A review of time-memory trade-off techniques

Liu Teng[1,2], Li Zelong[1]

[1]School of Information Science and Engineering, University of Jinan, Shandong Province, China

[2]202121200914@stu.ujn.edu.cn

Abstract. This paper delves into the discussion of time-memory trade-off techniques in the field of cryptanalysis. The method was initially introduced by Hellman in 1980, subsequently, DP trade-off, rainbow trade-off, and checkpoint trade-off have been proposed to enhance the efficiency of cryptographic attacks. This paper elaborates on the concept of rainbow trade-off and their variants and presents optimizations in terms of storage and runtime speed for time-memory trade--off methods. Ingenious storage optimization significantly reduces the storage overhead of pre-computed tables, and the rapid advancement of implementation platforms achieves speed optimization for the online phase. Through these optimization measures, time-memory trade-off methods exhibit even more remarkable performance in practical applications. For researchers and practitioners in the field of cryptography, the content of this paper provides valuable references and insights for their work.

Keywords: time-memory trade-off, cryptanalysis, rainbow trade-off, storage optimization.

1. Introduction

In today's context, many systems require password authentication to access a significant portion of their internal content. These systems typically rely on password-hashing techniques. This technology involves subjecting passwords to one-way function calculations. The definition [1] is that given an input, computing its function value is straightforward, but computing the input from a function value is challenging. In such systems, the hashed values of passwords are commonly stored. When a user logs in, the newly inputted password undergoes a one-way function calculation, which is then compared to the stored hashed value within the system.

Any cryptographic analysis problem can be understood as the process of reversing a one-way function, which involves finding the input to a given one-way function. Let f: $X \rightarrow Y$ be an arbitrary one-way function. In conventional approaches, there are two methods to reverse a one-way function. Given target $y=f(x) \in Y$, we can exhaustively try all possible values of x until we find $x' \in X$ that satisfies $f(x')=y$, referred to as an exhaustive method. The second method involves precomputing (x, f(x)) pairs and storing them in a table. Given a target x, the corresponding f(x) can be directly looked up in the table, known as the look-up table method. Time-memory trade-off methods lie between the exhaustive method and the look-up table method. They offer shorter search time compared to the exhaustive method and occupy less storage space than the look-up table.

In summary, this paper explores the advancements in cryptographic attacks, particularly focusing on time-memory trade-off methods. The research aims to comprehensively delve into the underlying concepts of these methods, providing a concise overview and discussing optimization strategies. This study contributes to a deeper understanding of these techniques, enabling more effective solutions to cryptographic analysis challenges and enhancing support for system security strategies.

2. Time-Memory Trade-Off

2.1. Relevant Concepts

2.1.1. Reduction Function

Assuming F is a one-way function from space N to space H, and R is a reduction function from space H to space N, then we have G(x)=R(F(x)), where G is a one-way function from space N to N. The role of a reduction function [2] in time-memory trade-off algorithms is to map a value from one space to another. Typically, it is implemented using the modulo operation: R(y)=y mod N.

2.1.2. Two Phases

Time-memory trade-off algorithms consist of two phases [3]: the pre-computation phase (offline phase) and the online phase. The pre-computation phase involves generating the precomputation table by creating precomputation chains according to specific rules. The starting and ending points of these precomputation chains are saved to form the precomputation table. The online phase is the actual attack phase, which includes three components: generating precomputation chains, collision detection, and precomputation chain reconstruction.

2.1.3. False Alarm

After finding a matching endpoint during the online phase, the reconstruction of the precomputed chain needs to start from the beginning. If the currently generated chain merges with a chain that does not contain the key K, there is a chance that after the chain reconstruction, the key K might not be found. This situation is referred to as a false alarm [4].

2.2. Hellman Trade-Off

In 1980, Hellman introduced a classic time-memory trade-off method [5] for attacking the Data Encryption Standard (DES) algorithm, a symmetric-key block cipher. The Hellman trade-off involves a pre-computation phase wherein m distinct starting points are chosen from the key space, denoted as S_1, S_2, S_3, …, S_m. These starting points undergo t rounds of the DES's G function calculations, resulting in final values known as endpoint values: E_1, E_2, E_3, ..., E_m. The mt nodes in the computation paths form a Hellman matrix. The key-value pairs of the starting and ending points are preserved as entries in a precomputed table, referred to as the Hellman table. Figure 2-(a) depicts the structure of the Hellman matrix generation during the precomputation phase.

