol 26, 1980, pp.401-406.

[3] Ç.Çalık *How to invert one-way functions: Time-Memory Trade Off Method*, M.Sc. Thesis, 2007