A New Variant of Time Memory Trade-Off on the Improvement of Thing and Ying's Attack

Zhenqi Li[1], Yao Lu[2], Wenhao Wang[2], Bin Zhang[2], and Dongdai Lin[2]

[1] Institute of Software Chinese Academy of Sciences, Beijing, China
[2] The State Key Laboratory of Information Security
Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{lizhenqi,luyao,wangwenhao,zhangbin,ddlin}@is.iscas.ac.cn

Abstract. In this paper, we present a rigorous evaluation of Thing and Ying's attack (TY attack) [11] along with practical implementations. We find that the cryptanalysis time of their attack is too high to be practical. We also propose a more general time memory trade-off by combining the distinguished points strategy with TY attack. Both theoretical analysis and experimental results show that our new design can save about 53.7% cryptanalysis time compared to TY attack and can reduce about 35.2% storage requirement compared to the original rainbow attack.

Keywords: time memory tradeoff, cryptanalysis, rainbow attack.

1 Introduction

A basic problem in symmetric-key cryptology is the computation of preimages or inversion of one-way functions. There are two straightforward ways (suppose the function has an n-bit input): first one can perform an exhaustive search over an average of 2^{n-1} values until the target is reached. A second solution is to precompute and store 2^n input and output pairs in a table. If one then needs to invert a particular value, one just looks up the preimage in the table, so inverting requires only a single table lookup. Both methods will be impractical if n becomes larger. Cryptanalytic time memory trade-off (TMTO) is a technique that comes between these two extremes. It inverts a one-way function in time shorter than the exhaustive search method, using a storage smaller than the table lookup method.

Since the first TMTO algorithm was proposed by Hellman [6], many of its extensions [5,3,1] and variants [9,7,2,8,4] have appeared. In 2009, Thing and Ying proposed a new TMTO [11] for password recovery. Compared to the traditional rainbow table, it has higher success probability and lower storage requirements. In this paper, we present a rigorous evaluation on the performance of TY attack along with practical implementations, we find that it has high cryptanalysis time. Combining the *distinguished point* (DP) [5] strategy with TY attack, we design a new variant of TMTO, which is a general framework not only applicable to password crack but also to cryptanalysis of cryptosystems. We also make a comparison between our new design and TY attack. Experimental results show

T.W. Chim and T.H. Yuen (Eds.): ICICS 2012, LNCS 7618, pp. 311–320, 2012.
© Springer-Verlag Berlin Heidelberg 2012

that new design can save about 53.7% cryptanalysis time compared to TY attack and can reduce about 35.2% storage requirement compared to original rainbow attack.

The paper is organized as follows. In Section 2, some basic TMTO methodologies are provided, followed with the analysis of TY attack. Formal definitions and algorithms of our new design are given in Section 3. Section 4 identifies the performance evaluation and experimental results. Finally, conclusions are in Section 5.

2 Time-Memory Trade-Off Methodology and Analysis of TY Attack

2.1 Time-Memory Trade-Off Methodology

Let f be a one-way function, given output y ($y \in Y$), the trade-off target is to recover the corresponding preimage x ($x \in X$) satisfying $f(x) = y$, where X and Y are the input space and the output space respectively. In the *offline* stage of Oechslin's TMTO [9], we randomly choose m starting points: $SP_0, SP_1, ..., SP_{m-1}$ ($SP_i \in X,\ 0 \leq i \leq m-1$) and iteratively compute SP_i for t times by using a compound function: $F_j = R_j \circ f$, where R_j is called *reduction* function or *mask* function which maps Y to X, $1 \leq j \leq t$ and $\circ$ denotes function composition. The *offline* computation is as follows.

$$
\begin{array}{lll}
SP_0 & = x_{0,0} \xrightarrow{F_1} x_{0,1} \xrightarrow{F_2} x_{0,2} \cdots \xrightarrow{F_t} x_{0,t} = & EP_0 \\
SP_1 & = x_{1,0} \xrightarrow{F_1} x_{1,1} \xrightarrow{F_2} x_{1,2} \cdots \xrightarrow{F_t} x_{1,t} = & EP_1 \\
 & \vdots & \vdots \\
SP_{m-1} & = x_{m-1,0} \xrightarrow{F_1} x_{m-1,1} \cdots \xrightarrow{F_t} x_{m-1,t} = & EP_{m-1}
\end{array}
$$

we only store $(SP_0, EP_0), (SP_1, EP_1), ..., (SP_{m-1}, EP_{m-1})$ in a table called rainbow table and sort the table with respect to ending points. In the *online* stage, we firstly apply R_t to y and look up the result in the ending points of the table. If we find a matching ending point, we know how to rebuild the chain using the corresponding starting point and locate x. If we don't find a match, we try if we find it by applying R_{t-1}, F_t to see if the preimage was in the second last column of the table. Then we try to apply R_{t-2}, F_{t-1}, F_t, and so forth.