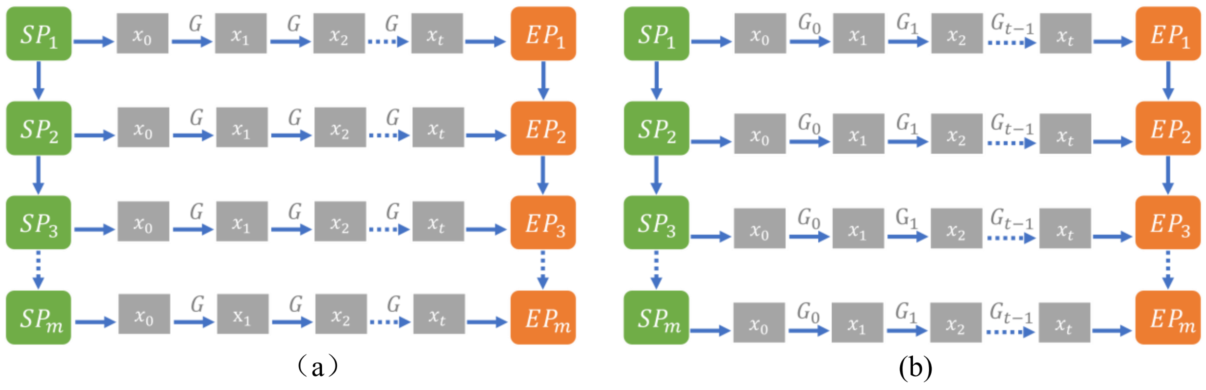
SP_1 x_0 G x_1 G x_2 G x_t EP_1
SP_2 EP_2
SP_3 EP_3
SP_m EP_m
(a)
SP_1 x_0 G_0 x_1 G_1 x_2 G_{t-1} x_t EP_1
(b)

Figure 2. Schematic of the Hellman Matrix and Rainbow Matrix.

Within the framework of the Hellman trade-off, we assume the presence of m table entries in each table, with each pre-computed chain undergoing *t* iterations. We offer the probability of effectively locating the desired key within a specified table:

$$P_{single} \geq \frac{1}{\mathrm{N}} \sum_{i=1}^{m} \sum_{j=0}^{t-1} \left(1-\frac{it}{\mathrm{N}}\right)^{j+1} \tag{1}$$

Increasing the success rate of attacks using the Hellman trade-off usually entails the utilization of distinct G functions to produce multiple Hellman tables. Assuming the existence of *l* tables, the probability of successfully determining the target key is as follows:

$$P_{success} \geq 1-\left(1-\frac{1}{\mathrm{N}} \sum_{i=1}^{m} \sum_{j=0}^{t-1} \left(1-\frac{it}{\mathrm{N}}\right)^{j+1}\right)^{l} \tag{2}$$

The total storage space required for the precomputation phase of the Hellman trade-off is M=*ml*. In the online phase, disregarding the iterations for handling false positives in each table, each table requires *t* iterations. Thus, the time complexity of the online phase can be represented as T=*tl*. Hellman recommended setting $m=t=l=\mathrm{N}^{1/3}$, resulting in a balanced trade-off curve with the equation $\mathrm{TM}^2=\mathrm{N}^2$.

2.3. DP Trade-Off

To address the issue of frequent false positives in the Hellman trade-off and reduce the number of table lookups during the online phase, Rivest introduced the Distinguishing Point (DP) method [6] in 1982. The DP trade-off involves introducing a discernible attribute into the key space with a fixed probability of 1/*t*. Typically, this attribute is easy to detect, such as setting the first log*t* bits of an element to 0.

Due to the requirement of satisfying the DP attribute, there's a possibility of encountering cycles in the process of generating a precomputed chain from a starting point S_i. Consequently, a chain length restriction must be imposed. Precomputed chains that exceed this restriction are discarded, and new chains are generated.

