# HOW TO INVERT ONE-WAY FUNCTIONS: TIME-MEMORY TRADE-OFF METHOD

ÇAĞDAŞ ÇALIK

JANUARY 2007

# HOW TO INVERT ONE-WAY FUNCTIONS: TIME-MEMORY TRADE-OFF METHOD

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF APPLIED MATHEMATICS

OF

THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÇAĞDAŞ ÇALIK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF CRYPTOGRAPHY

JANUARY 2007

Approval of the Graduate School of Applied Mathematics

Prof. Dr. Ersan AKYILDIZ
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Rüyal ERGÜL
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ali DOĞANAKSOY
Supervisor

Examining Committee Members

Prof. Dr. Ersan AKYILDIZ

Assoc. Prof. Dr. Ali DOĞANAKSOY

Assist. Prof. Dr. Ali Aydın SELÇUK

Assist. Prof. Dr. Ahmet Emrah ÇAKÇAK

Dr. Muhiddin UĞUZ

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

| | | |
|---|---|---|
| Name, Last name | : | Çağdaş ÇALIK |
| Signature | : | |

# Abstract

HOW TO INVERT ONE-WAY FUNCTIONS:
TIME-MEMORY TRADE-OFF METHOD

ÇALIK, Çağdaş
M.Sc., Department of Cryptography
Supervisor: Assoc. Prof. Dr. Ali DOĞANAKSOY

January 2007, 48 pages

Security of various encryption schemes, authentication mechanisms and other cryptographic protocols depend on the hardness of inverting one-way functions which they are based on. Time-Memory Trade-off (TMTO) method, proposed by Hellman [11], is a generic method to invert one-way functions, by enabling a trade-off to be made between the memory and the time required to find an inverse, at the expense of a precomputation effort. In this thesis, an analysis of the TMTO method is made and the application of the method to symmetric-key cryptosystems and hash functions is presented. A new asymptotic expression for the coverage of a single Hellman table which helps to approximate the success probability of the method is introduced. As an application, the method is applied to SHA-1 hash algorithm and the results are presented.

Keywords: Cryptography, One-Way Functions, Time-Memory Trade-off, Cryptanalysis

# Öz

## TEK-YÖNLÜ FONKSİYONLAR NASIL ÇEVRİLİR: ZAMAN-HAFIZA ÖDÜNLEŞİMİ

ÇALIK, Çağdaş
Yüksek Lisans, Kriptografi Bölümü
Tez Yöneticisi: Doç. Dr. Ali DOĞANAKSOY

Ocak 2007, 48 sayfa

Birçok şifreleme yönteminin, kimlik doğrulama algoritmalarının ve kriptografik protokollerin güvenliği tek yönlü fonksiyonların çevrilmesinin zorluğuna dayanır. Hellman tarafından önerilen [11] zaman-hafıza ödünleşimi (TMTO) tek yönlü fonksiyonların çevrimi için genel bir metotdur. Bu tezde, TMTO metodunun analizi yapılmış ve metodun simetrik anahtarlı sistemler ve özetleme fonksiyonlarına uygulanması gösterilmiştir. Metodun başarı oranını hesaplamaya yardımcı olan, bir Hellman tablosunun kapsama oranını veren yeni bir asimptotik ifade tanıtılmıştır. Bir uygulama olarak TMTO metodu SHA-1 özetleme algoritmasına uygulanmıştır.

Anahtar Kelimeler: Kriptografi, Tek yönlü fonksiyonlar, Zaman-Hafıza ödünleşimi, Kriptanaliz

To my mother with love

# ACKNOWLEDGMENTS

I would like to thank all the academic and administrative staff and my friends at the Institute of Applied Mathematics for creating a wonderful atmosphere for study. I would especially like to express my gratitude to my advisor Assoc. Prof. Dr. Ali Doğanaksoy for believing in me and giving me an opportunity. Without his guidance and motivation, this thesis and most of my other studies wouldn't have been completed successfully. I would also like to thank the following people:

- Meltem Sönmez Turan for her never ending 'new ideas' and the never ending smile on her face, always trying to keep me in good spirits, for her all support,
- Nurdan Saran for working with me since the beginning of the thesis and for her encouragement,
- Dr. Muhiddin Uğuz, Elif Saygı, Zülfükar Saygı and Fatih Sulak for working with us,
- Assist. Prof. Dr. Cem Özdoğan at Çankaya University for allowing us to make our computations on his parallel computing system,
- Ahmet Yafa for his present which made my work a lot easier,
- Pınar Tankut for reviewing the text,

Lastly, I would like to thank my family for their great patience and sacrifice.

# Table of Contents


Plagiarism . . . . . . . . . . iii

Abstract . . . . . . . . . . iv

Öz . . . . . . . . . . v

Acknowledgments . . . . . . . . . . vii

Table of Contents . . . . . . . . . . vii

List of Tables . . . . . . . . . . ix

List of Figures . . . . . . . . . . x

CHAPTER

1 Introduction . . . . . . . . . . 1

2 One-Way Functions . . . . . . . . . . 4

2.1 One-Way Functions . . . . . . . . . . 4

2.2 The Inversion Problem . . . . . . . . . . 6

3 Random Mappings . . . . . . . . . . 7

3.1 Random Mapping Statistics . . . . . . . . . . 7

3.2 Graph Representation of Functions . . . . . . . . . . . . . . . . . 10

4 Time-Memory Trade-off Method . . . . . . . . . . . . . . . . . 14

4.1 Exhaustive Search vs. Lookup Table . . . . . . . . . . . . . . . 14

4.2 Hellman's Method . . . . . . . . . . . . . . . . . . . . . . . . 16

4.3 Distinguished Points . . . . . . . . . . . . . . . . . . . . . . . 30

4.4 Rainbow Tables . . . . . . . . . . . . . . . . . . . . . . . . . 31

5 Applications . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

5.1 Block Ciphers . . . . . . . . . . . . . . . . . . . . . . . . . . 34

5.2 Stream Ciphers . . . . . . . . . . . . . . . . . . . . . . . . . 36

5.3 Hash Functions . . . . . . . . . . . . . . . . . . . . . . . . . 38

5.4 An Application of TMTO to SHA-1 . . . . . . . . . . . . . . . . 38

6 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43

References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46

# List of Tables


3.1 Probability of having a distinct collection for various multiples of $\sqrt{M}$. . . . . . 10

4.1 Comparison of exhaustive search and table lookup methods. . . 16

4.2 Comparison of the coverage formulas with empirical values for a single table of size $m \times t$ on a domain of size $2^{24}$. . . . . . . . . 26

4.3 The relation between $m$, coverage, time and memory complexities for a fixed total complexity value of $C = 2^{85.33}$ . . . . . . . . . . 28

5.1 TMTO parameters for the SHA-1 application. . . . . . . . . . . 39

5.2 Store requirements for the SHA-1 application. . . . . . . . . . . 41

5.3 Online phase results for the SHA-1 application. . . . . . . . . . 41

5.4 Comparison of the application results with Exhaustive Search and Table Lookup methods. . . . . . . . . . . . . . . . . . . . . . 42

# List of Figures


3.1 Graphical representation of an iteration operation . . . . . . . . 11

3.2 A rendering of the giant component of DES due to Quisquater and Delescaille. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

4.1 Construction of a chain . . . . . . . . . . . . . . . . . . . . . . . . . 17

4.2 Hellman's Matrix . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

4.3 Construction of a chain under the reduction function $r$ . . . . . 20

4.4 Comparison of Hellman tables and Rainbow tables . . . . . . . . 32

5.1 Generation of a chain in block ciphers . . . . . . . . . . . . . . . . 35

5.2 SHA-1 application, one-way and reduction function. . . . . . . . 39

# Chapter 1

# Introduction

One-way functions exist in almost all areas of cryptography. Securities of encryption schemes, authentication mechanisms and various other cryptographic protocols depend on the hardness of inverting one-way functions which they are based on. Roughly speaking, a one-way function is a function $f : X \to Y$ such that for an $x \in X$ it is 'easy' to compute the image $y = f(x)$, but for any $y \in Y$ it is 'hard' to find a preimage $x$ satisfying $y = f(x)$. Here 'easy' and 'hard' are considered in the context of complexity theory and they informally correspond to being computationally feasible and infeasible, respectively.

By inversion of a one-way function, we mean a process which finds a preimage of any point in $Y$, if there exists any. An obvious method to find a preimage of a given point is trying all possible inputs and checking whether they yield the given value. This method, which is known as exhaustive search, requires expectedly $|X|/2$ operations to find a preimage. This may be acceptable if the search operation is to be carried out only once. However, if the search operation is expected to be carried out for different points in the future -a more likely case to happen in real life, the time required by exhaustive search may turn out to be excessive. To overcome this problem, one may first construct a lookup table containing the preimages of all the points. This precomputation process requires an effort equal to exhaustive search and is to be performed once. Afterwards,

any preimage finding task can be accomplished via one table lookup operation which requires a negligible amount of time. Although table lookup is quite fast, it requires extreme amounts of memory for the functions acting on large sets. A method to balance this huge gap between the solution time and the required memory, in other words trading memory for time, would be particularly useful.

In 1980, Hellman proposed the Time-Memory Trade-off (TMTO) method [11], in which the effort spent on the precomputation phase can be shared between the memory to store the results and the time to perform a search. Unlike the exhaustive search and table lookup methods, TMTO is a probabilistic method, that is, the search operation may not find a preimage even if there exists one. The inversion of a function by TMTO method can be stated as follows: We are given a precomputation resource equivalent to the exhaustive search effort. At the end of the precomputation phase we store $M$ units of data and each preimage finding operation takes $T$ units time. The critical point here is that we must balance $M$ and $T$ so that both of them are affordable.