2.2 Analysis of TY Attack

The basic idea of TY attack is similar to rainbow attack. The difference lies in their table structures. Suppose h is a hash function which is often used to password encryption. The precomputation of the i-th table is as follows.

$$
\begin{array}{lll}
x^i \xrightarrow{h} & H & \xrightarrow{R_1} x^i_{1,1} \xrightarrow{F_2} x^i_{1,2} \cdots \xrightarrow{F_t} x^i_{1,t} \\
& H+1 & \xrightarrow{R_1} x^i_{2,1} \xrightarrow{F_2} x^i_{2,2} \cdots \xrightarrow{F_t} x^i_{2,t} \\
& H+2 & \xrightarrow{R_1} x^i_{3,1} \xrightarrow{F_2} x^i_{3,2} \cdots \xrightarrow{F_t} x^i_{3,t} \\
& \vdots & \qquad\qquad \vdots \\
& H+k & \xrightarrow{R_1} x^i_{k+1,1} \xrightarrow{F_2} x^i_{k+1,2} \cdots \xrightarrow{F_t} x^i_{k+1,t}
\end{array}
$$

where H is the hash value of x^i and $F_j = R_j \circ h$ ($1 \leq j \leq t$). It only store one starting point of x^i and $k+1$ ending points for the i-th table. k is a constant value to control the table size. The *online* analysis is the same to that of rainbow attack. When a match is found in the table, it is easy to rebuild the corresponding $H + d$ ($0 \leq d \leq k$) by using the matched chain index and the stored x^i, then locate the possible preimage.

In [11], Thing et al. said that the optimal value of k is $2m-2$, where m is number of chains in rainbow table. In this way, storage usage can be maximum and only one table is computed in the *offline* stage. It can save 50% storage requirement in comparison to rainbow table. However, we found that it requires higher cryptanalysis time which makes it to be impractical in real world applications. In traditional TMTO, we often sort the precomputed table and the searching time in a sorted table is often ignored. But in TY attack, hash value of the second column increased in order and sorting will break this order, thus disturbing the correctness of the attack. Searching in an unsorted table will greatly increase the *online* cost. *Online* time comparisons between TY attack and rainbow attack are listed in Table 1.

Table 1. *Online* time complexity comparison

Attack	Rainbow attack	TY attack
Parameters	(m, t, r)	(k, t, r)
Function calculation	$O(\frac{t^2}{2})$	$O(\frac{t^2}{2})$
Table look-up	$O(t)$	$O(t)$
Comparison of each table look-up	$O(log(m))$	$O(k+1)$

From table 1, given $N = 2^{24}, m = 2^{15}, t = 2^9, r = 1$ for rainbow attack and $N = 2^{24}, k = 2m-2 = 2^{16}-2, t = 2^9, r = 1$ for TY attack, table look-up of TY attack needs totally $t(k+1) = 2^{25} - 2^9 \approx 2^{25}$ comparisons, which is slightly larger than brute force comparisons of 2^{24}. The *online* performance comparisons[1] among rainbow attack, TY attack and brute force attack are given in Table 2.

[1] We randomly generate 500 integers in the searching space $\{i | 0 \leq i \leq 2^{24} - 1, i \in \mathbf{Z}\}$ and calculate their digest values by using MD5. Inversion of each digest value is done by rainbow attack, TY attack and brute force attack.

Table 2. Experimental results of the *online* time complexity comparison

	Rainbow attack	TY attack	Brute force attack
$(m,t,r)\|(k,t,r)\|N$	$(2^{15}, 2^9, 1)$	$(2^{16}-2, 2^9, 1)$	2^{24}
	Average cryptanalysis time		
to success	1.86 sec	7.59 sec	14.70 sec
to failure	2.58 sec	15.18 sec	-
total	2.22 sec	9.83 sec	14.70 sec
	Average function calculations		
to success	43505	74978	4499522
to failure	174654	215898	-
total	110391	116549	4499522
	Average false alarms		
to success	52	139	-
to failure	256	497	-
total	156	245	-

From Table 2, the average function calculations (total) of TY attack is almost the same to that of rainbow attack as we just expected in Table 1, but it takes more cryptanalysis time than rainbow attack in all cases, since the cost of table look-up dominates the total *online* cost. In failure case, TY attack takes more time than brute force attack.