Precomputation phase, the average length of DP chains is *t*. Consequently, the total number of nodes covered by the m DP chains is *mt*. Online phase, an average of t iterations is required to locate a DP point. Thus, the time complexity T=*tl*. Combining this with the size of the precomputed table M=*ml*, it can be deduced that the trade-off curve for the DP trade-off follows $\mathrm{TM}^2=\mathrm{N}^2$.

2.4. Rainbow Trade-Off

The DP trade-off significantly reduces the number of table lookups during the online phase. However, due to its use of variable chain lengths, it has some inherent flaws when it comes to practical system implementations, particularly in parallel settings. To address these issues, a refinement to the Hellman trade-off was proposed by Philippe Oechslin in 2003, known as the rainbow trade-off [7]. The rainbow trade-off builds upon the foundation of the Hellman trade-off but introduces improvements. It applies different G_i ($1 \leq i \leq t$) functions to each column of the precomputed rainbow matrix. Figure 3 demonstrates the distinction between the Hellman trade-off and the Rainbow trade-off G functions. Through this approach, the precomputed chains generated will only merge when the collisions between two chains occur at the same position in both chains. When the length of the precomputed chain is t, the probability of collision occurrence is 1/*t*. Figure 2-(b) illustrates the structure of the Rainbow Matrix generated during the precomputation phase.

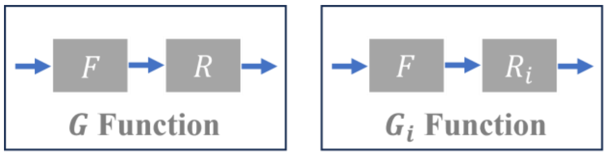

Figure 3. Comparison of Reduce Function.

In terms of a singular table, the success rate of a rainbow trade-off is:

$$P_{single}=1-\prod_{i=0}^{t-1}\left(1-\frac{\mathrm{m}_i}{\mathrm{N}}\right) \tag{3}$$

When t is the length of a chain, $\mathrm{m}_0=m$ and $\mathrm{m}_{i+1}=\mathrm{N}(1-\exp(-\mathrm{m}_i/\mathrm{N}))(0\leq i\leq t-1)$, assuming the existence of l tables, the success rate of a rainbow trade-off is:

$$P_{success}=1-\prod_{i=0}^{t-1}\left(1-\frac{\mathrm{m}_i}{\mathrm{N}}\right)^{l} \tag{4}$$

Indeed, in the precomputation phase of the rainbow trade-off, the required space is $\mathrm{M}=ml$. The number of iterations needed during the online phase is $\mathrm{T}=t^2l/2$. Oechslin suggested setting $m=\mathrm{N}^{2/3}$, $l=1$, and $t=\mathrm{N}^{1/3}$. Therefore, the trade-off curve for the rainbow trade-off is $\mathrm{TM}^2=\mathrm{N}^2/2$.

In the context of cryptographic analysis, the time-memory-data trade-off is a variation of the classic time-memory trade-off problem [8]. This problem involves taking D inputs and successfully attacking at least one of them. Specifically, when D=1, this problem simplifies to the conventional time-memory trade-off problem.

When studying this problem, the original rainbow trade-off's trade-off curve is $\mathrm{TM}^2\mathrm{D}=\mathrm{N}^2$. Compared to the Hellman method's trade-off curve $\mathrm{TM}^2\mathrm{D}^2=\mathrm{N}^2$, the original rainbow trade-off curve is less efficient. As a result, several variants of the rainbow trade-off have been proposed to improve the time complexity in the online phase and enhance its overall performance in practical applications. Figure 4 illustrates the structures of precomputed chains for various variants of rainbow trade-offs.

2.4.1. *Thin-Rainbow Trade-Off*

The concept of a "thin-rainbow trade-off " involves reducing the number of distinct G_i functions to s and cycling through these s G_i functions in a periodic manner [9]. The trade-off curve generated by this approach is $\mathrm{TM}^2\mathrm{D}^2=\mathrm{N}^2$, like that of the Hellman trade-off.

