Virtual Expansion of Rainbow Tables

Vrizlynn Thing

Chapter 17

VIRTUAL EXPANSION OF RAINBOW TABLES

Vrizlynn Thing

Abstract Password recovery tools are often used in digital forensic investigations to obtain the passwords that are used by suspects to encrypt potential evidentiary data. This paper presents a new method for deterministically generating and efficiently storing password recovery tables. The method, which involves the virtual expansion of rainbow tables, achieves improvements of 16.92% to 28.15% in the password recovery success rate compared with the original rainbow table method. Experimental results indicate that the improvements are achieved with the same computational complexity and storage requirements as the original rainbow table method.

Keywords: Password recovery, cryptanalysis, rainbow table, time-memory trade-off

1. Introduction

Password protection of potential digital evidence by suspects makes investigative work more complex and time consuming. Traditional password recovery techniques include password guessing, dictionary attacks and brute force attacks.

A password guessing technique attempts easily-formed and common passwords such as "qwerty" and "password." These passwords could be based on a user's personal information or a fuzzy index of words on the user's storage media. A statistical analysis of 28,000 passwords revealed that 16% of the users relied on their first names as passwords and 14% relied on "easy-to-remember" keyboard combinations [2]. Therefore, the password guessing method can be effective in cases where users are willing to compromise security for convenience.

A password dictionary attack attempts to match the hash values of dictionary words with the stored password hash value. Well-known tools that employ this technique include Cain and Abel [7], John the Ripper [12] and LCP [6].

In a brute force cryptanalytic attack, the hash value of each unique combination of password characters is compared with the password hash value until a match is found. Although such an attack is extremely time consuming, the password is recovered eventually. Cain and Abel, John the Ripper and LCP also support brute force attacks.

Traditional password recovery techniques are losing their effectiveness as suspects use stronger passwords to protect their data. Hellman [5] introduced a method that trades off the computational time and storage space needed to make a hash-to-plaintext recovery. This method can retrieve Windows login passwords as well as passwords used in other applications (e.g., Adobe Acrobat) that employ the LM and NTLM hash algorithms [15]. Also, the method supports encryption key recovery for Microsoft Word and Excel documents. Passwords encrypted with hashing algorithms such as MD5 [13], SHA-2 [9] and Ripemd-160 [4] can also be recovered using this method. In general, Hellman's method is applicable to searching for solutions to knapsack and discrete logarithm problems.

Oechslin [11] proposed a cryptanalytic time-memory trade-off method that is based on Hellman's password recovery method. This "rainbow table" method generates password recovery tables with higher efficiency because it employs multiple reduction functions to reduce the probability of collisions in a single table.

Thing and Ying [14] enhanced the rainbow table method via initial chain generation. In this technique, a plaintext value is chosen and its hash value is computed by applying the password hash algorithm. Based on this hash value, other hashes are computed; these form the branches of the initial plaintext value. Multiple blocks are created from the different initial plaintext values. The final values of the chains in the blocks are stored with the single initial plaintext value in each block.

The rainbow table method is widely used for password recovery. Products that use pre-computed rainbow tables include RainbowCrack [16] and Ophcrack [10]. RainbowCrack is an implementation of Oechslin's method that supports hash algorithms such as LM, NTLM, MD5 and SHA-1 [8]. Ophcrack also implements the rainbow table method, but it only supports the LM and NTLM hash algorithms. Rainbow tables are also used in several popular commercial tools such as AccessData's Password Recovery Toolkit [1] to perform efficient and effective password recovery.

This paper proposes the virtual expansion of rainbow tables (VERT) method, a new time-memory trade-off technique that relies on a pre-computed table structure. Comparisons with the original rainbow table method [11] and the enhanced rainbow table method [14] demonstrate higher password recovery success rates of 16.92% to 28.15% without corresponding increases in computational complexity and storage space.

2. Related Work

In 1980, Hellman [5] conceived of a general time-memory trade-off hash-to-plaintext recovery method. We begin by describing Hellman's algorithm in the context of password recovery.

Let X be the plaintext password and Y be its corresponding stored hash value. Given Y, it is necessary to find X that satisfies the equation $h(X) = Y$ where h is the known hash function. However, finding $X = h^{-1}(Y)$ is not feasible because the hash values are computed using one-way functions for which the reversal function h^{-1} is unknown.