In this thesis we study the inversion problem of one-way functions defined over finite sets with TMTO method. Although this method has applications only in cryptography, we present a general framework for the method.

In Chapter 2, we explain the notion of a one-way function and give examples of some of the one-way functions used in cryptography and define the inversion problem of one-way functions.

Chapter 3 is devoted to random mapping statistics where we derive some of the important properties of random mappings which will help us in selecting various TMTO parameters. The properties declared here are of great importance since we model our function to be inverted as a random function.

Chapter 4 constitutes the main content of this study. We first explain exhaustive search and table lookup methods and show how the TMTO method can be used as an alternative to these methods. After explaining the basic method, we derive our calculations regarding the method and compare these with the previous ones in the literature. We basically examine three variants of the TMTO method, namely the original method by Hellman, the distinguished points method and the rainbow tables method.

In Chapter 5 we show how the TMTO method can be applied to symmetric-key cryptosystems and hash functions. As an example, we apply the method to the SHA-1 algorithm with a key space of $2^{40}$ inputs, with three different time and memory settings. For each of the settings we present the results.

Chapter 6 concludes the thesis with a summary. We also mention future work and open problems related to the TMTO method in this chapter.

# Chapter 2

# One-Way Functions

This chapter deals with one-way functions and the problem of inverting these functions. After introducing the concept of one-wayness, the importance of these functions in cryptography is emphasized. It is made clear that inverting these functions is equivalent to breaking the cryptosystem that depends on them. Finally, the inversion problem is introduced. This problem can be classified into two categories; then why we concentrate on one of these types is explained .

## 2.1 One-Way Functions

We begin by defining a one-way function.

**Definition 2.1.1.** A function $f : X \to Y$ where $X$ and $Y$ are non-empty finite sets is called one-way if for any $x \in X$, there exists a polynomial time algorithm to compute $y = f(x)$, but for an arbitrary $y \in Y$ a polynomial time algorithm to find $x = f^{-1}(y)$ does not exist.

It must be noted that the proof of the non-existence of a polynomial time inversion algorithm for such a function may not be so easy to provide. In fact, the existence of one-way functions is an open problem in complexity theory.

Therefore, we treat the function as one-way unless a method for inversion is known.

Some of the examples of one-way functions worth mentioning are:

- Integer factorization: It is easy to compute the product of two integers but no efficient way to factorize an arbitrary integer is known.
- Discrete Logarithm Problem: In a finite field $F_{p^k}$ of order $p^k$ where $p$ is a prime, given $g, a \in F_{p^k}$, computing $y = g^a \ (mod\ p^k)$ requires polynomial time. However given $y, g \in F_{p^k}$ there exists no such algorithm to find $a$, such that $y = g^a \ (mod\ p^k)$.
- Knapsack Problem: Given a nonempty set $A$ of integers, it is quite easy to calculate the sum of the elements of any subset of $A$. However, given the set $A$ and an integer $y$ it is hard to find a subset whose elements sum up to $y$.

There is a special class of one-way functions called trapdoor functions.

**Definition 2.1.2.** A function $f : X \to Y$ where $X$ and $Y$ are non-empty finite sets is called a trapdoor function if it is a one-way function with the distinction that it is possible to compute $x = f^{-1}(y)$ easily with some extra information.

As with the definition of one-way functions, the meanings of 'easy' and 'hard' also apply here. The difference between a one-way function and a trapdoor function is that while it is always hard to invert a one-way function, trapdoor functions enable the inversion if some additional data is known. Trapdoor functions are mostly used in public key cryptosystems. Perhaps the most famous trapdoor function is the RSA encryption function $RSA(n, e, m) = m^e (mod\ n)$. A sender $A$ who wishes to send a message $m$ to a receiver $B$ having RSA

public key parameters $n$ and $e$, computes $c = RSA(n, e, m)$ and sends $c$ to $B$. For anybody without the knowledge of secret parameter $d$, obtaining $m$ from $c$ is considered to be a hard problem and equals the breaking of the system. However the recipient of the message who possesses secret information $d$ can easily compute $m = c^d \ (mod\ n)$, and hence can invert the encryption function.

## 2.2 The Inversion Problem

By inversion of a one-way function, we mean a process which finds a preimage of any point in the range set, if there exists any. Here we specifically work on the functions defined on finite sets. Depending on the structure of the function, whether it is 1-1 or not, for example, some of the points may have more than one preimage, and some of them may not have a preimage at all. Therefore, we analyze the problem of inversion in two categories.

Let $f : A \rightarrow B$ be a function defined over the finite sets $A$ and $B$.

- 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 $x_0$.

While in some situations finding one out of many preimages may be sufficient, finding a specific preimage may also be of interest.

It must be noted that the solution of the problem of type 2 is equivalent to the solution of all the instances of type 1: that is, in order to find a specific preimage, all the preimages should be searched.

# Chapter 3

# Random Mappings

In this chapter we study some properties of random mappings which will provide us with a number of useful results to be used in the next chapter.

Functions we are particularly interested in are the ones which are defined on large finite sets. In most cases they are defined by complicated mechanisms; therefore, we may assume that for each input value, the output is selected among possible outputs equally likely.

Some of the properties given in this chapter are well known and can be obtained by elementary methods. For details and further properties of random mappings, the reader may refer to article [8], where most of the **basic** properties are given.

## 3.1 Random Mapping Statistics

Let $X$ and $Y$ be finite sets with $|X| = N$ and $|Y| = M$. The following can be found in any elementary text book.

- The number of all functions $X \to Y$ is $M^N$.
- The number of one-to-one functions $X \to Y$ is $\frac{M!}{(M-N)!}$, provided $M > N$.

- The number of onto functions $X \to Y$ is

$$\sum_{i=0}^{m}(-1)^i\binom{m}{i}(m-i)^n$$

provided $M < N$.

- The number of bijective functions $X \to Y$ is $M!$, provided $M = N$.

Now, fix a positive integer $k \leq N$ and let $X_k$ be a random subset of $X$. By $E_k(M)$, we denote the expected size of $f(X_k)$. In other words, if we pick (with replacement) $k$ integers randomly, out of $M$ integers, then $E_k(M)$ is the expected number of 'distinct' integers in our collection. If $M$ is clear for the given context, we just write $E_k$ to mean $E_k(M)$. Throughout the chapter, $X, Y, M, N, k, E_k, E_k(M)$ are as defined here.

**Proposition 3.1.1.** $E_k = M - M(1 - \frac{1}{M})^k$.

*Proof.* Obviously $E_1 = 1$ and $E_2 = 1 + (\frac{M-1}{M})$. Since $E_{k+1} = E_k + (1 - \frac{E_k}{M})$, we get $E_{k+2} = E_{k+1}(2 - \frac{1}{M}) - E_k(1 - \frac{1}{M})$. Characteristic equation of this recursion is $(r-1)(r-(1-\frac{1}{M}) = 0$, hence $E_k$ is of the form $E_k = A + B(1-\frac{1}{M})^k$, for some $A, B \in R$. Initial conditions imply that $A = -B = M$ and consequently $E_k = M - M(1 - \frac{1}{M})^k$ □

If a few number of integers are picked randomly out of the first 1000 integers, it is quite likely to have all of them distinct. On the other hand, if we pick a large number of integers, intuitively we may expect a large number of collisions, that is to say, $E_2(1000) \approx 2$, $E_3(1000) \approx 3$, $E_{900}(1000) < 900$. It is natural to ask for the largest $k$ for which we can expect (with high probability) a collection of all distinct integers or the smallest $k$ for which we expect (with high probability) a collision.

It is obvious that $E_k \leq k$. More precisely, we have $E_1 = 1$ and $E_k < k$ for $k \geq 2$. In choosing $k$ integers out of $M$ integers, we can expect a collection of distinct integers if $k - E_k$ is small enough. If $k - E_k < \epsilon$ is required to accept a collection to be distinct we have to pick at most $k_0$ integers such that $E_{k_0} > k_0 - \epsilon$. Now we solve $E_k > k - \epsilon$ for $k$.

$$M(1 - (1 - \frac{1}{M})^k) > k - \epsilon$$

gives

$$1 - \frac{k}{M} + \frac{\epsilon}{M} > 1 - \frac{k}{M} + \frac{k(k-1)}{2M^2} + \ldots > \frac{k^2}{2M^2}$$

which yields $2M\epsilon > k^2$. That is to say, if we choose $k_0$ integers where $k_0 < \sqrt{2M\epsilon}$, the expected value of distinct integers will be $k - \epsilon$ on the average. In fact, in this case the probability of having a distinct collection is given by $\frac{M!M^{-k}}{k_0!}$. Substituting $k_0 = \sqrt{2M\epsilon}$ we get $\frac{M!M^{-\sqrt{2M\epsilon}}}{\sqrt{2M\epsilon}!}$ which, by Stirling's approximation can be written as

$$Probability(|f(X_k)| = k) = \left(\frac{M}{M - \sqrt{2M\epsilon}}\right)^{M - \sqrt{2M\epsilon} + \frac{1}{2}} e^{-\sqrt{2M\epsilon}} \qquad (3.1.1)$$

As $k$ gets larger, it is quite likely to have collisions and on the other extreme, as $k$ gets smaller, the choice will consist of distinct integers with high probability. By choosing $\epsilon = 1/2$, the above equation implies that $k = \sqrt{M}$ is a critical point. That is, for that choice of $k$, $E_k \approx k$ and a collection may or may not have collisions with considerable probability values. For example, for $M = 10000$

and $k = 100$, $E_k \approx 99.5$ and the probability that all integers are distinct is 0.61, the probability of having a collision is 0.39.