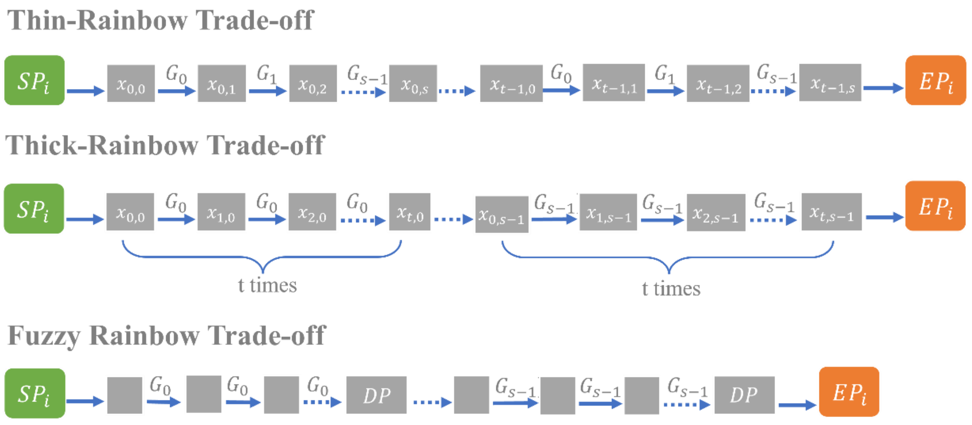

Figure 4. The precomputed chain structures of rainbow trade-off variants.

2.4.2. *Fuzzy-Rainbow Trade-Off*

The counterpart to the thin-rainbow trade-off is the "thick-rainbow trade-off". In this method, the number of distinct G_i functions is also reduced to s. However, the key difference is that instead of repeating a single G_i function with t times before switching to the next one, the thick-rainbow trade-off applies a different G_i function after each t repetition of a single G_i function [9]. The resulting trade-off curve, in this case, is $\mathrm{TM}^2\mathrm{D}=\mathrm{N}^2$, which achieves efficiency like the original rainbow trade-off in handling time-memory-data trade-offs. The concept of "fuzzy rainbow trade-off" enhances rainbow-based attacks by combining the DP trade-off with the thick-rainbow scheme. Incorporating DP point improvements led to a trade-off curve of $2TM^2D^2=N^2+ND^2M$, where $T\geq D^2$.

2.5. Perfect Table Trade-Off

The DP trade-off indeed significantly reduces the number of table lookups required during the online phase compared to the Hellman trade-off. Furthermore, it offers the convenience of easily checking for chain merges. After calculating DP points, if a merge is detected within the current computation chain, the approach involves retaining the longer chain from multiple merged chains until m precomputed chains are generated. This process constructs a perfect DP table without any merges. The Hellman trade-off can also generate unmerged perfect Hellman tables. However, unlike the DP trade-off, creating a perfect Hellman precomputation table requires extensive checks, making its cost prohibitively high and impractical.

Similarly, when applying rainbow trade-off, effective merge detection can be achieved as well. This trade-off enables the creation of a perfect rainbow table without merges. However, an important distinction to note is that perfect tables created through the DP trade-off contain unique nodes, while in rainbow tables, merges between different columns can be retained. As a result, perfect rainbow tables may contain duplicate nodes in certain cases where merges are allowed across different columns.

Perfect tables offer superior search space coverage due to minimal collisions, leading to higher success rates in password attacks compared to non-perfect tables under the same storage constraints. However, generating perfect tables for efficiency demands substantial precomputation time, often overlooked in analyses. In resource-limited scenarios, precomputation costs become pivotal. Extending precomputation duration significantly, even for substantial gains, is impractical.

3. Storage Optimization

In the given trade-off curves, the symbol "M" refers to the total number of starting and ending points key-value pairs stored in the pre-computed table. However, in practical applications, understanding the actual physical size of the precomputed table is of paramount importance. Traditional time-memory trade-off methods typically store the starting and ending point key-value pairs in 2logN bits. This can be regarded as an upper limit on the storage for storing the starting and endpoint values. Below, I will outline several techniques to assist us in more efficiently utilizing storage space.