To solve this problem, Hellman suggested applying alternate hashing and reduction operations to plaintext values in order to generate a pre-computed table. For example, given a 7-character password (using the English alphabet), the MD-5 hash algorithm is used to compute the corresponding 128-bit hash value. Using a reduction function such as $H \bmod 26^7$, where H is the hash value converted to decimal digits, the resulting values are distributed in a best-effort uniform manner.

For an initial plaintext password of `abcdefg`, the binary hash output could be:

```
00000000,00000000,00000000,00000000,00000000,00000000,
00000000,00000000,10000000,00000000,00000000,00000000,
00000000,00000000,00000000,00000001.
```

Therefore, $H = 9223372036854775809$. The reduction function converts this value to `3665127553`, which corresponds to the plaintext representation of `lwmkgij`. This is computed as:

$$11(26^6) + 22(26^5) + 12(26^4) + 10(25^3) + 6(26^2) + 8(26^1) + 9(26^0).$$

In table generation, the hashing and reduction operations are repeatedly performed on different initial plaintext values to produce different rows or chains. The pre-defined multiple rounds of hashing and reduction operations in each chain increase the length of the chains and the table contents; this contributes to higher computational overhead during password recovery. The reason is that the initial and final plaintext values (i.e., "heads" and "tails") of the chains are the only elements stored in the table. The recovery of passwords "residing" in the intermediate columns requires computations to regenerate the plaintext-hash pairs.

The password recovery success rate depends on the size of the pre-computed table. A larger pre-computed table contains more plaintext-hash pairs, which increases the password recovery success rate. However, the generation of the intermediate chain columns may result in collisions of elements in the table. Collisions cause the chains to merge and reduce the number of (distinct) plaintext-hash pairs. This decreases the password recovery success rate.

To increase the success rate, Hellman proposed using multiple tables where each table has a different reduction function. If $P(t)$ is the success rate using t tables, then $P(t) = 1 - (1 - P(1))^t$, which is an increasing function of t. Hence, introducing more tables increases the password recovery success rate but increases the computational complexity and storage requirements.

A plaintext password is recovered from its hash value by performing a reduction operation on the hash value. Next, a search is conducted for a match of the computed plaintext value with a final value in the table. If a match is not found, the hashing and reduction operations are performed on the computed plaintext value to obtain a new plaintext value for another round of search. The maximum number of rounds of hashing, reduction and searching operations depend on the pre-defined chain length.

Rivest [3] suggested using distinguished points as end points when generating the chains. Distinguished points are values that satisfy a given criterion (e.g., the first q bits are 0). In this method, chains are not generated with a fixed length; instead, they terminate upon reaching a pre-defined distinguished point.

This method decreases the number of memory lookups compared with Hellman's method and is capable of loop detection. If a distinguished point is not reached after a large number of operations, it is assumed that the chain contains a loop and is discarded. One limitation is that the chains will merge if there are collisions within the same table. The variable lengths of the chains also increase the number of false alarms. Additional computations are needed to rectify these errors.

Oechslin [11] proposed a new table structure that reduces the probability of chains merging. The method uses a different reduction function to generate the elements in each chain column. Therefore, chains merge only when a collision occurs in the same column. For m chains of length n, the total number of possible chain merges (i.e., when two similar elements appear in the same column) is:

$$\frac{nm(m-1)}{2}.$$

The total number of possible pairs not in the same chain column is:

$$\frac{n^2m(m-1)}{2}.$$

Therefore, given a collision in the table, the probability that there is a chain merge is:

$$\frac{\frac{nm(m-1)}{2}}{\frac{n^2m(m-1)}{2}} = \frac{1}{n}.$$

Oechslin showed that the measured coverage in a single "rainbow" is 78.8% compared with 75.8% in a classical rainbow table constructed using distinguished points. In addition, the search effort is reduced, which contributes to an improvement in performance.

Thing and Ying [14] proposed an enhancement to the rainbow table method. The enhancement is achieved through initial chain generation by systematically manipulating the initial hash values based on an adjustable parameter k. A plaintext value is chosen and its hash value is computed using the password hash algorithm. The resulting hash value H is used to compute:

$$(H+1)\ mod\ 2^j, (H+2)\ mod\ 2^j, \ldots, (H+k)\ mod\ 2^j,$$

where j is the number of bits in the hash output. These hash values form the branches of the initial plaintext value. Alternate hashing and reduction operations are then applied to these branches. The resulting extended chains form a block. The final values of the chains in each block are stored with the single initial plaintext value. The same operations are performed on all the other initial plaintext values. These sets of initial and final values form the new pre-computed table.

