Cryptanalytic Time-Memory Tradeoffs on COPACOBANA

Tim Güneysu, Andy Rupp and Stefan Spitz

Horst-Görtz Institute for IT-Security, Ruhr-University Bochum, Germany.
{gueneysu, arupp}@crypto.rub.de, stefan.spitz@rub.de

Abstract: This paper presents our ongoing work on the analysis and optimization of cryptanalytic time-memory tradeoffs targeting the COPACOBANA architecture [KPP$^+$06] as platform for the precomputation/online phase.

1 Introduction

A common computational problem frequently appearing in cryptanalysis can be generally described as follows: Let $f : S \to D$ be a one-way function with a domain S of size $|S| = N$. Given an image $y \in D$, the challenge is to find a preimage of y, i.e., some element $x \in S$ s.t. $f(x) = y$. For instance, in a known-plaintext attack on a cipher E one tries to invert the function $f : x \mapsto E_x(P)$, where P is the *fixed* known-plaintext and x is a key, for a given ciphertext $f(k)$.

There are two naive approaches to solve instances of this problem: performing an exhaustive search for each wanted k or precomputing and accessing an exhaustive table containing all $(x, f(x))$ pairs. However, in practice these solutions are unsatisfying since they require an infeasible amount of time respectively disk space. By using a cryptanalytic time-memory tradeoff (TMTO) method, one tries to find a compromise between reasonably reducing the actual search complexity (by doing some kind of precomputation) and keeping the amount of precomputed data reasonably low. What “reasonably” means here depends on the concrete attack scenario (e.g., real-time attack), the function f and the available resources for the precomputation and online phase.

Existing TMTO methods [Hel80, D. 82, Oec03] share the natural property that the success probability of the online phase, i.e., the probability that the wanted preimage is actually covered by the precomputed data, depends on the complexity of the precomputation phase. As a consequence, to achieve a success rate that is significant in practice a lot of time must be spent on the precomputation phase (typically in the order of N). However, providing a sufficiently large cluster of PCs for performing this task is usually too costly or difficult. In fact, to the best of our knowledge nobody has reported a completed precomputation for a full 56-bit DES TMTO attack so far.

Recently, the benefits of special-purpose hardware in terms of Field Programmable Gate Arrays (FPGAs) for doing the time-consuming TMTO precomputations have been discovered. In [SRQL02] an FPGA design for a TMTO attack on a 40-bit DES variant using

Rivest's approach was proposed. In [MBPV06] a hardware architecture for UNIX password cracking based on Oechslin's method was presented.

The paper at hand describes our ongoing research project dealing with the realization of TMTO attacks on the COPACOBANA (Cost-Optimized Parallel Code Breaker) hardware architecture [KPP$^+$06]. In the course of this project, the suitability of COPACOBANA for implementing the precomputation/online phase of different TMTO methods is analyzed. Moreover, we optimize the various TMTO parameter choices with respect to different attack configurations. As a practical result, we like to provide precomputation data for the first TMTO attack on full 56-bit DES.

2 Time-Memory Tradeoff Methods in Cryptanalysis

In this section we sketch Hellman's original TMTO method as well as the the variants proposed by Rivest and Oechslin. For concreteness, the methods are considered in the case of a block cipher E given a fixed known plaintext P, i.e., $f(x) = E_x(P)$.

2.1 Hellman's Original Approach

In Hellman's TMTO attack, published in 1980 [Hel80], one tries to precompute all possible $(x, f(x))$ pairs in advance by encrypting P with all N possible keys. However, to reduce memory requirements these pairs are organized in several *chains* of a fixed length. The chains are generated deterministically and are uniquely identified by their respective start and end points. In this way, it suffices to save its start and end point to restore a chain later on. In the online phase of the attack, one then simply needs to identify and reconstruct the right chain containing the given ciphertext to get the wanted key. The details of the two phases are described in the following.

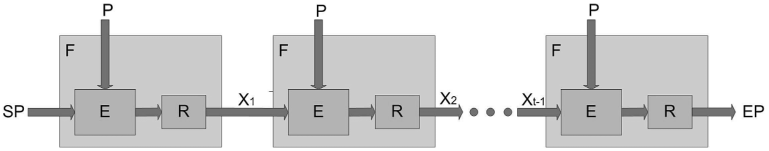

Figure 1: Chain generation due to Hellman

Precomputation phase. In this phase, first m different keys are chosen serving as start points SP of the chains. To generate a chain, one first computes $E_{SP}(P)$ resulting in some ciphertext C (see Figure 1). In order to continue the chain, C is used to generate a new key. To this end, a so-called *reduction function* R is applied reducing the bit length of C to the bit length of a key for the cipher E (if necessary) and performing a re-randomization of the output. By means of R we can continue the chain by computing $R(E_{SP}(P)) = X_1$, using the resulting key X_1 to compute $R(E_{X_1}(P)) = X_2$ and so on. The composition of