3.1. Consecutive Starting Points

The first approach involves storage optimization by selecting starting points that occupy fewer storage bits. Conclusions can be drawn from the definition of random functions, indicating that as long as the choice of starting points is unrelated to the one-way function or G function being targeted by the current system, the selection of starting points can be made in any manner without affecting the success rate of the attack on the system. One typical method for choosing starting points is to opt for consecutive starting points. This approach enables the utilization of log*m* space to store m consecutive starting points [10].

3.2. DP Definition

In schemes applying the DP trade-off, any information that can be reconstructed from the DP attribute can be discarded when storing endpoint values. For instance, if the first d bits are defined as 0 for DP points, then eliminating these d bits when storing endpoint values would have no impact on the overall computational efficiency. Furthermore, the cost of recovering this information is negligible.

3.3. Index Table

The index table method can be regarded as a specific case of the data structure known as a hash table. In recent years, research on the index table method and its relationship with time-memory trade-off methods has gained prominence. Figure 5 depicts a basic schematic of an index table structure.

It dissects the endpoint into an index part and a data part. The data part is stored in the pre-computed table alongside the starting point, while the index part is retained within the index table. By utilizing an index, the corresponding table entry in the pre-computed table can be located. The advantage of using the index table method lies in its fast retrieval speed, yet a drawback is that the additional index tables can consume extra storage space. In practice, the index table can store the number of entries corresponding to each index entry, rather than the complete physical address. Moreover, as long as the appropriate ratio is chosen between the index part and the data part of the table entries in the precomputed table, the additional space consumed by the index table can be negligible [11].

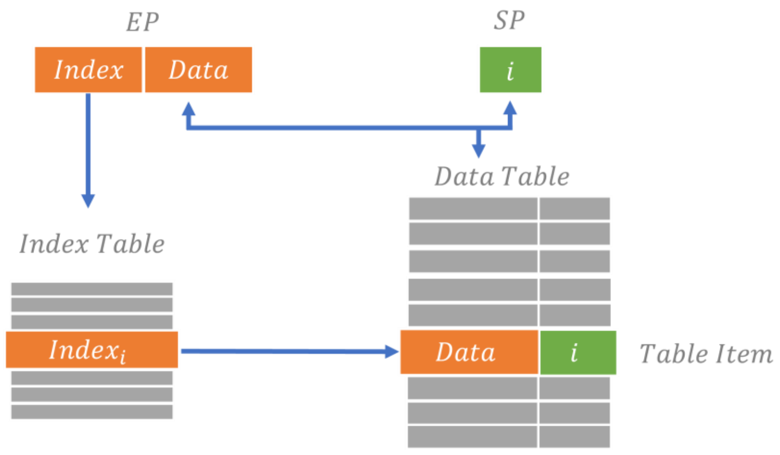

Figure 5. Index table technique.

3.4. Ending Point Truncation

The three methods mentioned above, while reducing storage space, retain all the information of the pre-computed table entries. In contrast, the Endpoint Truncation method is different. It involves truncating the endpoint while storing the key-value pairs of the starting and endpoint. This approach can lead to certain false alarms, serving as a trade-off between storage space and online phase complexity [11]. In the pre-computation phase, when truncating the endpoints in the table entries, it is essential to retain enough bits (slightly exceeding log$m$) to uniquely identify each pre-computed chain. During the online phase, when conducting table lookups, the subject of the search will be truncated to the same length before being compared with the entries in the precomputation table. Although truncating the endpoints could potentially trigger false alarms even for two distinct non-merged precomputed chains, it's unnecessary to devise new procedures explicitly to counter this false alarm. This is because false alarms can still occur even in the absence of endpoint truncation. Nevertheless, excessively aggressive endpoint truncation positions can lead to a higher frequency of false alarms, underscoring the need to carefully manage the length of the truncated endpoints.

4. Implementation Platforms

In recent years, with the rise of fields like artificial intelligence and deep learning, compute-intensive tasks have become the mainstream of computation. Traditional computing technologies represented by CPUs face numerous challenges when dealing with compute-intensive tasks, such as inadequate bandwidth, low energy efficiency, and high computational latency. When both the pre-computation and attack phases are CPU-driven, the problem size that time-memory trade-off methods can handle becomes limited. In this scenario, the processing bottleneck is no longer the size of memory but rather the unacceptable time taken by the precomputation phase [12]. Subsequently, we will introduce hardware platforms that utilize brute-force attacks.