In the meantime, Ying and Thing themselves also found the existing drawback of TY attack and proposed a sorting method [12] to improve the performance of the recovery process. The basic idea is to add some tags to each ending point and sort the ending points in the usual alphabetical order. These tags can be used to derive the corresponding initial hash value, which correctly solve the sorting problem. However, the revised attack can only be applied to password cracking scenario and the existence of these reserved tags will add difficulties in designing the index algorithm. They also did not present any experimental comparisons between their improved version and original rainbow attack but only gave a theoretical estimation of 23% reduction in storage requirement, which seems to be lack of convincing.

3 A New Design

In this section, we propose a new variant of TMTO by combining DP strategy with TY attack. It is a general framework and can be applied not only to password cracking but also to the cryptanalysis of cryptosystem.

3.1 *Offline* Stage

In the *offline* stage, we choose a constant value X (i.e.,$X = 0$) and compute $k_1 \times k_2$ starting points through $H(X + i) + j$ $(0 \leq i < k_1,\ 0 \leq j < k_2)$. Then, we choose t_{max} different evaluation function: $F_1, F_2, ..., F_{t_{max}}$, where $F_k = R_k \circ h,\ 1 \leq k \leq t_{max}$ and R_k is the *reduction* function. We iteratively compute the (i, j)-th chain through $X^j_{i,k} = F_k(X^j_{i,k-1}), X^j_{i,0} = (H(X + i) \oplus j)\ mod\ N, 1 \leq k \leq t_{max}$. The chain stops when the most significant $|k_2|$[2] bits of some $X^j_{i,k}$ is found to be j or the current chain length exceeds t_{max}. If the chain stops in the latter case, we discard it. The *offline* stage can be shown as follows.

$$SP^0_0 = H(X+0) \oplus 0 \xrightarrow{F_1} \circ \xrightarrow{F_2} \cdots \xrightarrow{F_{l^0_0}} (0 \parallel R^0_0) = EP^0_0$$
$$SP^0_1 = H(X+1) \oplus 0 \xrightarrow{F_1} \circ \xrightarrow{F_2} \cdots \xrightarrow{F_{l^0_1}} (0 \parallel R^0_1) = EP^0_1$$
$$\vdots \qquad\qquad\qquad \vdots$$
$$SP^0_{k_1-1} = H(X+k_1-1) \oplus 0 \xrightarrow{F_1} \circ \xrightarrow{F_2} \cdots \xrightarrow{F_{l^0_{k_1-1}}} (0 \parallel R^0_{k_1-1}) = EP^0_{k_1-1}$$

$$SP^1_0 = H(X+0) \oplus 1 \xrightarrow{F_1} \circ \xrightarrow{F_2} \cdots \xrightarrow{F_{l^1_0}} (1 \parallel R^1_0) = EP^1_0$$
$$SP^1_1 = H(X+1) \oplus 1 \xrightarrow{F_1} \circ \xrightarrow{F_2} \cdots \xrightarrow{F_{l^1_1}} (1 \parallel R^1_1) = EP^1_1$$
$$\vdots \qquad\qquad\qquad \vdots$$
$$SP^1_{k_1-1} = H(X+k_1-1) \oplus 1 \xrightarrow{F_1} \circ \xrightarrow{F_2} \cdots \xrightarrow{F_{l^1_{k_1-1}}} (1 \parallel R^1_{k_1-1}) = EP^1_{k_1-1}$$