E and R is called *step-function* F. After t applications of F the chain computation stops and we take the last output as the end point EP of the chain. The pair (SP, EP) is stored in a table sorted by the end points. The number of distinct keys contained in a table divided by N is called the *coverage* of a table. Unfortunately, the occurence of a key in a table is not necessarily unique because (due to the nature of R) there is a chance that two chains collide and merge or that a chain runs in a loop. Since the probability of merges increases with the size of a table, at a certain point we cannot significantly improve the coverage by simply adding more and more chains. To cope with this problem, Hellman suggested to generate multiple tables each associated with a different reduction function.

Online phase. To find out if a given ciphertext C was generated using a key k that is covered by a particular table, we compute all chains starting from $R(C)$ up to a length of t and compare their end points with the end points in the table. More precisely, we first check if $R(C)$ is contained. If not we compute $F(R(C))$ and look for a match, then we do this for $F(F(R(C)))$ and so on. If a match occurs after the i-th application of F for a pair (SP, EP), then $F^{t-i-1}(SP) = X_{t-i-1}$ is a key candidate which needs to be verified. If $E_{X_{t-i-1}}(P) = C$ we are successful and the online phase ends. If no valid key is found in the current table, we repeat above procedure for another table (and another R and F).

2.2 Variants of Hellman's Approach

Distinguished Points. In practice, the time required to complete the online phase of Hellman's TMTO is dominated by the high number of table accesses. The distinguished point (DP) method, introduced by Rivest [D. 82] in 1982, addresses this problem. A DP is a key that fulfills a certain simple criterion (e.g., the first 20 bits are 0) which is usually given as a mask of length dp_l. Rivest's idea was to admit only DPs as end points of a chain. For the precomputation phase this means that a chain is computed until a DP or a maximal chain length $t_{max} + 1$ is reached. Only chains of length $\ell \leq t_{max} + 1$ ending in a DP are stored. Using DPs also merging and looping chains can be detected and are discarded. In the online phase, the table does not need to be accessed after every application of F but only for the first occuring DP. If we have no match for this DP we can proceed with the next table.

Rainbow Tables. Rainbow tables were introduced by Oechslin [Oec03] in 2003. He suggested not to use the same R when generating a chain for a single table but a (fixed) sequence $R_1, \ldots, R_t$ of different reduction functions. More precisely, due to the different reduction functions we get t different step-functions $F_1, \ldots, F_t$ that are applied one after another in order to create a chain. This modification reduces the number of merges considerably while loops are prevented completely. Hence, regarding a space efficient coverage, these characteristics allows to put much more chains into a rainbow table than into a Hellman table. This in turn significantly reduces the total number of tables needed in order to achieve a certain coverage. Since fewer rainbow tables must be searched in the online phase (what is however a bit more complex) a lower number of calculations and table accesses is required. To lookup a key in a rainbow table, we first compute $R_t(C)$ and compare it to the end points, then we do this for $F_t(R_{t-1}(C))$, $F_t(F_{t-1}(R_{t-2}(C)))$, etc.

3 Using COPACOBANA for TMTOs on DES

The COPACOBANA [KPP+06] is an existing FPGA cluster designed for parallel applications with a focus on massive computations and a minor demand on communication and memory. It integrates a total number of 120 Xilinx Spartan-3 XC3S1000 FPGAs in a modular design. In a DES brute force attack configuration, the machine can compute $2^{35.9}$ DES-encryptions per second resulting in an average key search duration of 6.4 days.

In this section, we will employ COPACOBANA for accelerating the precompution and online phase of a TMTO attack on DES. In such a scenario, primarily the hardware limitations of COPACOBANA need to be considered. Since COPACOBANA does not allow for installation of directly attached storage, all TMTO tables must be managed by a connected host computer. The recent interface of COPACOBANA to the host provides a communication bitrate of $24 \cdot 10^6 \approx 2^{24.5}$ bit per second. Compared to the number of possible DES encryptions per second, the bottleneck of the COPACOBANA is the data throughput for transferring (SP, EP) tuples to the host. To address the constraint of limited bandwidth, a minimum number of $2^{11.4} \cdot b$ computations must be performed until a data transfer is initiated, where b denotes the bitlength of a tuple (SP, EP). For practical reasons, we have fixed the memory usage to a maximum of two terabyte and the required success rate to 80%. Based on experiments, we found following parameters for chain length, number of tables and start points to satisfy the given constraints.