4.1. FPGA
The most distinctive feature of Field-Programmable Gate Arrays (FPGAs) lies in their configurational flexibility. FPGAs are chips that can be reconfigured, constituting a hardware-reconfigurable architecture. Through programming, their application scenarios can be altered at will, greatly reducing the development time and costs of cryptographic attack accelerators. In 2006, the COPACOBANA machine, composed of 120 Xilinx Spartan-3 1000 FPGAs, was manufactured. Its cost was only $10,000, and on average, it took less than 6.4 days to crack a DES key. Later, COPACOBANA was employed for the A5/1 algorithm in GSM voice encryption. Leveraging a thin-rainbow trade-off + DP trade-off time-memory-data trade-off scheme, a success rate of 96% was achieved with knowledge of only 4 frames of the key stream [13].

4.2. GPU
Since the emergence of General-purpose computing on graphics processing units (GPGPU) , GPUs have been applied to various domains beyond image processing. In the context of the DES encryption algorithm, within the pre-computation phase of the rainbow trade-off, the calculation of rainbow chains is distributed to each GPU thread. This significantly enhances the efficiency of rainbow chain generation. However, the high performance of GPUs comes at the cost of high energy consumption. Numerous studies indicate that the energy consumption of GPU-based password recovery accelerators is twice as much as that of FPGA-based accelerators [14]. The primary reason for this discrepancy is that GPUs were not designed with the specific application of password recovery in mind.

4.3. CPU-GPU Heterogeneous Platform
Within the GPU, there exists a multitude of processing cores capable of simultaneously executing thousands of computational tasks. The CPU-GPU architecture is particularly well-suited for processing scenarios demanding high performance, making it an apt platform for password recovery. On a heterogeneous architecture powered by an Intel Xeon 8176M CPU and two NVIDIA Tesla V100 GPUs, a series of comparative experiments were conducted using different cryptographic hash functions to draw conclusions. When compared with state-of-the-art third-party tools like Cryptohaze and RainbowCrack, the application of index table techniques resulted in a reduction of approximately 57.1% in memory space usage. The acceleration ratios for the precomputation phase were 2.03 times and 131.3 times, respectively. Meanwhile, the acceleration ratios for the online phase were 1.97 times and 1.44 times, respectively [15].

4.4. CPU-FPGA Heterogeneous Platform
CPU-FPGA heterogeneous devices are also noteworthy heterogeneous platforms in the industry. They integrate CPUs and FPGAs on the same chip and connect them through high-speed communication interfaces. The FPGA is responsible for accelerating data processing, while the CPU handles other minor computational tasks such as data interaction. There are currently several relatively mature products, such as the Xilinx Zynq-7000 SoC and Intel Stratix 10 SoC. When compared with CPU+GPU heterogeneous devices, CPU-FPGA heterogeneous devices demonstrate remarkable energy efficiency [16]. A SHA256Crypt password recovery accelerator based on a CPU-FPGA device, specifically the Xilinx Zynq-7000 XC7Z030-3 SoC, improves energy efficiency by 2.54 times compared to Hashcat running on the NVIDIA GTX1080Ti GPU platform. In comparison to accelerators purely based on FPGA implementation, the energy efficiency is enhanced by 1.64 times, while the resource efficiency is improved by 1.69 times [17].

5. Conclusion
This paper provides a concise overview of various methods within the domain of time-memory trade-off techniques. It comprehensively outlines the Hellman trade-off, the DP trade-off, and the rainbow trade-off, elucidating the implementation principles, pros and cons, and security implications of time-

memory trade-off techniques. The performance optimization of these techniques is discussed from the perspectives of storage space and implementation platforms.