Instead of storing all the initial and final plaintext values as pairs as in the rainbow table, an initial plaintext value is stored with multiple output plaintext values. This results in significant storage space savings compared with the rainbow table method. In particular, for the same storage space as the rainbow table method, the enhanced method yields 13.28% to 19.14% improvement in the total number of distinct plaintext-hash pairs generated. However, some passwords "residing" in the first column will not be recovered because they are not stored in the table. This limits the search to $n-1$ chain columns instead of n columns.

3. Rainbow Table Method

We use the following notation in our discussion of the rainbow table method and its extensions: x_i denotes the initial value of a chain; y_i

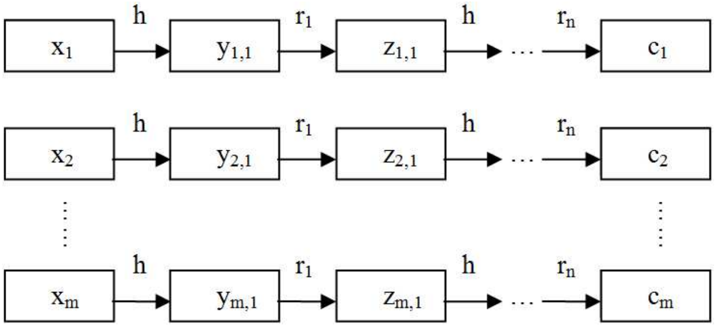

Figure 1. Rainbow table.

denotes the hash value of a password; z_i denotes the reduced value of a hash; c_i denotes the final value of a chain; h denotes a hash function; r_i denotes a reduction function; n denotes the number of reduction functions or chain length; and m denotes the number of chains.

In order to generate a rainbow table, it is necessary to specify reduction functions $r_1, r_2, \ldots, r_n$ that convert hash values y_i to plaintext values z_i. A large set of plaintext values $x_1, x_2, \ldots, x_m$ are then chosen as the initial values of the table. As shown in Figure 1, these plaintext values are alternately hashed using the password hashing algorithm and reduced using the reduction functions.

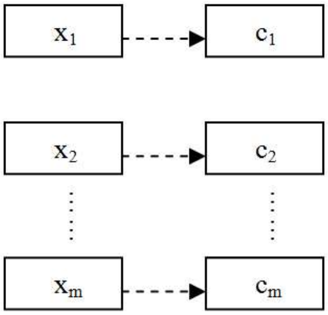

Figure 2. Stored values in a rainbow table.

A rainbow table is created by only storing the final values c_i along with their corresponding initial values x_i (Figure 2). A password is recovered by alternately applying reduction and hashing operations to the corresponding hash values until a value is obtained that matches one of the final values in the rainbow table.

Consider the situation where it is necessary to find the password corresponding to the password hash v. One round of the reduction operation is applied to the password hash v to obtain the plaintext value w. Next,

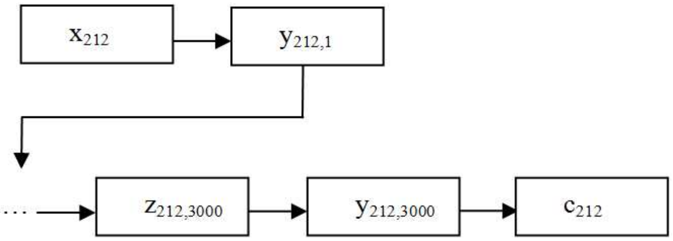

Figure 3. Password recovery example.

a search is performed and a match of w is found at the final value of the 212^{th} chain in the rainbow table in Figure 3 (i.e., $w = c_{212}$). The chain is then computed from its initial value x_{212} until the password hash v is reached, which is equal to $y_{212,3000}$. The password is the plaintext value $z_{212,3000}$, which is before the password hash $y_{212,3000}$.

If no matching value is found, it is assumed that the particular password does not exist in the rainbow table and cannot be recovered. The password recovery success rate can be improved by increasing the number of reduction functions and chains, but this increases the computational complexity and storage requirements.