The following table shows the probabilities of having distinct collections for the set size $M$ and the selection size of various multiples of $\sqrt{M}$. From the table it can be deduced that for in order to have a distinct collection with a high probability, the selection size can be chosen as a little fraction of the square root of the set size. For example, taking the selection size 1/8 square root of the set size guarantees having a distinct collection with a probability over 99%.

| $M$ | $Pr(\lvert f(X_{\sqrt{M}})\rvert = \sqrt{M})$ | $Pr(\lvert f(X_{\sqrt{M}/2})\rvert = \sqrt{M}/2)$ | $Pr(\lvert f(X_{\sqrt{M}/8})\rvert = \sqrt{M}/8)$ |
|---|---|---|---|
| 1,000 | 61.31% | 88.90% | 99.42% |
| 10,000 | 60.86% | 88.45% | 99.28% |
| 100,000 | 60.72% | 88.31% | 99.24% |
| 500,000 | 60.68% | 88.28% | 99.23% |

Table 3.1: Probability of having a distinct collection for various multiples of $\sqrt{M}$.

## 3.2 Graph Representation of Functions

In this section we consider the graph representation of functions. Because the TMTO method which will be explained in the next chapter depends on the iterative evaluation of the functions, this representation and its various statistical properties will provide us with a very good understanding of the method.

**Definition 3.2.1.** A function graph of a function $f : X \to X$ 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 iteratively apply $f$, 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.

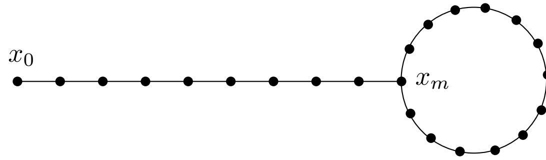


Figure 3.1: Graphical representation of an iteration operation

In Fig. 3.1 we can see the typical behaviour of an iteration operation. Starting with a point $x_0$, the iteration enters a loop after the point $x_m$, forming a cycle. The points in a function graph, therefore, can be categorized as belonging to a cycle or belonging to a path that connects to a cycle.

A function graph consists of a set of connected components. A component consists of a number of trees connected to a cycle. A tree is a connected subgraph with $n$ nodes and $n-1$ edges. Terminal points are the points that do not have a preimage. Image points are the points having at least one preimage. The length of the path is called the tail length. The length of the cycle is called the cycle length. Rho-length is the sum of the tail length and the cycle length.

For the proofs of the following theorems, the reader may refer to [8].

**Theorem 3.2.2.** *The expectations of parameters, the number of components, the number of cyclic points, the number of terminal points, the number of image points and the number of k-th iterate image points in a random mapping of size n have the following asymptotic forms as $n \to \infty$.*

1. The number of components in a function graph of size $n$ is $\frac{1}{2} logn$.

2. The expected number of cyclic nodes in a function graph of size $n$ is $\sqrt{\frac{\pi n}{2}}$.

3. The expected number of terminal points in a function graph of size $n$ is $e^{-1}n$.

4. The expected number of image points in a function graph of size $n$ is $n(1-e^{-1})$.

5. The expected number of $k-th$ iterate image points in a function graph of size $n$ is $(1-\tau_k)n$, where $\tau_k$ satisfies the recurrence $\tau_0 = 0$, $\tau_{k+1} = e^{-1+\tau_k}$.

The tree size of node $\mu$ is the size of the maximal tree (rooted on a cycle) containing $\mu$. The component size means the size of the connected component that contains $\mu$. The predecessors size of $\mu$ is the size of the tree rooted at $\mu$ or equivalently, the number of iterated preimages of $\mu$.

**Theorem 3.2.3.** *The expectation of the parameters tail length, cycle length, rho length, tree size, component size, and predecessors size have the following asymptotic forms.*

- The expected tail length $\lambda$ of a function graph of size $n$ is $\sqrt{\frac{\pi n}{8}}$.
- The expected cycle length $\mu$ of a function graph of size $n$ is $\sqrt{\frac{\pi n}{8}}$.
- The expected rho length $\rho = \lambda + \mu$ of a function graph of size $n$ is $\sqrt{\frac{\pi n}{2}}$.
- The expected tree size of a function graph of size $n$ is $\frac{n}{3}$.
- The expected component size of a function graph of size $n$ is $\frac{2n}{3}$.
- The expected predecessors size of a function graph of size $n$ is $\sqrt{\frac{\pi n}{8}}$.

As an example, the functional graph of the giant component of the block cipher DES is given in Figure 3.2. This figure, taken from [8], is due to Quisquater and Delescaille. The graph is formed by iterating the DES cipher for its $2^{56}$ key values for a fixed plaintext. In the graph, one point in every $10^6$ points is sampled.

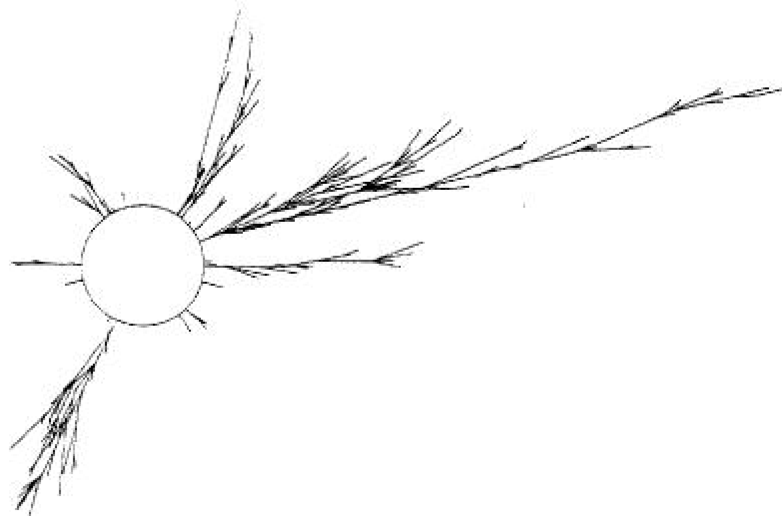

Figure 3.2: A rendering of the giant component of DES due to Quisquater and Delescaille.

# Chapter 4

# Time-Memory Trade-off Method

In this chapter, we give an analysis of the TMTO method introduced by Hellman in [11]. Before presenting the method we briefly describe exhaustive search and table lookup methods and mention the drawbacks that come about whenever they are expected to be used multiple times to find a solution to a problem. We describe the three variants of the TMTO method; namely the original method by Hellman, distinguished points by Rivest [7] and rainbow tables by Oechslin [14].

## 4.1 Exhaustive Search vs. Lookup Table

Whenever it is the case that there is a finite number of **solutions** to a problem, exhaustive search is the simplest way of finding the **solution** by checking each possible **solution** one by one until the correct **solution** is found. The method is a generic one, that is, the existence of a procedure that checks whether a specific solution is valid or not is sufficient for the method to work.

In the context of this thesis, we are interested in the problem of finding the preimages of one-way functions. More specifically, we are given a function

$f : X \rightarrow Y$ with $|Y| = N$ and an element $y \in Y$, and we try to find an element $x \in X$ such that $f(x) = y$. We consider $f$ as a black box function, and hence are not interested in how the computation is carried out in order to find the output.

The expected number of operations to find a solution in exhaustive search is $|N|/2$, which is considerably a large value depending on the size of the range set. This becomes a handicap when the search operation is to be repeated over and over. Memory usage is negligible.

A way of speeding-up the search operation is employing a lookup table. Here, one stores the results of a precomputation in memory. After this, any search operation can be done via one lookup operation. The drawback of this method is that it requires too much memory when the size of the range set is large.

A comparison of exhaustive search and table lookup methods is given in Table 4.1. In exhaustive search, no precomputation is performed and the search operation does not need any memory. However, each search operation requires approximately $|N|/2$ operations. This is the major drawback of the method if the search operation is to be carried out many times. For $m$ searches, the expected search time will be in the order of $m\frac{|N|}{2}$.

On the other hand, the table lookup method needs a precomputation phase which requires a computation effort equivalent to $|N|$ operations, and the results of this precomputation are stored in memory requiring a storage capacity in the order of $|N|$. However once this step is completed, each search means a single lookup operation which requires a negligible amount of time.

| | Exhaustive Search | Table Lookup |
|---|---|---|
| Precomputation | - | $N$ |
| Memory | 1 | $N$ |
| Time | $\frac{N}{2}$ | 1 |
| Time (m) | $m\frac{N}{2}$ | $m$ |

Table 4.1: Comparison of exhaustive search and table lookup methods.

## 4.2 Hellman's Method

The TMTO method consists of two phases, namely the online phase and the offline phase. The offline phase consists of a precomputation process where the results are stored. In the online phase we try to find a preimage of a given point in the range.

Let $f : \{0, 1, \ldots, N-1\} \rightarrow \{0, 1, \ldots, N-1\}$ be a random function. We assume that it is easy to compute $y = f(x)$, but hard to compute $x = f^{-1}(y)$. Our aim is to find a preimage of a given point in the range of $f$. In order to achieve this, we generate chains by iteratively evaluating $f$ starting at random points. A *chain* of length $t$ starting at point $x$ is the sequence $C_0 = x, C_1 = f(x), C_2 = f(f(x)) = f^2(x), \ldots, C_t = f^t(x)$. The first element of a chain is called the starting point and the last element the end point.