$$\vdots \qquad\qquad\qquad \vdots$$

$$SP^{k_2-1}_0 = H(X+0) \oplus (k_2-1) \xrightarrow{F_1} \circ \xrightarrow{F_2} \cdots \xrightarrow{F_{l^{k_2-1}_0}} (k_2 - 1 \parallel R^{k_2-1}_0) = EP^{k_2-1}_0$$
$$SP^{k_2-1}_1 = H(X+1) \oplus (k_2-1) \xrightarrow{F_1} \circ \xrightarrow{F_2} \cdots \xrightarrow{F_{l^{k_2-1}_1}} (k_2 - 1 \parallel R^{k_2-1}_1) = EP^{k_2-1}_1$$
$$\vdots \qquad\qquad\qquad \vdots$$
$$SP^{k_2-1}_{k_1-1} = H(X+k_1-1) \oplus (k_2-1) \xrightarrow{F_1} \circ \xrightarrow{F_2} \cdots \xrightarrow{F_{l^{k_2-1}_{k_1-1}}} (k_2 - 1 \parallel R^{k_2-1}_{k_1-1}) = EP^{k_2-1}_{k_1-1}$$

[2] For all $||$ in this paper, $|\alpha|$ means the binary length of integer α.

For each chain, we only store:

$$\begin{array}{ll} S[0,0] = \{R_0^0, l_0^0, 0\} & 0 < l_0^0 \le t_{max} \\ S[1,0] = \{R_1^0, l_1^0, 1\} & 0 < l_1^0 \le t_{max} \\ \vdots & \\ S[i,j] = \{R_i^j, l_i^j, i\} & 0 < l_i^j \le t_{max} \\ \vdots & \\ S[k_1-1, k_2-1] = \{R_{k_1-1}^{k_2-1}, l_{k_1-1}^{k_2-1}, k_1-1\} & 0 < l_{k_1-1}^{k_2-1} \le t_{max} \end{array}$$

where R_i^j $(0 \le i \le k_1-1, 0 \le j \le k_2-1)$ is the rest $(n-|k_2|)$ bits of the ending point in the (i,j)-th chain, l_i^j is the length of the (i,j)-th chain and $n = |EP_i^j|$.

All these chains have different lengths and can be split into groups of size k_2 according to their definition of DP. We sort each DP group with respect to R_i^j and store them in k_2 tables indexed by their DP definition, which is also equal to j of the starting points and also to the most significant $|k_2|$ bits of ending points.

Let $d_1 = |k_1|$, $d_2 = |k_2|$, $l = |t_{max}|$ and $n = |N|$, then $|S[i,j]| = |R_i^j| + |l_i^j| + |i| = n - d_2 + l + d_1$ bits. We have $k_1 \times k_2$ starting points, thus the storage requirement is $M = P \times k_1 \times k_2 \times (n - d_2 + l + d_1)$ bits, where P $(0 < P \le 1)$ is the proportion of chains which meet a predefined DP before their length reach t_{max}. More details of P is given in the next section.

3.2 *Online* Stage

Give Y (ciphertext in block ciphers and MACs, key stream segment in stream ciphers, hash value in password encryptions, etc), to lookup the preimage (secret key in block ciphers and MACs, internal state in stream ciphers, password in password encryptions, etc), we proceed in the following manner: First we apply $R_{t_{max}}$ to the ciphertext Y and get $Y_0 = R_{t_{max}}(Y)$, Y_0 is now a DP for some definition of DP. The value of the most significant $|k_2|$ bits of Y_0 is corresponding to a table in which the most significant $|k_2|$ bits of each ending point equals to that of Y_0. Then, we look up the rest $n - |k_2|$ bits in this table as follows.

$$\begin{cases} R_i^j \stackrel{?}{=} (rest\ (n - |k_2|)\ bits\ of\ Y_0) \\ l_i^j \stackrel{?}{=} t_{max} \end{cases} \tag{1}$$

If both equations succeed, then a match is found and we get the corresponding index i stored in the match $S[i,j]$ and compute $X_{i,t_{max}-1}^j$ from the starting point $(H(X+i) \oplus j)\ mod\ N$, where j is the value of the most significant $|k_2|$ bits of Y_0. Then we check whether $X_{i,t_{max}-1}^j$ is the preimage or a *false alarm.*

If either equation of (1) fails, then we apply $R_{t_{max}-1}, F_{t_{max}}$ to Y as $Y \xrightarrow{R_{t_{max}-1}} Y_1 \xrightarrow{F_{t_{max}}} Y_0$ and check Y_0 and Y_1 separately. Provided that we have computed Y iteratively for k times and $1 \le k \le t_{max}$:

$$X \xrightarrow{R_{t_{max}-(k-1)}} Y_{k-1} \xrightarrow{F_{t_{max}-(k-2)}} Y_{k-2} \cdots \xrightarrow{F_{t_{max}}} Y_0. \tag{2}$$