In practical applications, time-memory trade-off techniques are not limited to the adoption of a single method; rather, they often involve the combination of two or even three different methods. This principle extends to both storage optimization and speed enhancement. Furthermore, the combinations are not confined solely to the methods introduced within this paper. Concerning implementation platforms, heterogeneous setups like CPU-GPU and CPU-FPGA configurations are currently prominent. The integration with GPUs generally provides heightened performance, while partnering with FPGAs offers a favorable balance between cost and performance. It's essential to note that the choice between these methods is context-dependent, relying on considerations such as success rates and budget constraints.

For anyone interested in password attacks and time-memory trade-off methods, the content of this paper can serve as a comprehensive reference. Additionally, it offers valuable insights for those involved in the implementation of time-memory trade-off systems.

References

[1] Goldreich O, Levin LA. A hard-core predicate for all one-way functions. In: Proceedings of the twenty-first annual ACM symposium on Theory of computing. New York, NY, USA: Association for Computing Machinery, pp. 25–32.

[2] Avoine G, Carpent X. Optimal Storage for Rainbow Tables. In: Lee H-S, Han D-G (eds) Information Security and Cryptology -- ICISC 2013. Cham: Springer International Publishing, 2014, pp. 144–157.

[3] Hong J, Moon S. A Comparison of Cryptanalytic Tradeoff Algorithms. J Cryptol 2013; 26: 559–637.

[4] Avoine G, Junod P, Oechslin P. Time-Memory Trade-Offs: False Alarm Detection Using Checkpoints. In: Maitra S, Veni Madhavan CE, Venkatesan R (eds) Progress in Cryptology - INDOCRYPT 2005. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 183–196.

[5] Hellman M. A cryptanalytic time-memory trade-off. IEEE Transactions on Information Theory 1980; 26: 401–406.

[6] Denning, D. E. Cryptography and Data Security (p.100, Ron Rivest's distinguished points observation). Addison-Wesley, 1982.

[7] Oechslin P. Making a Faster Cryptanalytic Time-Memory Trade-Off. In: Boneh D (ed) Advances in Cryptology - CRYPTO 2003. Berlin, Heidelberg: Springer, 2003, pp. 617–630.

[8] Biryukov A, Mukhopadhyay S, Sarkar P. Improved Time-Memory Trade-Offs with Multiple Data. In: Preneel B, Tavares S (eds) Selected Areas in Cryptography. Berlin, Heidelberg: Springer, 2006, pp. 110–127.

[9] Barkan E, Biham E, Shamir A. Rigorous Bounds on Cryptanalytic Time/Memory Tradeoffs. In: Dwork C (ed) Advances in Cryptology - CRYPTO 2006. Berlin, Heidelberg: Springer, 2006, pp. 1–21.

[10] Borst J. Block Ciphers: Design, Analysis and Side-Channel Analysis, https://lirias.kuleuven.be/3006194 (2001, accessed 28 October 2022).

[11] Biryukov A, Shamir A, Wagner D. Real Time Cryptanalysis of A5/1 on a PC. In: Goos G, Hartmanis J, van Leeuwen J, et al. (eds) Fast Software Encryption. Berlin, Heidelberg: Springer, 2001, pp. 1–18.

[12] Avoine G, Carpent X, Leblanc-Albarel D. Rainbow Tables: How Far Can CPU Go? The Computer Journal 2022; bxac147.

[13] Güneysu T, Kasper T, Novotný M, et al. Cryptanalysis with COPACOBANA. IEEE Trans Comput 2008; 57: 1498–1513.

[14] Liu P, Li S, Ding Q. An Energy-Efficient Accelerator Based on Hybrid CPU-FPGA Devices for Password Recovery. IEEE Transactions on Computers 2019; 68: 170–181.

[15] Li P, Zhu W, Chen J, et al. High-speed implementation of rainbow table method on heterogeneous multi-device architecture. Future Generation Computer Systems 2023; 143: 293–304.

[16] Zhang Z, Liu P, Wang W, et al. High-Performance Password Recovery Hardware Going From GPU to Hybrid CPU-FPGA Platform. IEEE Consumer Electronics Magazine 2022; 11: 80–87.
[17] Zhang Z, Liu P. A Hybrid-CPU-FPGA-based Solution to the Recovery of Sha256crypt-hashed Passwords. IACR Transactions on Cryptographic Hardware and Embedded Systems 2020; 1–23.