A property of a chain is that once we compute it, we do not need to store all the elements in it. By knowing the starting point, we can recalculate the successive elements in the chain. Another property of a chain is that it acts like a lookup table, holding the preimages of the function for the points $C_1, \ldots, C_t$, that is $C_i = f^{-1}(C_{i+1})$ for $0 \leq i \leq t-1$. In Fig 4.1. the chain construction process is depicted.

The offline phase calculation in TMTO is carried as follows. We select $m$ distinct points in the domain of $f$ and generate $m$ *chains*, each of length $t$. In

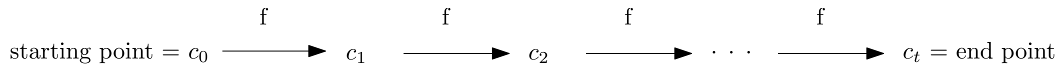


Figure 4.1: Construction of a chain

this way, we generate a $m \times (t+1)$ matrix which we call a Hellman Table or Hellman Matrix as shown in Fig 4.2.

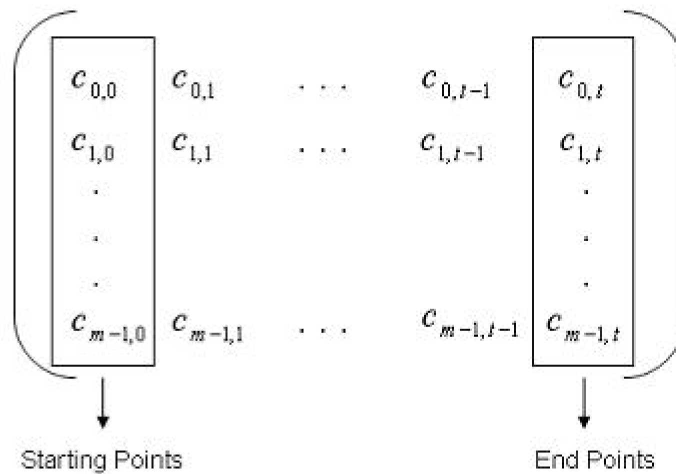


Figure 4.2: Hellman's Matrix

We only store the first and last columns of this matrix, namely the starting point and the end point tuples and sort these tuples with respect to the end points.

The online phase calculation is a search for a preimage of a given point $y$ in this matrix and is carried out as follows. First, we check if $y$ is equal to any of the end points. This can be done efficiently with binary search, since data is sorted with respect to end points. If $y$ is equal to an end point $EP_i$, then we start with the corresponding starting point $SP_i$ and calculate $f^{t-1}(SP_i) = x_{i,t-1}$, which is the entry just before the end point. By making this calculation, we check if $y$ is in the last column of the matrix, and hence try to find the preimage by computing the previous entry in the matrix. If $y$ is not equal to any of the

end points, we calculate $f(y)$ and check for a match in the $(t-1)^{st}$ column. If a match is found again we start with the corresponding starting point and calculate the $(t-2)^{nd}$ entry in the table. Unlike the search for the last column, in this case and for the searches from this point there may be two cases:

i) If the point in the $(t-2)^{nd}$ column is evaluated and the desired range point is found, a preimage has been found, hence the search process is completed.

ii) If the point in the $(t-2)^{nd}$ column is evaluated and the desired range point is not found, then a situation called a false has occurred. If this is the case then we continue the search operation since a preimage has not been found.

We repeat the search process until we check whether $y$ exists in the second column of the matrix.

Constructing a single table requires $m \times t$ evaluations of the function; the storage requirement is of order $m$ and the time to search for a preimage is of order $t$.

The number of preimages we can find in a table is the number of distinct entries in the first $t$ columns which is called the *coverage* of the table. The following theorem about the coverage of a single table is given in [11].

**Theorem 4.2.1.** *The probability of success of a table of size* $(m, t)$ *is bounded by*

$$P(S) \geq \sum_{i=1}^{m} \sum_{j=0}^{t-1} [(N - it)/N]^{j+1}$$

*Proof.* Letting $A$ denote the subset of points covered by the first $t$ columns of an Hellman matrix, (not including the end points). We have

$$P(S) = \frac{E|A|}{N},$$

where $|A|$ denotes the number of elements in $A$. Letting $I\{X\}$ denote the indicator function of the event $X$,

$$\begin{aligned} P(S) &= E\sum_{i=1}^{m}\sum_{j=0}^{t-1}\{I\{X_{ij} \text{ is new}\}\}/N \\ &= \sum_{i=1}^{m}\sum_{j=0}^{t-1}\{Pr(X_{ij} \text{ is new})\}/N \end{aligned}$$

where a point being 'new' means it has not occurred in a previous row or thus far in its row. Using

$$\begin{aligned} Pr(X_{ij} \text{ is new}) &\geq Pr(X_{i0}, X_{i1}, \ldots, X_{ij} \text{ are all new}) \\ &= Pr(X_{i0} \text{ is new})Pr(X_{i1} \text{ is new}|X_{i0} \text{ is new})\ldots \\ &\quad Pr(X_{ij} \text{ is new} \mid X_{i0}, X_{i1}, \ldots, X_{i,j-1} \text{ are new}) \\ &= \frac{N-|A_{i0}|}{N}\frac{N-|A_{i0}|-1}{N}\ldots\frac{N-|A_{i0}|-j}{N}, \end{aligned}$$

when $A_{ij}$ denotes the set of elements covered so far. Clearly each factor in Equation 4.2.1 is larger than $\frac{(n-it)}{N}$ since there are at most $t$ different elements in each row. Therefore

$$Pr(X_{ij} \text{ is new}) \geq [(N-it)/N]^{j} + 1$$

and

$$P(S) \geq \sum_{i=1}^{m} \sum_{j=0}^{t-1} [(N - it)/N]^{j+1}$$

completing the proof.

□

An important result Hellman deduced from Theorem 4.2.1. is that when $mt^2 >> N$, the ratio of distinct elements appearing in a table will start to decrease significantly. Hence, Hellman suggested that $m$ and $t$ should be chosen to satisfy $mt^2 = N$ to obtain maximum coverage rate. But if $m$ and $t$ are chosen in this way, a single table of dimension $m \times t$ will cover only $\frac{N}{mt}$ of the domain space. The solution is to construct $k = \frac{N}{mt}$ different tables, but another problem with this setting is the collision of entries of different tables which will result in generating the overlapping sequences and decreasing the overall coverage. In order to avoid collisions, the notion of a reduction function is defined. A reduction function $r : \{0, 1, \ldots, N-1\} \rightarrow \{0, 1, \ldots, N-1\}$ is a bijective function such as a permutation or adding by an integer which we compose with $f$. Thus, the iteration function in the chain generation becomes $r(f(x))$ as depicted in Fig 4.3. We choose a different $r$ for each table so if two entries in two tables are the same, the usage of different reduction functions will result in different elements in the next chain item. The chain construction under a reduction function is depicted in Fig. 4.3.

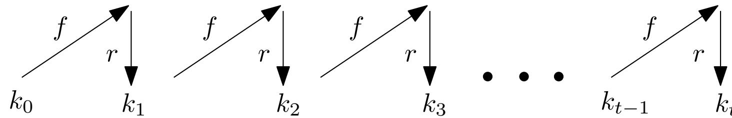


Figure 4.3: Construction of a chain under the reduction function $r$

Assuming that each reduction function defines an independent random func-

tion, the following theorem states the success probability of $k$ tables.

**Theorem 4.2.2.** *If the overall coverage of a single table of size $m \times t$ is $C(m,t)$, then the overall coverage of $k$ independent tables of size $m \times t$ is* $(1-(1-C(m,t)))^k$

*Proof.* The probability that a point is not in one table is $1-C(m,t)$. The probability that a point is not in $k$ different tables is $(1-C(m,t))(1-C(m,t))\ldots(1-C(m,t))$, which is equal to $(1-C(m,t))^k$. If we subtract this expression from 1 we obtain the probability that a point is in one of the $k$ tables, which is:

$$(1-(1-C(m,t)))^k.$$

□

Since the overall coverage of a single table is a small quantity, the above formula can be approximated by $1-e^{-C(m,t)k}$.

To summarize the Hellman's method, in the offline phase, we construct $k$ tables of dimension $m \times t$ and store the start point-end point tuples sorted with respect to the end points. In the online phase we are given a point in the range of $f$ and we search the preimage of this point in the tables we have constructed in the offline phase. We make $mtk = N$ evaluations of the function $f$ in the offline phase and need memory of order $mk$ to store the resulting data. In the online phase $kt$ calculations are required in the worst case for searching. The memory complexity is directly proportional to the number of starting points $m$, and the number of tables $k$ and the time complexity is directly proportional to the chain length $t$ and the number of tables $k$.

**An Explicit Asymptotic Expression for the Coverage of a Single Table**

Now we derive a closed formula for the number of distinct points covered by a single Hellman table which first appeared in [6].

**Definition 4.2.3.** Coverage of a function $f : \{0, 1, \ldots, N-1\} \to \{0, 1, \ldots, N-1\}$ is the number of distinct points appearing in the range of $f$.

We state a well-known property as the following theorem. The approach used in the proof of this theorem will be used in obtaining a more general result.