We search each $Y_q (0 \leq q \leq k-1)$ in a corresponding DP table and check

$$\begin{cases} R_i^j \stackrel{?}{=} (rest\ (n - |k_2|)\ bits\ of\ Y_q) \\ l_i^j \stackrel{?}{=} t_{max} - q \end{cases} \tag{3}$$

if both equations succeed, then a match is found. $X^j_{i,t_{max}-q}$ can be computed from the starting point $(H(X+i) \oplus j)\ mod\ N$ by iteratively doing the computation from F_1 to $F_{t_{max}-q-1}$. Then we check whether $X^j_{i,t_{max}-q}$ is a *false alarm* or the preimage. If no match is found or *false alarm* occurred, then we set $k \leftarrow k+1$ and repeat the process above until $k > t_{max}$. It is easy to know that new design needs $O(\frac{t_{max}^2}{2})$ function calculations and $O(\frac{t_{max}^2}{2})$ table look-ups, each table look-up only needs $log_2|k_1|$ comparison because of the sorted ending points.

3.3 The Selection of t_{max}

The main modification caused by the introduction of DP is the variable chain length. Therefore, the selection of t_{max} has a great influence on the performance of the new design. Let $k = |k_2|$, $n = |N|$ and $P_1(t)$ be the probability that a DP is reached in less than t iterations. Let $P_2(t)$ be the probability that no DP is reached in less than t iterations. Thus $P_1(t) = 1 - P_2(t)$ and we can easily get $P_2(t) = \prod_{i=0}^{t-1}(1 - \frac{2^{n-k}}{2^n - i})$. An approximate expression can be obtained knowing that $i \ll 2^n$. By fixing i to $\frac{t-1}{2}$, we have $P_2(t) \approx (1 - \frac{2^{n-k}}{2^n - \frac{t-1}{2}})^t$. Finally, we have $P_1(t) \approx 1 - (1 - \frac{2^{n-k}}{2^n - \frac{t-1}{2}})^t$, thus the probability to reach a DP in less than t_{max} iterations is $P_1(t_{max})$ which is also the P we defined in Section 3.3. According to [10], The average chain length of a DP table is $\bar{t} = 2^k = k_2$. Given $N = 2^{24},\ k_1 = 2^8,\ and\ k_2 = 2^8$, the theoretical and experimental results of $P_1(t_{max})$ are listed in Table 3.

Table 3. The value of $P_1(t_{max})$

t_{max}	Theoretical result	Experimental result
1.0×2^8	63.28%	63.26%
1.5×2^8	77.75%	77.56%
2.0×2^8	86.25%	86.43%
2.5×2^8	91.83%	91.90%
3.0×2^8	95.05%	95.13%

[a] $(N, k_1, k_2) = (2^{24}, 2^8, 2^8)$

From Table 3, we see that the larger t_{max} is, the higher $P_1(t_{max})$ will be. However, larger t_{max} also leads to higher time complexity in the *online* stage as described in Section 3.3. Therefore, t_{max} should not be too large or too small.

4 Performance Evaluation and Experimental Results

In this section, we present a rigorous evaluation on the performance of our new design correspond with experimental results (the analysis of success probability can be found in the full version).

4.1 *Online* Time Complexity Comparison

Given $N = 2^{32}$, common parameters of TY attack are $(k, t, r) = (2 \times 2^{20} - 2, 2^{12}, 1)$ and common parameters of new design are $(k_1, k_2, t_{max}) = (455, 2^{12}, 2 \times 2^{12})$. These chosen parameters can assure that both attacks have the same storage requirement and TY attack has the optimal table structure. Experimental results are listed in Table 4.

Table 4. Experimental results of the *online* comparison

	New design	TY attack
$(k_1, k_2, t_{max})\|(k, t, r)$	$(455, 2^{12}, 2^{13})$	$(2^{21} - 2, 2^{12}, 1)$
	Average cryptanalysis time	
to success	3 min, 23.10 sec	7 min, 17.74 sec
to failure	6 min, 7.11 sec	16 min, 46.89 sec
total	4 min, 31.00 sec	9 min, 45.72 sec
	Average function calculations	
to success	4,393,370	4,772,551
to failure	15,470,394	13,971,075
total	12,206,927	7,164,167
	Average false alarms	
to success	548	1153
to failure	1388	4099
total	896	1919

The experimental results show that the average function calculations of our new design is higher than TY attack, but it can save 53.7% cryptanalysis cost, since it needs less table look-ups and occurs less false alarms than TY attack.