4. Virtual Expansion of Rainbow Tables Method

Our proposed virtual expansion of rainbow tables (VERT) method virtually expands the rainbow table contents while maintaining the storage space requirements of the original rainbow table method. In VERT, the character set is first remapped to numerical equivalent values using the VERT mapping table.

An example VERT mapping for the alphanumeric character set is shown in Table 1. For a 7-character password, the initial plaintext value in the first chain of the VERT table is selected to be `0000000` and the initial plaintext value in the last chain is selected to be `zzzzzzz`. The initial plaintext values of the remaining chains are chosen from the rest of the password space based on evenly-distributed gaps. The gap size depends on the storage contraint. For example, if the storage space is only sufficient to store four plaintext values, the gap size is:

$$(36^7 - 1)/(4 - 1) = 26121388032$$

after rounding up to the next integer. The four initial plaintext values are `0000000`, `c000001` (computed from $26121388032 = 12(36^6)+1(36^0)$), `n000000` (computed from $2(26121388032) = 52242776064 = 24(36^6)$) and `zzzzzzz`. These initial plaintext values are not stored in the VERT

Table 1. VERT mapping table.

Character Set	Numerical Equivalent
0	0
1	1
2	2
...	...
a	10
b	11
...	...
z	35

table. Instead, only their final plaintext values are stored. Therefore, VERT provides a 100% increase in the number of chains compared with the original rainbow table [11]. Also, it supports password recovery to the first column unlike the enhanced rainbow table method [14].

The VERT method also incorporates an efficient storage mechanism that can support a larger table while using the same amount of storage as the original rainbow table. As seen in Table 1, each character in the VERT mapping table can be represented by six bits. Therefore, the final plaintext values of the chains are converted to their numerical representations before storage. Because eight bits of storage are required for each plaintext character in a rainbow table, the VERT method provides an additional storage conservation of up to 25% and an additional 33.33% increase in the number of chains without increasing the storage requirements.

5. Theoretical Analysis

This section presents a theoretical analysis and comparison of the performance of the VERT, original rainbow table and enhanced rainbow table methods.

5.1 Success Rate without Collisions

The password recovery success rate depends on the number of distinct plaintext-hash pairs generated in the chains, which, in turn, depends on the total number of plaintext-hash pairs generated. First, we perform the analysis ignoring element collisions. Next, we perform an analysis based on the number of distinct pairs and evaluate the effect on the password recovery success rate.

If there are m rainbow chains, each with n reduction functions and requiring storage for two plaintext values (initial value and final value of a chain), then a rainbow table has to store a total of $2m$ plaintext values. Thus, the number of plaintext-hash pairs is mn.

In the case of the enhanced rainbow table method, optimal performance is achieved when a single block is formed. Therefore, using the same storage space as in the original rainbow table, a total of $n(2m - 1)$ plaintext-hash pairs can be generated. This is computed from $2m(n-1) - (n-2) \approx 2mn - n \approx n(2m-1)$ assuming that $n \gg 2$.

The VERT method does not store the input plaintext values. Therefore, a VERT table can generate a total of $2(1.3333m) = 2.67m$ chains using the same storage space as the original rainbow table. Therefore, a VERT table would have $2.67mn$ plaintext-hash pairs. Compared with the original and enhanced rainbow table methods, this translates to 167% and 33% increases in the password recovery success rate, respectively. This success rate is based on the additional plaintext-hash pairs that are generated.

5.2 Success Rate with Distinct Pairs

The password recover success rate computed above ignores collisions. Because the collision probability increases with the size of a rainbow table, ignoring collisions is reasonable only for very small rainbow tables. We compute a more realistic password recovery success rate based on collisions with the distinct plaintext-hash pairs that are generated.

Our analysis assumes that storage space exists for m plaintext values. The password recovery success rate is computed based only on the distinct plaintext-hash pairs. The same number of reduction functions n is used for the original rainbow table, enhanced rainbow table and VERT methods.

Let N be the password space that comprises all possible plaintext passwords and let m_i be the number of distinct plaintext-hash pairs in the i^{th} column of the original rainbow table. Then, m_i and m_{i+1} satisfy the recurrence relation:

$$m_{i+1} = N(1 - (1 - \frac{1}{N})^{m_i})$$

where $m_1 = m$. Thus, the probability of successful password recovery for the original rainbow table method is:

$$P(M) = 1 - (1 - \frac{m_1}{N})(1 - \frac{m_2}{N}) \ldots (1 - \frac{m_n}{N}).$$

Table 2. Success rate based on distinct plaintext-hash pairs.