**Theorem 4.2.4.** *The expected value of the coverage of a random function* $f : \{0, 1, \ldots, N-1\} \to \{0, 1, \ldots, N-1\}$ *is*

$$N(1 - e^{-1}).$$

*Proof.* At each step, we choose an integer between 1 and $N$. After $k^{th}$ step, let $A_k$ be the expected number of distinct integers we have collected so far. Obviously $A_1 = 1,\ A_{n+1} = A_n + 1 - \frac{A_n}{N}$ and

$$A_{n+1} - A_n \;=\; 1 - \frac{A_n}{N}.$$

Let $\Delta$ be the difference operator. Since $\Delta A_n = A_{n+1} - A_n$ and $\Delta n = 1$ we have

$$\Delta A_n \;=\; (1 - \frac{A_n}{N})\Delta n.$$

For large values of $N$, since $\Delta n = 1$ is very small we may expect a "continuous-like" behaviour of $A_n$. Thus, we consider the continuous function $A(n)$ and

replace the discrete difference operator $\Delta$ with its continuous counterpart $d$, namely the differential operator. Then the last equation can be rewritten as

$$dA = (1 - \frac{A}{N})dn,$$

which gives the solution

$$N - A = Ke^{-n/N}.$$

The initial conditions $A(0) = 0$ forces $K = N$. Thus we conclude

$$A(n) = N(1 - e^{-n/N}).$$

For $n = N$ we have $A(n) = N(1 - e^{-1})$. □

We now calculate the coverage of a single table. We choose $m$ distinct starting points. The complexity requirements indicate that the size of $m$ should be about $N^{1/3}$ which is obviously quite smaller than $N^{1/2}$. So there is no harm in assuming that $m << N^{1/2}$. Thus, we can safely assume that all elements in a column are distinct. Let $a_j$ denote the number of distinct elements in column $j$. Then

$$\begin{aligned} a_1 &= m, \\ a_j &= \frac{N - \sum_{i=1}^{j-1} a_i}{N} \cdot a_{j-1}. \end{aligned}$$

Now let $y_j$ be the total number of distinct elements in the first $j$ columns.

Then

$$y_j - y_{j-1} = \frac{N - y_{j-1}}{N} \cdot (y_{j-1} - y_{j-2}),$$

or employing the difference operator $\Delta$,

$$\Delta y_j = \frac{N - y_{j-1}}{N} \cdot \Delta y_{j-1},$$

and

$$N\Delta^2 y_j \quad + \quad \Delta y_{j-1} y_{j-1} = 0.$$

As in the proof of the theorem we now replace $\Delta$ with $d$,

$$Nd^2 y + dy.y = 0,$$

or

$$Ny^{''} + y^{'} y = 0.$$

Subtituting $y^{'} = z$ we get

$$\frac{dz}{dy}\frac{dy}{dt} = -\frac{y}{N}\frac{dy}{dt},$$

or

$$dz = -\frac{y}{N}dy,$$

which gives the solution

$$y^{'} = c_1 - \frac{y^2}{2N},$$

where $c_1$ is a constant. The initial conditions, $y(0) = 0,\ y^{'}(0) = m$ give $c_1 = m$. So

$$y^{'} \quad = \quad m - \frac{y^2}{2N},$$

and

$$\frac{dy}{2Nm - y^2} \quad = \quad \frac{dt}{2N},$$

hence

$$\frac{1}{2\sqrt{2Nm}} ln\left(\frac{\sqrt{2Nm} + y}{\sqrt{2Nm} - y}\right) \quad = \quad \frac{t}{2N} + c_2,$$

where $c_2$ is a constant. From the initial conditions we find that $c_2 = 0$. Putting $\alpha = \sqrt{2Nm}$, we have

$$\frac{1}{\alpha} ln(\frac{\alpha + y}{\alpha - y}) = \frac{t}{N},$$

or

$$
\begin{aligned}
y &= \frac{e^{\frac{\alpha t}{N}}-1}{e^{\frac{\alpha t}{N}}+1}\ .\ \alpha, \\
&= \alpha.tanh\left(\frac{\alpha}{2N}t\right), \\
&= \sqrt{2Nm}.tanh\left(\sqrt{\frac{m}{2N}}t\right).
\end{aligned}
$$

The following table compares the Hellman's coverage formula with our new asymptotic expression as well as with the empirical values. The rows of the table are the height and the columns of the table are the width of a single Hellman table. Here, the domain size is $2^{24}$. The $H$, $N$ and $E$ columns correspond to the Hellman's coverage formula, the new coverage formula and the empirical coverage values, respectively. The table shows how closely our new asymptotic expression approximates the empirical values.

| t | $2^7$ | | | $2^8$ | | | $2^9$ | | |
|---|---|---|---|---|---|---|---|---|---|
| m | H | N | E | H | N | E | H | N | E |
| $2^5$ | 0.9947 | 0.9948 | 0.9948 | 0.9790 | 0.9797 | 0.9799 | 0.9220 | 0.9242 | 0.9217 |
| $2^6$ | 0.9896 | 0.9897 | 0.9898 | 0.9597 | 0.9603 | 0.9601 | 0.8595 | 0.8611 | 0.8612 |
| $2^7$ | 0.9795 | 0.9797 | 0.9781 | 0.9237 | 0.9242 | 0.9219 | 0.7623 | 0.7616 | 0.7597 |
| $2^8$ | 0.9602 | 0.9603 | 0.9593 | 0.8609 | 0.8611 | 0.8580 | 0.6353 | 0.6282 | 0.6254 |
| $2^9$ | 0.9241 | 0.9242 | 0.9221 | 0.7633 | 0.7616 | 0.7601 | 0.5006 | 0.4820 | 0.4803 |
| $2^{10}$ | 0.8612 | 0.8611 | 0.8596 | 0.6359 | 0.6282 | 0.6260 | 0.3798 | 0.3511 | 0.3500 |
| $2^{11}$ | 0.7633 | 0.7616 | 0.7602 | 0.5008 | 0.4820 | 0.4803 | 0.2814 | 0.2498 | 0.2488 |
| $2^{12}$ | 0.6356 | 0.6282 | 0.6242 | 0.3797 | 0.3511 | 0.3487 | 0.2054 | 0.1768 | 0.1756 |

Table 4.2: Comparison of the coverage formulas with empirical values for a single table of size $m \times t$ on a domain of size $2^{24}$.

**Results Emerging From the New Formula**

The ratio of overall coverage to $N$ of $k$ tables is given by

$$Y(m,t,k) = 1 - \left(1 - \sqrt{\frac{2m}{N}}.tanh\left(\sqrt{\frac{m}{2N}}t\right)\right)^k.$$

Defining the total complexity as $C = \frac{N^2}{mt}$ and imposing the natural condition $mtk = N$, we obtain $mt = \frac{N^2}{C}$ and $k = \frac{C}{N}$. Substituting these values in the expression for overall coverage we get

$$Y(m,C) = 1 - \left(1 - \sqrt{\frac{2m}{N}}.tanh\left(\sqrt{\frac{N}{2m}}\frac{N}{C}\right)\right)^{\frac{C}{N}}.$$

As an example, consider $N = 2^{64}$, $C = 2^{85.33}$. For various values of $m$, we obtain the Table 4.3.

In the table, $m$, $t$ and $k$ correspond to the usual parameters of the classical TMTO method. Coverage is the success rate of the method, $C$ is the total complexity which is also the product of time and memory complexities and $M$ and $T$ are the memory and time complexities. The table shows us how to adjust memory and time complexities to get the corresponding success rate for a fixed total complexity value.

| $m$ | $t$ | $k$ | Coverage | $C$ | T | M. |
|---|---|---|---|---|---|---|
| 1,321,123 | 5,284,491 | 2,642,247 | 0.5331 | 85.3333 | 43.6667 | 41.6667 |
| 2,642,246 | 2,642,245 | 2,642,247 | 0.5773 | 85.3333 | 42.6667 | 42.6667 |
| 3,963,369 | 1,761,497 | 2,642,246 | 0.5942 | 85.3333 | 42.0817 | 43.2516 |
| 5,284,492 | 1,321,122 | 2,642,247 | 0.6032 | 85.3333 | 41.6667 | 43.6667 |
| 6,605,615 | 1,056,898 | 2,642,246 | 0.6087 | 85.3333 | 41.3447 | 43.9886 |
| 7,926,738 | 880,748 | 2,642,247 | 0.6125 | 85.3333 | 41.0817 | 44.2516 |
| 9,247,861 | 754,927 | 2,642,247 | 0.6152 | 85.3333 | 40.8593 | 44.4740 |
| 10,568,984 | 660,561 | 2,642,247 | 0.6172 | 85.3333 | 40.6667 | 44.6667 |
| 11,890,107 | 587,165 | 2,642,249 | 0.6188 | 85.3333 | 40.4967 | 44.8366 |
| 13,211,230 | 528,449 | 2,642,246 | 0.6201 | 85.3333 | 40.3447 | 44.9886 |
| 14,532,353 | 480,408 | 2,642,247 | 0.6212 | 85.3333 | 40.2072 | 45.1261 |
| 15,853,476 | 440,374 | 2,642,247 | 0.6221 | 85.3333 | 40.0817 | 45.2516 |
| 17,174,599 | 406,499 | 2,642,248 | 0.6229 | 85.3333 | 39.9662 | 45.3671 |
| 18,495,722 | 377,463 | 2,642,250 | 0.6235 | 85.3333 | 39.8593 | 45.4740 |
| 19,816,845 | 352,299 | 2,642,249 | 0.6241 | 85.3333 | 39.7598 | 45.5736 |
| 21,137,968 | 330,280 | 2,642,251 | 0.6246 | 85.3333 | 39.6667 | 45.6667 |
| 22,459,091 | 310,852 | 2,642,249 | 0.6250 | 85.3333 | 39.5792 | 45.7541 |
| 23,780,214 | 293,582 | 2,642,253 | 0.6254 | 85.3333 | 39.4967 | 45.8366 |
| 25,101,337 | 278,131 | 2,642,247 | 0.6257 | 85.3333 | 39.4187 | 45.9146 |
| 26,422,460 | 264,224 | 2,642,251 | 0.6261 | 85.3333 | 39.3447 | 45.9886 |
| 27,743,583 | 251,642 | 2,642,250 | 0.6263 | 85.3333 | 39.2743 | 46.0590 |

Table 4.3: The relation between $m$, coverage, time and memory complexities for a fixed total complexity value of $C = 2^{85.33}$

**False Alarms**

False alarms are an important factor that affect the online time complexity of the TMTO method.

