# A Non-Probabilistic Time-Storage Trade-off for Unsalted Hashes

**Frederico Schardong**[1], **Daniel Formolo**[2]

[1] Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brazil

[2]Universidade do Vale do Rio dos Sinos (UNISINOS)
São Leopoldo - RS - Brazil

fschardong@inf.ufrgs.br, danielformolo@unisinos.br

***Abstract.*** *This work proposes a new cryptanalytic non-probabilistic trade-off for unsalted hashes. It presents the main cryptanalytic trade-offs, making a comparison with the proposed method. Although the number of hash operations to recover an element is high compared with the traditional methods, the new method has the advantage of guaranteed success on the recovery of hashes, minimal and sequential disk read operations, unlike the existing probabilistic trade-offs.*

## 1. Introduction

Hash functions are widely used to protect sensitive information such as passwords. Unlike symmetric cryptography where a key is needed to recover the content encrypted by the same key, hash functions are not reversible. This fundamental difference allows the result of a hash function to be stored in an insecure environment, provided that the hash function is considered safe, i.e., does not possess known attacks or flaws. However, if the hash function's input is considered predictable, an attacker that eventually obtain such hash can compare it to hashes that she possesses and know their original content. If any hash matches then the attacker can use it against the target system and easily pretend to be the legitimate user, if the system has no additional authentication mechanism [Stalings 2005].

Concatenating either fixed or random data to the information being hashed before submitting to the hash function provides additional entropy as an attacker that possesses any hash will have to generate the salt value in addition to the password itself. Although salting can mostly overcome trade-off attacks, this technique is often forgotten by developers and analysts. Hash functions such as bcrypt [Provos and Mazieres 1999] and scrypt [Percival and Josefsson 2015] invalidate brute-force and trade-off attacks by internally using salt and large amounts of memory and processing power to compute a single hash. Even though solutions for trade-off attacks are known to the cryptanalytic community for a long time, many developers and analysts are not aware of them and use insecure hash functions with no extra care.

Using a single hash function to protect user passwords might compromise the security of cryptosystems. Such implementations are vulnerable to many attacks: brute-force, dictionary of passwords/words and more recently rainbow table. Both dictionary and rainbow table attacks require some pre-computation so the attack can be applied: a list of words need to be generated and either their hashes calculated or the rainbow table

computed. Currently, the processing power and capacity of storage are large. Even so, these resources are heavily used by attacks, so the balance in the usage of these resources adapting to situations and hardware conditions are essential for breaking cryptographic keys in an acceptable time. This paper presents a new cryptanalytic non-probabilistic trade-off and compares its results to the two most significant cryptanalytic trade-offs. The next section provides a brief description of the traditional methods. Section 3 describes Hellman's tables, the first method to be compared, while Section 4 presents rainbow table, the second method to be compared. Section 5 describes the delta encoding technique and Section 6 the proposed algorithm. Section 7 presents the results of experiments and performance comparison with the other two existing algorithms. Finally, section 8 concludes the study.

## 2. Traditional Attacks

The two main methods for breaking hashes are brute force and dictionary. Both attacks aim to discover the input value applied to a known hash function $H(x)$, which generated a hash $h$, also known. Each of the two approaches has their characteristics regarding the use of computational resources. To better understand these features and how traditional and modern methods balance the use of processing and memory the concept of trade-off was created.

[Hellman 1980] defines trade-off as: "if there are $N$ possible solutions to search over, the time-memory trade-off allows the solution to be found in $T$ operations (time) with $M$ words of memory, provided the time-memory product $TM$ equals $N$". Hellman's trade-off definition allows two extremes: $T = 1$ and $M = N$, or $M = 1$ and $T = N$. Although only one hash operation is needed in the former, the amount of memory required is equal to the dictionary attack ($N$). The latter case defines the brute force attack, where almost no memory is used ($M = 1$) and the number of hash calculations is equal to the number of possible solutions ($T = N$).

Given a hash $h$, one wants to discover the plaintext which generated $h$. Through the brute force attack, a set of plain words is chosen then one or more computers strive to produce this set and simultaneously apply the same hash function $H(x)$ to each word. Each hash generated by the function $H(x)$ is compared with the hash $h$ provided, if they are equal then the plaintext that generated the hash $h$ is the same plaintext that generated the hash $H(x)$ (ignoring the low probability of a collision). If no generated hash is equal to the hash $h$ provided, then the plaintext that resulted in $h$ is not part of the set of plain words chosen at the start of algorithm execution.

The main difference between dictionary and brute force attacks is that hashes are stored along with their respective plaintext in the dictionary attack. This means that the elements from the set of plain words are submitted to the hash function $H(x)$ only once. This dictionary creation process is called pre-processing or pre-computation, and its cost is usually negligible as it is executed only once. After its creation, the dictionary is sorted by hashes so that it is possible to find any element in up to $log_2(N)$ operations, where $N$ is the number of elements in the dictionary. [Sleator and Tarjan 1985].