Storage Size (m)	Original Rainbow Table	Enhanced Rainbow Table	VERT Table
10×10^6	45.07%	65.33%	73.22%
15×10^6	56.94%	76.15%	82.62%
20×10^6	65.33%	82.60%	87.82%
25×10^6	71.50%	86.74%	91.00%
30×10^6	76.15%	89.57%	93.07%

In the case of the enhanced rainbow table method, let s_i be the number of distinct plaintext-hash pairs in the i^{th} column. Then, s_i and s_{i+1} satisfy the following recurrence relation:

$$s_{i+1} = N(1-(1-\frac{1}{N})^{s_i})\ \forall i > 1; s_1 = 1; s_2 = 2m-1.$$

Thus, the probability of successful password recovery for the original rainbow table method is:

$$P(S) = 1-(1-\frac{s_1}{N})(1-\frac{s_2}{N})\dots(1-\frac{s_n}{N}).$$

In the case of the VERT method, let v_i be the number of distinct plaintext-hash pairs in the i^{th} column of the VERT table. Then, v_i and v_{i+1} satisfy the recurrence relation:

$$v_{i+1} = N(1-(1-\frac{1}{N})^{v_i})$$

where $v_1 = 2.67m$. Thus, the probability of successful password recovery for the VERT method is:

$$P(V) = 1-(1-\frac{v_1}{N})(1-\frac{v_2}{N})\dots(1-\frac{v_n}{N}).$$

Note that $v_i > m_i$ for all $i \geq 1$. Thus, $P(V) > P(M)$. In addition, $P(V) > P(S)$ since $v_i > s_i$ for all $i \geq 1$.

The password recovery success rates for the original and enhanced rainbow tables and for the VERT tables for different numbers of stored plaintext values (i.e., storage space) are computed based on the equations presented above. The results are presented in Table 2. The common parameters used in the methods are: (i) number of reduction functions (n): 5,700; (ii) character set: alphanumeric; (iii) plaintext/password length: 1-7 characters; and (iv) storage space (m): same for all methods.

Table 2 shows that the VERT method yields password recovery success rate improvements ranging from 16.92% for storage size $m = 30 \times 10^6$ to 28.15% for storage size $m = 10 \times 10^6$. When compared with the enhanced rainbow table method, the VERT method provides password recovery success rate improvements ranging from 3.50% for $m = 30 \times 10^6$ to 7.89% for $m = 10 \times 10^6$. The results show that even when collisions are considered, the VERT method offers substantial improvements in performance. Note also that the storage size constraint impacts the original rainbow table method much more significantly than the enhanced rainbow table and VERT methods.

6. Experimental Results

This section compares the results obtained with the VERT method and those obtained using RainbowCrack (source code version 1.2) [16].

6.1 Distinct Passwords and Success Rate

To evaluate the password recovery success rate when considering distinct pairs, we made a slight modification to RainbowCrack to log all the plaintext passwords (i.e., the stored initial and final columns as well as the intermediate chain columns). This logging was also performed for the VERT method. We also implemented scripts to detect collisions and count the distinct passwords in the logs.

The experiments were conducted using the following common parameters: (i) number of reduction functions (n): 3,000; (ii) character set: lower case alpha; (iii) plaintext/password length: 1-7 characters; and (iv) storage space (m): 10^6. For fixed $m = 10^6$, this translates to 10^6 chains for the RainbowCrack method and 2.67×10^6 chains for the VERT method.

The theoretical password recovery success rates for RainbowCrack and VERT when considering distinct pairs are 28.13% and 54.31%, respectively. In the experiments, the total plaintext-hash pairs generated were 3×10^9 by RainbowCrack and 8.01×10^9 by VERT. The total plaintext space was 8,353,082,582. A total of 2,349,122,955 distinct passwords were identified among the 3×10^9 plaintext passwords generated by RainbowCrack, corresponding to an actual password recovery success rate of 28.12%. On the other hand, VERT generated 4,536,258,880 distinct passwords, yielding an actual password recovery success rate of 54.31%. Thus, the experimental results match the theoretical results.

Note that the experiments conducted are preliminary in nature due to the scale of collision detection and distinct password count computations. Additional experiments will be performed to study the impact of collisions for larger numbers of chains and reduction functions.