Hellman gives the following bound for the number of expected false alarms in [11].

**Theorem 4.2.5.** *The expected number of false alarms per table tries, $E(F)$, is bounded by*

$$E(F) \leq mt(t+1)/2N$$

*Proof.* Letting $F_{ij}$ denote the occurrence of a false alarm due to $Y_j = EP_i$,

$$E(F) \leq \sum_{i=1}^{m} \sum_{j=1}^{t} Pr(F_{ij}).$$

$F_{ij}$ can occur in $j$ different ways: due to $f(k)$ merging immediately with the $i$th row of the matrix, that is if $f(K) = f^{t-j+1}(SP_i)$, or merging after one iteration, that is if $f(K)$ is not in the $i$th row of the matrix, but $f^2(K)$ equals $f^{t-j+2}(SP_i)$; etc. Each of these $j$ different ways of causing $F_{i,j}$ to occur has probability at most $1/N$ because, up tp the merging, $K$, $f(K)$, etc. are independent random variables uniformly distributed over $\{1, 2, \ldots, N\}$. Therefore

$$\begin{aligned} E(F) &\leq \sum_{i=1}^{m} \sum_{j=1}^{t} j/N \\ &= mt(t+1)/2N, \end{aligned}$$

completing the proof. $\square$

In [1], Avoine et al. study the false alarms in detail and propose a method to detect them. In usual TMTO online search, once a match with an end point is found, the chain is regenerated from the starting point to reach to the desired position. However, if the situation is a false alarm, the performed computation becomes useless.

The new method uses extra memory for each chain of a table. This memory is used to store a hash value of some specific column position called 'checkpoint'. The hash value of the checkpoint is calculated in the precomputation process and is stored with each start-end point pair.

According to [1], a performance improvement of 10.99% is possible at the expense of using 0.89% extra memory. As the authors of the article state, this is a novel technique that needs to be examined.

## 4.3 Distinguished Points

One of the drawbacks of Hellman's TMTO method is the amount of table lookups. In the online phase, a table lookup operation is performed after each invocation of the function $f$. To search a table having $t$ columns, $t$ many table lookups have to be made. A method decreasing the number of lookups would also decrease the total search time.

The distinguished points method was suggested by Rivest in [7]. The idea was used in finding collisions in DES by Quisquater and Delescaille [18]. It was analyzed by Borst [5] and applied to DES by Quisquater et. al. [16],[17].

The notion of a distinguished point is defined in [5] as follows;

**Definition 4.3.1.** Distinguished Point

Let $K \in \{0,1\}^k$ and $d \in \{0,1,\ldots,k-1\}$. Then $K$ is a distinguished point (DP) of order $d$ if there is an easily checked property which holds for $K$ and which holds for $2^{k-d}$ different elements of $\{0,1\}^k$.

Here, 'easily checked' means a property that can be checked with a lower complexity than needed for a search in a table of about $2^d$ elements. A easily checked DP-property is for example having $d$ bits with a fixed value on $d$ specific places.

In the distinguished points method, we perform the same calculations as in classical TMTO except generating chains of fixed length $t$. Here, we generate

the chain until a DP is encountered. If a DP is not encountered in $t+1$ iterations, the chain is discarded. Only chains that reach to a DP are stored, along with the chain's length. If two different starting points reach to the same DP, the one with the maximum chain length is stored. Thus, no two chains overlap in a table.

Because the end points consist of DP's, in the online phase we only lookup an entry for the DP's. Since the probability to reach a DP is $2^{-d}$, the number of lookups decrease significantly, improving the online search performance.

## 4.4 Rainbow Tables

A major improvement over Hellman's original TMTO method was given by Oechslin [14]. Oechslin suggested to use a single table of size $mt \times t$ satisfying $mt^2 = N$ which he called a rainbow table. The main difference is the usage of different reduction functions in each column as opposed to Hellman's reduction functions which are constant for each table. The advantage of rainbow tables is that it decreases the online search computations by a factor of 2 compared to classical TMTO.

In Fig 4.4 a comparison of both methods is shown. On the left side, the classical TMTO method with $t$ tables each of size $m \times t$ is shown. In each table a different reduction function is used. On the right side, rainbow tables calculation is shown where there exists one table of size $mt \times t$, and at each column a different reduction function is used.

In Hellman's TMTO method online search complexity (worst case) is $tk$ where t is the iteration count for one table and $k$ is the number of tables. If we take $m = t = k = N^{\frac{1}{3}}$ which are the suggested parameters, the number of online

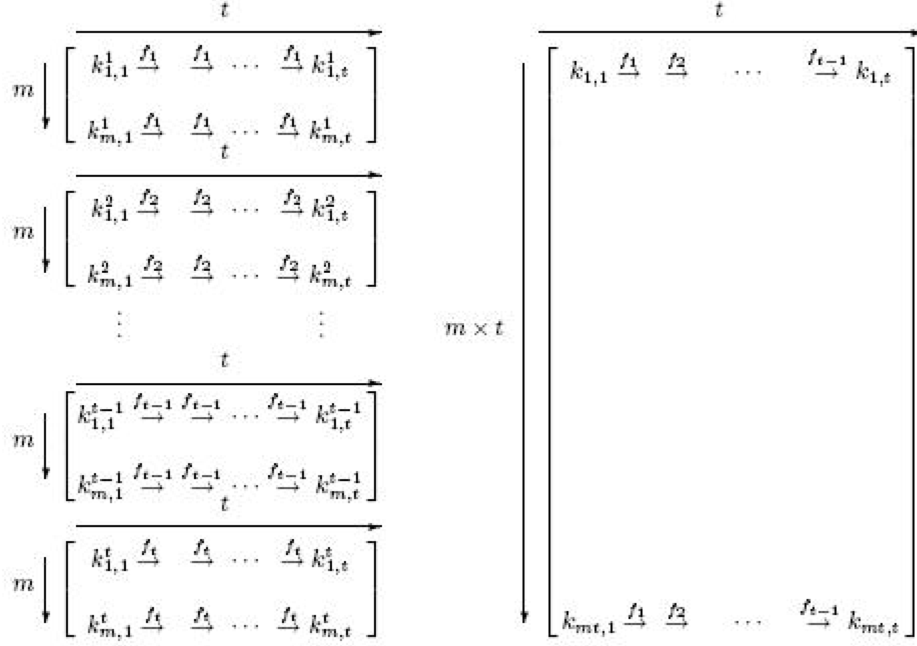


Figure 4.4: Comparison of Hellman tables and Rainbow tables

computations turn out to be $tk = tt = t^2$. However in rainbow tables method, we perform $\frac{(t-1)t}{2} \approx \frac{t^2}{2}$ computations which is approximately half of those of Hellman's method. This difference arises from the fact that in a rainbow table, in order to make a search for a preimage in column $i$, $i - t$ computations need to be done where $t$ is the total column count of the table. Hence, the search operation requires $0 + 1 + 2 + \ldots + (t-2) + (t-1) \ = \ \frac{(t-1)t}{2}$ computations.

The success probability of a single rainbow table can be calculated by counting the total number of different elements in each column. Since the table has $m$ rows, the first column contains $m_1 = m$ different elements. The second column contains $m_2$ distinct elements which are also the images of first column's elements. Because of the collisions, as the column index gets larger, each column will contain less number of distinct elements.

The calculation has been done in [14] as follows.

$$m_2 = N(1 - \left(1 - \frac{1}{N}\right)_1^m) \approx N\left(1 - e^{-\frac{m_1}{N}}\right)$$

Each column $i$ has $m_i$ distinct keys. The success rate of the table is thus:

$$P = 1 - \prod_{i=1}^{t}(1 - \frac{m_i}{N})$$

where

$$m_1 = m \ , \ m_{n+1} = N\left(1 - e^{-\frac{m_n}{N}}\right)$$

.

Although the precomputation phase of rainbow method differs from that of classical TMTO, the success rates of the two methods are very close.

# Chapter 5

# Applications

This chapter is dedicated to the applications of the TMTO method on symmetric-key cryptosystems and hash functions. We explain how to relate one-way functions with these systems and also describe how the TMTO method can be used as a cryptanalysis method on these systems. We finally present our implementation of the TMTO method on SHA-1 hash algorithm with a key space of size $2^{40}$.

## 5.1 Block Ciphers

Block ciphers are the class of symmetric-key encryption algorithms that transform $n$-bit plaintext blocks to $n$-bit ciphertext blocks with a user-provided secret $k$-bit key $K$. Decryption is performed by applying the reverse transformation to the ciphertext block using the same secret key.

Each key identifies one permutation from the possible set of $2^n!$. To provide a unique decryption, the encryption function must be a bijective mapping. For each key $K$, since each plaintext block is mapped into different a ciphertext block, $E_K$ is a permutation over the set of input blocks. However, when the value of plaintext is fixed, the block cipher behaves like a random one-way function that takes key as input.