4.2 Storage Requirement Comparison

In this part, the basic consumption is that all these attacks have the same precomputation time. Therefore, given $N = 2^{24}$, the common parameters for these attacks and the storage space comparison are listed in Table 5 (more details can be found in the full version).

Table 5. Storage space comparison

Attack	Rainbow attack	New design	TY attack
$(m,t,r)\|(k_1,k_2,t_{max})\|(k,t,r)$	$(2^{15},2^9,1)$	$(2^6,2^9,2^{10})$	$(2^{15},2^9,1)$
Entries	2^{15}	0.8643×2^{15}	$2^{15}+2$
Entry size	48-bit	31-bit	24-bit
Experimental result	256 KB	166 KB	161 KB

$N=2^{24}$

The storage medium is '.txt' file and we put each entry in a single line to the file. Results in Table 5 show that our new design can save about 35.2% storage requirement compared with rainbow attack, TY attack can save about 37.1% storage requirement compared with rainbow attack. For more details and further discussion, please refer to the full version.

5 Conclusion

In this paper, we present a rigorous analysis on the performance of TY attack and find that it has higher precomputation time and its *online* attack time is no better than brute force attack. Therefore, TY attack is an impractical attack even though it has higher success probability and lower storage requirement than rainbow attack. By combining the DP strategy with TY attack, we propose a new variant of TMTO, which is a general framework and can be applied not only to password cracking, but also to cryptanalysis of cryptosystems. Evaluations of the performance show that our new design has higher success probability than rainbow attack and has slightly lower success probability than TY attack under the basic assumption that all these three attacks use the same storage space. It can save about 53.7% cryptanalysis time compared with TY attack and can save about 35.2% storage requirement compared with original rainbow attack. The amount of storage requirement we have saved is slightly lower than that of TY attack (37.1%), but we achieved a great improvement on the cryptanalysis time, making our new design to be a practical TMTO which can well be used in the storage limited applications.

References

1. Avoine, G., Junod, P., Oechslin, P.: Characterization and improvement of time-memory trade-off based on perfect tables. ACM Transactions on Information and Systems Security 11(4), Article 17 (2008)
2. Biryukov, A., Mukhopadhyay, S., Sarkar, P.: Improved Time-Memory Trade-Offs with Multiple Data. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 110–127. Springer, Heidelberg (2006)
3. Biryukov, A., Shamir, A.: Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 1–13. Springer, Heidelberg (2000)

4. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Efficient Dissection of Composite Problems, with Applications to Cryptanalysis, Knapsacks, and Combinatorial Search Problems. Cryptology ePrint Archive. Report 2012/217 (2012)
5. Denning, D.: Cryptography and Data Security, p. 100. Addison-Wesley (1982)
6. Hellman, M.: A Cryptanalytic Time-Memory Trade-Off. IEEE Transactions on Information Theory 26(4), 401–406 (1980)
7. Hong, J., Jeong, K.C., Kwon, E.Y., Lee, I.S., Ma, D.: Variants of the Distinguished Point Method for Cryptanalytic Time Memory Trade-Offs. In: Chen, L., Mu, Y., Susilo, W. (eds.) ISPEC 2008. LNCS, vol. 4991, pp. 131–145. Springer, Heidelberg (2008)
8. Hong, J., Sarkar, P.: New Applications of Time Memory Data Tradeoffs. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 353–372. Springer, Heidelberg (2005)
9. Oechslin, P.: Making a Faster Cryptanalytic Time-Memory Trade-Off. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 617–630. Springer, Heidelberg (2003)
10. Standaert, F.X., Rouvroy, G., Quisquater, J.J., Legat, J.D.: A Time-Memory Tradeoff Using Distinguished Points: New Analysis & FPGA Results. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 593–609. Springer, Heidelberg (2003)
11. Thing, V.L.L., Ying, H.M.: A novel time-memory trade-off method for password recovery. Digital Investigation 6, S114–S120 (2009)
12. Ying, H.M., Thing, V.L.L.: A Novel Rainbow table sorting method. In: Proceedings of the 2nd International Conference on Technical and Legal Aspects of the e-Society, CYBERLAWS 2011 (2011)