Method	Chain Length	# Start Points	# Tables	Bits p. table entry
Hellman	$t = 2^{19.2}$	$2^{16.7}$	2^{21}	73
DP	$t_{min} = 2^{18}, t_{max} = 2^{20}, dp_l = 19$	2^{18}	2^{21}	55
Rainbow	$t = 2^{19.5}$	2^{35}	5	91

To optimize the bandwidth usage, we use the first m integers as start points SP so that each SP can be stored with only $log_2(m)$ bits. For distinguished points, the minimum chain length t_{min} is necessary to ensure that the generated data traffic from tuples (SP, EP) complies with the available bandwidth on the COPACOBANA. More precisely, each DP chain leading to a total chain length less than $t_{min} + 1$ is discarded and not transfered to the host. Furthermore, the storage of end points for DP can be limited to the remaining $56 - dp_l$ bits not covered by the DP-property. The following table presents our worst case expectations concerning success rate, memory usage, the duration of the precomputation phase on COPACOBANA as well as the number of table accesses (TA) and calculations (C) during the online phase. Note that these figures for use with COPACOBANA are based on estimations given in [Hel80, Oec03, SRQL02] (false alarms are neglected).

Method	Success Rate	Memory Usage	Precomputation	Online Phase
Hellman	0.80	1897 GB	24 days	$2^{40.2}$ TA + $2^{40.2}$ C
DP	0.80	1690 GB	95 days	2^{21} TA + $2^{39.7}$ C
Rainbow	0.80	1820 GB	23 days	$2^{21.8}$ TA + $2^{40.3}$ C

According to our findings, precomputations for distinguished points on a single COPACOBANA take roughly four times longer compared to Hellman's and rainbow methods based on given constraints. Contrary, the subsequent online attack has the lowest complexity for the distinguished point method. Considering a TMTO scenario to use the COPACOBANA for precomputation only (implying that the online attack is performed by a

PC), the rainbow table method provides best performance. When using COPACOBANA as well for precomputation *and* online phase, there is a strong indicator to select distinguished points as the method of choice. For distinguished points, we can assume the frequency of table accesses to follow a uniform distribution, hence, we expect balanced bandwidth requirements over time. With rainbow tables, the online attacks starts with very short but incrementing computation trails. This results in significant congestion on the COPACOBANA's communication interface since a large number of table lookups are required in the beginning of the online phase. Therefore, a scenario running both precomputations and the online attack on the COPACOBANA, should be based on the distinguished points method since this method is most promising with respect to the restrictions of the machine.

4 Future Work

Our goal is to realize a DES-TMTO attack providing a good success rate ($\geq 80\%$), a reasonable memory usage ($\approx$ 2 TB) and an online phase that can be executed *within a few minutes*. Considering only the number of calculations required for this phase which is about 2^{40} (see Section 3), we conclude that a hardware implementation of the online phase (e.g., on COPACABANA), is mandatory since a contemporary PC only achieves about 2^{21} DES encryptions per second. However, even assuming such a fast hardware implementation the high number of table accesses (about 2^{21}, where one access takes about 4 ms) still prevent us from achieving this goal using existing TMTO methods. To this end, we are working on a new TMTO variant that is suitable for COPACOBANA and allows to further reduce the number of table accesses. The new scheme, called distinguished rainbow points (DRP), is essentially a slightly modified version of Rivest's DP approach combined with Oechslin's rainbow tables. Preliminary analyses show that its online time should be within minutes including table accesses while the precomputation time is even lower than for Rivest's scheme.

References

[D. 82] D. Denning. Cryptography and Data Security, p.100. Addison-Wesley, 1982.

[Hel80] M. E. Hellman. A Cryptanalytic Time-Memory Trade-Off. In *IEEE Transactions on Information Theory*, volume 26, pages 401–406, 1980.

[KPP$^+$06] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler. Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker. In *Proc. of CHES 2006*, volume 4249, pages 101–118. Springer-Verlag, LNCS, 2006.

[MBPV06] N. Mentens, L. Batina, B. Prenel, and I. Verbauwhede. Time-Memory Trade-Off Attack on FPGA Platforms: UNIX Password Cracking. In *Proc. of ARC 2006*, volume 3985 of *LNCS*, pages 323–334. Springer-Verlag, 2006.

[Oec03] P. Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. In *Proc. of CRYPTO 2003*, volume 2729 of *LNCS*, pages 617–630. Springer-Verlag, 2003.

[SRQL02] F. Standaert, G. Rouvroy, J. Quisquater, and J. Legat. A Time-Memory Tradeoff using Distinguished Points: New Analysis & FPGA Results. In *Proc. of CHES 2002*, volume 2523 of *LNCS*, pages 596–611. Springer-Verlag, 2002.