TMTO attacks for block ciphers is a chosen plaintext attack. Given a plaintext and ciphertext pair $(P_0, C_0)$, the aim is to find the secret $k \in N$. The attack consists of two phases. In the offline phase, $P_0$ is encrypted using various key values and a table summarizing the calculations is stored in the memory. Firstly, $m$ random plaintext values are selected, and then they are encrypted and reduced to create another key value. This is repeated $t$ times and $m$ chains are produced. The first and last elements of each chain is stored. This process is represented in the Figure 5.1.

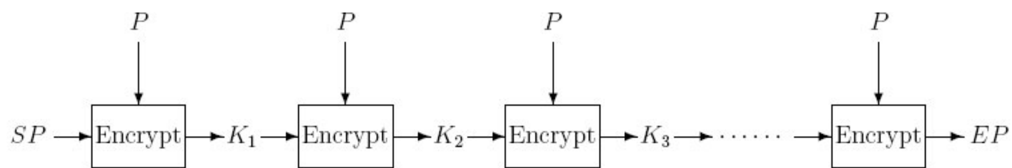


Figure 5.1: Generation of a chain in block ciphers

In the online phase, the aim is to figure out if the key to generate $C_0$ is one of the keys used in the offline phase. Since only the first and last values of the table is stored, a similar chain is produced by applying reduction to $C_0$. After each reduction and encryption, the obtained value is compared to the last values in the table. If a match is obtained, then the whole chain is regenerated and the key just before reduction of $C_0$ is the secret key. The success probability of the attack is closely related to the percentage of the keys that are covered in the offline phase.

For block ciphers with large keys, this attack is not considered to be a serious threat since the precomputation requirement is about exhaustive search complexity. Another disadvantage is that the attack does not benefit whenever more plaintext and ciphertext pairs are available.

Apart from Hellman's work on DES in 1980 [11], the first improvement was Rivest's distinguished points [7], which reduces the memory accesses. In [13] Kusuda and Matsumoto optimize a relation among the breaking cost, time and success probability and in [12] they show that the table parameters can be adjusted to achieve higher success probability. Efficient mask functions in terms of hardware cost and probability of success are proposed and experimentally confirmed and used to build an FPGA design to perform realistic tradeoffs against the block cipher DES. It is reported that the resulting online attack recovered 40 bit of key in about 10 seconds. In 2003, Oechslin [14] proposed a new way of precomputation called Rainbow tables which reduces the online time complexity. To demonstrate this, he implemented an attack on Windows password hashes as. In [10], it has been shown how to apply nontrivial multiple data TMTO to both CBC and CFB modes of operations.

## 5.2 Stream Ciphers

Stream ciphers constitute the other important class of symmetric encryption algorithms. The basic design philosophy of stream ciphers was inspired by the perfectly secure One Time Pad cipher invented by Vernam in 1917. Stream ciphers generate keystream independent of plaintext using following equations;

$$
\begin{aligned}
S_0 &= f_{init}(K, IV), \\
S_{t+1} &= f(S_t), \\
z_t &= g(S_t), \\
c_t &= h(z_t, p_t),
\end{aligned}
$$

where $S_t$ is the *internal state* at time $t$, $f_{init}$ is the *initialization function* that uses secret key $K$ and public *initialization vector (IV)*, $f$ is the *next state function* that only depends on the current state, $g$ is the filter function that produces keystream $z_i$ and $h$ is the encryption function that combines *plaintext* $p_t$ to produce *ciphertext* $c_t$.

For a stream cipher with $k$ bit key and $v$ bit IV, keystream of length $k+v$ bit keystream is generated. This mapping is usually close to a bijection. Using the data of $D = 2^{\frac{1}{4}(k+v)}$ frames, it is possible to recover a single key and IV pair within time and memory complexity $T = M = 2^{\frac{1}{2}(k+v)}$. For ciphers with $v < k$, the complexity is less than exhaustive key search. Therefore, if the size of IV is less than key size, the cipher is vulnerable to TMTO [10]. Also, IVs should not be predicted.

Stream ciphers behave like random one-way functions, whenever we try to obtain key using the keystream. TMTO on stream ciphers was first proposed by Babbage [2] and Golic [9] through independent works. In[2], Babbage suggested as a design principle for stream cipher that a state size of $2i$ bits is desirable for a secret key length of $i$ bits. Golic applied attack on A5/1( an encryption algorithm used in GSM standard). Biryukov Shamir and Wagner [4] improved the attack in which key is computed in about one second during the first two minutes of the conversation on a single PC. In [15], a tradeoff attack against LILI-128 is proposed using $2^{64}$ bits of keystream, a lookup table of $2^{45}$ 89-bit words and $2^{48}$ DES operations.

## 5.3 Hash Functions

A hash function $H$ maps an input of arbitrary length to an output $H(x)$ with fixed length.

For a hash function to be cryptographically secure, it should satisfy three properties: (i) one-wayness, (ii) second preimage resistancy, and (iii) collision resistancy. A hash function $H$ is said to be one-way, if it is hard to find an input $x$ given $H(x)$. For second preimage resistant, given $x_1$, it should be hard to find $x_2$ with the same hash value. The strongest property is the collision resistance which is satisfied if it is hard to find $x_1$ and $x_2$ with the same hash value.

Hash functions are typical one-way functions where the TMTO method can be applied. Although input space of hash functions is unlimited, the attack can be implemented so that the input space is restricted to some character set with short lengths. Considering the practical usage of hash functions such as password storage and digital signatures, it is often the case that input space can be taken as human readable characters, which may render the attack more successful than expected. The most common scenario of TMTO method usage is the breaking of authentication mechanisms with passwords. In order to resist TMTO attacks, the input of hash functions should be forced to comply with some rules, such as the input length. Salting method can also be used to increase the input space.

## 5.4 An Application of TMTO to SHA-1

Secure Hash Algorithm (SHA) was published by NIST (National Institute of Standards and Technology) as a U.S. government standard. SHA is a hash

| | $m$ | $t$ | $k$ | $P$ | $M$ | $T$ |
|---|---|---|---|---|---|---|
| Setting 1 | $10,321$ | $10,321$ | $10,322$ | $\approx 2^{40}$ | $\approx 2^{26.66}$ | $\approx 2^{26.66}$ |
| Setting 2 | $5,160$ | $20,642$ | $10,322$ | $\approx 2^{40}$ | $\approx 2^{25.66}$ | $\approx 2^{27.66}$ |
| Setting 3 | $20,642$ | $5,160$ | $10,322$ | $\approx 2^{40}$ | $\approx 2^{27.66}$ | $\approx 2^{25.66}$ |

Table 5.1: TMTO parameters for the SHA-1 application.

function family consisting of five algorithms; SHA- 1, SHA-224, SHA-256, SHA-384, and SHA-512. SHA-1 is widely used in various applications such as TLS and SSL, PGP, SSH, S/MIME, and IPsec.

We have implemented our attack on SHA-1 as an example of the TMTO application. Our input domain is chosen as 29 letters of Turkish alphabet plus the characters $q$, $w$, $x$. The input length is 8 characters. In this case the domain size becomes $N = 32^8 = 2^{40}$. The method is implemented with classical Hellman tables. In order to show the effect of time-memory trade-off, we applied the attack with three different set of parameters as shown in Table 5.1. In the table, $m$, $t$ correspond to the width and height of a single Hellman table, respectively. $k$ is the number of tables. $P$ is the product of these three quantities which is also the offline complexity. $M$ is the memory complexity and $T$ is the online time complexity of the attack.

The following figure describes a single iteration of the one-way function chosen from SHA-1.

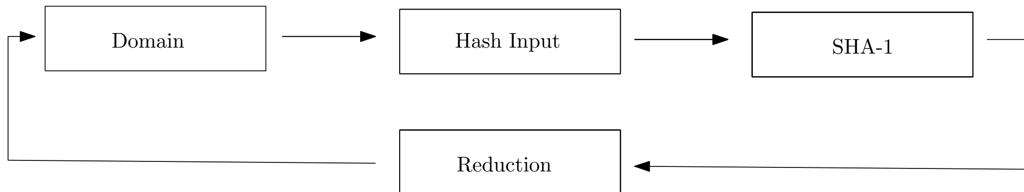


Figure 5.2: SHA-1 application, one-way and reduction function.

Since there are $2^{40}$ possible input values, we choose the domain as 40-bit

vectors starting from 0 to $2^{40} - 1$. Afterwards we map each vector to a 8-character input value. We do this by splitting 40-bit domain vector into eight 5-bit vectors. Each 5-bit vector represents a character from our character set of size 32. Once the 8-character input is constructed, it is hashed with the SHA-1 algorithm. The reduction function takes the first 40-bits which enters the next iteration as domain value.

There are $10,322$ tables for each setting, so we must choose this number of different reduction functions. We just implemented the reduction function of each table as the regular 40-bit reduction function XOR'ed with the table's index value. That is, the first table's reduction function just truncates the SHA-1 hash output to 40-bits and the second table's reduction function XOR's this 40-bit with the binary vector 0x0000000001, and so on.

The precomputation for each setting requires $\approx 2^{40}$ evaluations of the SHA-1 algorithm. We performed the offline computations on multiple PC's with cpu speed of 3.00GHz and memory capacities ranging from 1.0GB to 2.0GB. The precomputation for each setting took approximately 20 days of cpu time on one computer.

The results of the precomputation are stored on files, each containing the start-end point tuples of the associated table, and are sorted with respect to the end points. The sorting is essential in order to make a fast binary search in the online phase. The following table shows the storage amounts for the three settings.