6.2 Computational Complexity

A password is computed by alternatively applying the reduction and hashing operations to the password hash value. However, VERT requires an additional final step of processing (numerical representation conversion) on the last computed plaintext value before it is stored in the table. This conversion is a simple operation; thus, it incurs insignificant computational overhead compared with hashing and reduction.

We conducted experiments on an Intel P4 3.06 GHz system to compute the time taken to perform: (i) random value generation for the initial column for RainbowCrack; (ii) deterministic value generation for the initial column for VERT, (iii) reduction and MD5 hashing operations for RainbowCrack and VERT (used for intermediate column processing); and (iv) single conversion of the final plaintext password to its numerical representation. A total of 10^9 rounds were performed for each operation and the average time for each round was computed. The average computation times were: (i) 1,656 ns; (ii) 7 ns; (iii) 644 ns; and (iv) 211 ns. Note that the final conversion operation only incurs an additional 211 ns for each chain. The initial value generation speed is greatly enhanced in VERT. The total time taken to generate the initial values for RainbowCrack is 1,656m ns while the time taken for VERT is $(2.67 \times 7)m = 18.69m$ ns. However, the main computational overhead is due to the reduction and hashing operations, which require a total of 644mn ns.

7. Conclusions

The novelty of the VERT method lies in the virtual expansion of the pre-computed tables, which increases the password recovery success rate while limiting the storage requirements. Compared with the original rainbow table method, the VERT method increases the password recovery success rate by 16.92% to 28.15% for the distinct pairs comparison while considering collision effects. The VERT method also shows an improvement in the password recovery success rate compared with the enhanced rainbow table method. In particular, the VERT method yields up to 33% improvement for the total plaintext-hash pairs comparison and 3.5% to 7.89% improvement for the distinct pairs comparison. Note also that it is possible to trade-off storage conservation in favor of high password recovery success rates for longer passwords (i.e., larger password spaces).

Our future work related to the VERT method will analyze collisions and password recovery success rates in larger tables. Additionally, we plan to investigate improvements in the password recovery time achieved by reducing the number of columns while maintaining the same storage requirements and password recovery success rates as the original and enhanced rainbow table methods.

References

[1] AccessData, Decryption tools, Lindon, Utah (www.accessdata.com /decryptionTool.html).

[2] Agence France-Presse, Favorite passwords: "1234" and "password," Paris, France, February 11, 2009.

[3] D. Denning, *Cryptography and Data Security*, Addison-Wesley, Reading, Massachusetts, 1982.

[4] H. Dobbertin, A. Bosselaers and B. Preneel, Ripemd-160: A strengthened version of RIPEMD, *Proceedings of the Third International Workshop on Fast Software Encryption*, pp. 71–82, 1996.

[5] M. Hellman, A cryptanalytic time-memory trade-off, *IEEE Transactions on Information Theory*, vol. 26(4), pp. 401–406, 1980.

[6] LCPSoft, LCP, Moscow, Russia (www.lcpsoft.com).

[7] M. Montoro, Cain and Abel (www.oxid.it/cain.html).

[8] National Institute of Standards and Technology, Secure Hash Standard, Federal Information Processing Standards Publication 180-1, Gaithersburg, Maryland, 1995.

[9] National Institute of Standards and Technology, Secure Hash Standard, Federal Information Processing Standards Publication 180-2, Gaithersburg, Maryland, 2002.

[10] Objectif Securite, Ophcrack, Gland, Switzerland (ophcrack.source forge.net).

[11] P. Oechslin, Making a faster cryptanalytic time-memory trade-off, *Proceedings of the Twenty-Third International Cryptology Conference*, pp. 617–630, 2003.

[12] Openwall Project, John the Ripper password cracker (www.open wall.com/john).

[13] R. Rivest, The MD5 Message-Digest Algorithm, IETF RFC 1321, 1992.

[14] V. Thing and H. Ying, A novel time-memory trade-off method for password recovery, *Digital Investigation*, vol. 6(S1), pp. S114–S120, 2009.

[15] D. Todorov, *Mechanics of User Identification and Authentication: Fundamentals of Identity Management*, Auerbach Publications, Boca Raton, Florida, 2007.

[16] S. Zhu, RainbowCrack: The time-memory trade-off hash cracker (project-rainbowcrack.com).