In the online phase we chose 100 random input values for each of the settings from our domain set and calculated their SHA-1 hash values. The hash values are given as input to the online search algorithm. The computations are carried

|  | $m$ | $k$ | Storage Per Table | Total Storage |
|---|---|---|---|---|
| Setting 1 | $10,321$ | $10,322$ | $103,210$ bytes | ≈1,016 MB |
| Setting 2 | $5,160$ | $10,322$ | $51,600$ bytes | ≈508 MB |
| Setting 3 | $20,642$ | $10,322$ | $206,420$ bytes | ≈2,032 MB |

Table 5.2: Store requirements for the SHA-1 application.

|  | $m$ | $t$ | Success Rate | Search Time | False Alarms |
|---|---|---|---|---|---|
| Setting 1 | $10,321$ | $10,321$ | 60% | 229 sec | $5,171$ |
| Setting 2 | $5,160$ | $20,642$ | 55% | 470 sec | $10,310$ |
| Setting 3 | $20,642$ | $5,160$ | 60% | 105 sec | $2,251$ |

Table 5.3: Online phase results for the SHA-1 application.

out on a PC with a cpu speed of 3.0GHz and a memory capacity of 2.0GB.

**Interpretation of the Results**

The results are presented in Table 5.3, where success rate is the ratio of the found preimages among 100 random inputs, search time is the average time in order to search all the tables (including false alarm calculations), in other words, the average time passed when a preimage is not found, and false alarm is the number of average false alarms occurred in a full table search.

From the Table 5.3., it can be clearly seen that how a trade-off between the time and memory has been made. If an exhaustive search was to be carried out, the search operation would last $\approx$ 20 days. If on the other hand, a lookup table had been created, it would have cost $5.2^{40}$ bytes = 5 Terabytes (TB) of storage. Both of these time and memory complexities required by exhaustive search and table lookup are not feasible if the search operation will be performed many times.

In Table 5.4., a comparison of the implemented TMTO application with

| | Memory | Time |
|---|---|---|
| Ex. Search | - | $\approx$ 20 days |
| Table Lookup | 5 TB | - |
| Setting 1 | $\approx$ 1.0 GB | 229 sec |
| Setting 2 | $\approx$ 0.5 GB | 470 sec |
| Setting 3 | $\approx$ 2.0 GB | 105 sec |

Table 5.4: Comparison of the application results with Exhaustive Search and Table Lookup methods.

exhaustive search and table lookup methods is given. Compared to exhaustive search, we can decrease the search time from 20 days to a couple of minutes by using a memory around $1GB$, and compared to table lookup method, we can decrease the required memory from $5TB$ to the order of Gigabytes in the expense of increasing the search time from a lookup operation to a few minutes.

# Chapter 6

# Conclusion

One-way functions play an important role in the security of various encryption schemes, authentication mechanisms and other cryptographic protocols. For most of the cryptographic systems, inverting these functions results in breaking the system.

TMTO is a generic method to find inverses of one-way functions. In this thesis, we studied the TMTO method and showed how it can be applied to symmetric-key cryptosystems and hash functions as examples.

Our contribution is an explicit asymptotic expression for the coverage of a singe Hellman table, which has not been stated thus far. This expression not only allows us to understand the behaviour of the success rate of the classical method proposed by Hellman [11], but also helps us to choose appropriate time and memory complexities for a fixed total complexity value and a fixed success rate.

We believe that the method does not pose a threat for block ciphers with long key sizes, since it requires a precomputation complexity equal to the exhaustive search. However, it is possible to mount the attack for smaller key spaces consisting of restricted character sets of shorter lengths. Another reason why the TMTO method is less effective in block ciphers is that the precomputation is done for only one chosen plaintext: hence multiple plaintext-ciphertext pairs

do not improve the success rate of the method.

Stream ciphers are more vulnerable to the TMTO method, both because there are more ways to choose the one-way function (input space can be key or cipher state), and because the availability of more keystream data increases the success rate.

Hash functions are typical one-way functions where the TMTO method can be applied. Although input space of hash functions is unlimited, the attack can be implemented so that the input space is a restricted character set with short lengths. Considering the practical usage of hash functions such as password storage and digital signatures, it is often the case that input space can be taken as human readable characters, which may render the attack more successful than expected. The most common scenario of TMTO method usage is the breaking of authentication mechanisms with passwords. In order to resist TMTO attacks, the input of hash functions should be forced to comply with some rules, such as the input length. Salting method can also be used to increase the input space.

TMTO attacks must be taken into consideration by the designers of not the systems mentioned above, but all the systems where it is applicable.

Some future studies and open problems about the TMTO can be listed as follows:

- Improving available methods so that higher success rates are obtained. Although there is a theoretical limit $\approx (1 - e^{-1})$ of success rate for a precomputation complexity equivalent to exhaustive search, the online complexity may be improved such as in Oecshlin's rainbow tables method [14].
- Another improvement we plan to work on, as a part of a future study, is

the cases where higher success rates are needed. In such cases, one has to make a precomputation of complexity equivalent to multiple times of an exhaustive search. This naturally increases the time complexity and the memory complexity in the order of precomputation effort. We believe that an algorithm which provides a higher success rate with a precomputation complexity higher than exhaustive search, while keeping the time and memory complexities closer to the usual TMTO is worth investigating.

- There are also open problems related to the distinguished points method suggested by Rivest [7]. These problems are stated in [5].

# References

[1] Avoine G., Junod P., Oechslin P., *Time-Memory Trade-Offs: False Alarm Detection Using Checkpoints*, In Progress in Cryptology - INDOCRYPT'05, Volume 3797 of Lecture Notes in Computer Science, pp. 183-196, 2005.

[2] Babbage S., *Improved "Exhaustive Search" Attacks On Stream Ciphers*, European Convention on Security and Detection, IEEE Conference Publication No. 408, May 1995.

[3] Biryukov A., Shamir A., *Cryptanalytic Time/Memory/Data tradeoffs for Stream Ciphers* In Proceedings of Asiacrypt'00 (T. Okamoto, ed.), volume 1976 in Lecture Notes in Computer Science, pp. 1-13, Springer-Verlag, 2000.

[4] Biryukov A., Shamir A., Wagner D., *Real Time Cryptanalysis of A5/1 On A PC, In Fast Software Encryption - FSE'00*, Volume 1978 of Lecture Notes in Computer Science, pp. 1-18, New York, USA, April 2000.

[5] Borst J., *Block Ciphers: Design, Analysis and Side-Channel Analysis*, Chapter 3: A Time-Memory Tradeoff Attack, Phd Thesis, Department of Electrical Engineering, Katholieke Universiteit Leuven, September 2001.

[6] Çalık Ç., Sulak F., Doğanaksoy A., *Observations on Hellman's Cryptanalytic Time-Memory Trade-off*, 2nd National Cryptology Symposium, Ankara, TURKEY, 2006.

[7] Denning D.E., *Cryptography and Data Security*, page 100. Addison-Wesley, Boston, Massachusetts, USA, June 1982.

[8] Flajolet P., Odlyzko A.M., *Random Mapping Statistics*, EUROCRYPT'89, volume 434 of Lecture Notes in Computer Science, pp. 329-254, Springer-Verlag, 1990.

[9] Golic Jovan Dj., *Cryptanalysis of Alleged A5 Stream Cipher.*, EUROCRYPT'97, pp.239-255, 1997.

[10] Hong J., Sarkar P., *Rediscovery of Time Memory Tradeoffs*, Cryptology ePrint Archive, Report 2005/090, 2005.

[11] Hellman M.E., *A Cryptanalytic Time-Memory Trade-Off*, IEEE Transactions On Information Theory, Vol. IT-26, pp. 401-406, July 1980.

[12] Kim I., Matsumoto T., *Achieving Higher Success Probability in Time-Memory Trade-Off Cryptanalysis without Increasing Memory Size* IEICE Transactions on Communications/Electronics/Information and Systems, Vol.E82-A, No.1, pp. 123-129, January 1999.

[13] Kusuda K., Matsumoto T., *Optimization of Time-Memory Trade-Off Crypyanalysis and Its Application to DES, FEAL-32, and Skipjack* IEICE Transactions on Fundamentals, Vol.E79-A, No.1, pp. 35-48, January 1996.

[14] Oechslin P., *Making a Faster Cryptanalytic Time-Memory Trade-Off* In Advances in Cryptology - CRYPTO'03 (D. Boneh, ed.), volume 2729 of Lecture Notes in Computer Science, pp. 617-630, Springer-Verlag, 2003.

[15] Saarinen M.-J.O., *A Time-Memory Attack Against LILI-128*, In Proceedings of Fast Software Encryption - FSE'02 (J. Daemen and V. Rijmen, eds.), volume 2365 of Lecture Notes in Computer Science, pp. 231-236, Springer-Verlag, 2002.

[16] Standaert F.X., Rouvroy G., Quisquater J.J., Legat J.D., *A Cryptanalytic Time-Memory Tradeoff: First FPGA Implementation*, Field-Programmable Logic and Applications - FPL'02, volume 2438 of Lecture Notes in Computer Science, pp. 780-789, 2002.

[17] Standaert F.X., Rouvroy G., Quisquater J.J., Legat J.D., *A Time-Memory Trade-Off Using Distinguished Points: New Analysis & FPGA Results* Workshop on Cryptographic Hardware and Embedded Systems - CHES'02, volume 2523 of Lecture Notes in Computer Science, pp. 593-609, Redwood Shores, California, USA, August 2002.

[18] Quisquater J.J., Delescaille J.P., *How easy is collision search? New results and applications to DES*, In Proceedings of CRYPTO'89, volume 435 of Lecture Notes in Computer Science, pp. 408-413, Santa Barbara, California, USA, August 1989.