## 3. Hellman's Method

To demonstrate that valuable information should not be entrusted to systems with short keys, [Hellman 1980] proposed a trade-off able to reverse an encrypted element of a system with $N$ elements using $N^{2/3}$ operations and $N^{2/3}$ words of memory. At the time it was created, Hellman was able to reduce the cost of breaking DES on block mode from $5'000$ USD dollars using brute force to $10$ USD dollars using his method (not considering the cost of the precomputation required to create the tables).

Given a set of $N$ elements to be stored in this system, $m$ elements are randomly selected from the set $\{1, 2, ..., N\}$, creating starting points $SP_1, SP_2, ..., SP_m$ obtaining:

$$X_{i0} = SP_i \qquad 1 \leq i \leq m \tag{1}$$

Then computes:

$$X_{ij} = f(X_{ij-1}) \qquad 1 \leq j \leq t \tag{2}$$

$t$ being the number of rounds that the function $f(x)$ is executed. After $t$ executions of function $f(x)$ the last element of a column $i$ will be:

$$EP_i = f^t(SP_i) \tag{3}$$

Function $f(x)$ is the result of some reduction function $r(x)$ applied to an encrypted element, thus $f(x) = r(H(x))$. The reduction function $r(x)$ must reduce the output of a hash function or symmetric encryption in such a way that the result is a random element of the same set $N$.

To reduce the amount of memory all the intermediate points are removed and $\{SP_i, EP_i\}_{i=1}^{m}$ are organized by the end points ($EP$) then stored in a table. Multiple tables can be created where each table must use a different reduction function to avoid collisions.

Given an encrypted element or some hash $K$ to be searched, $Y_1 = r(K)$ is computed and searched in the end points of the table. If it is not found, it means that the plaintext which generated $K$ is no $X_{i,t}$. So, $Y_2 = f(Y_1)$ is computed and searched in the table, if not found then the plaintext is not in position $X_{i,t-1}$. Repeating the previous step, $Y_3 = f(Y_2)$ is computed and another search is performed on the table. This process is repeated until some $EP_i$ is found or until $X_{i,0}$ is reached.

Assuming $EP_i$ is equal to $Y_3$, then the plaintext that generated $K$ is at the position $X_{i,t-2}$. However, since all the intermediate points were discarded, the entire line (starting on $SP_i$) needs to be recreated, so the element in position $X_{i,t-2}$ can be found and encrypted to test if it is equal to $K$. If different, a false alarm has been found and the algorithm execution continues. A false alarm is a consequence of the probabilistic nature of this method. As the result of encryption or hash function is shortened, the probability of false alarms (collisions) may be significant depending on the settings used.

Hellman states that if the function $f$ is modeled as a function that randomly maps elements of $N$ to itself then the probability of success of a table is limited by:

$$P(S) \geq (1/N) \sum_{i=1}^{m} \sum_{j=0}^{t-1} [(N - it)/N]^{j+1} \tag{4}$$

The mathematical proof of this theorem is found on Hellman's paper [Hellman 1980]. He also states that if $mt^2 = N$ then the resolution of equation 4 results in a probability of success of at least $80\%$, and the expected number of false alarms per table is:

$$E(F) \leq mt(t+1)/2N \tag{5}$$

The use of Hellman's method to break the 56-bit Data Encryption Standard, or DES, is less complex than applying brute force on a system with 38-bit key [Hellman 1980].

## 4. Rainbow Table

[Oechslin 2003] proposes to change Hellman's table structure, making it use different reduction functions rather than just one. In Hellman's method each shortening round uses the same function $r(x)$, while in rainbow table each round uses a shortening function $r_i(x)$ different. First round always uses shortening function $r_1(x)$, the second $r_2(x)$ and so on until the last round $t$ use the reduction function $r_{t-1}(x)$.

To recover a value $K$ from a rainbow table, $r_{t-1}(K)$ is applied first. If the result is not present in the end points of the table then $r_{t-2}(f_{t-1}(K))$ is applied, if this result is not present in the end points of the table then the next step is to apply $r_{t-3}(f_{t-2}(f_{t-1}(K)))$ and so on. Retrieving an element in a rainbow table need at most $t(t^2 - 1)/2$ operations whereas searching on the original table need at most $t^2$ operations, considering that both tables have $t$ columns.

Oechslin created $4'666$ Hellman's tables with dimensions of $m_h = 8'192$, $t_h = 4'666$ and a rainbow table with dimensions $t_r = 4'666$, $m = m_h * t_h = 38'223'872$ both using the same set $N = 2^{37}$ of 8 byte hashes. In his implementation the probability of success was measured by searching $500$ random hashes on each scheme and calculated to be $78,8\%$ on the rainbow table and $75,8\%$ on Hellman's method. Search for elements in rainbow tables was 7 times faster with an average of $9,3$ million hash calculations, against $67,2$ million on Hellman's tables.

[Zhang et al. 2010] propose to divide rainbow table into smaller pieces and distribute them among computers in a client-server relationship such as each customer receives a different part of the table and can contribute to search keys, turning the problem from two dimensions $\{M, T\}$ into a three-dimensional problem $\{M, T, R\}$. The protocol that coordinates the work between each client and the server consists of synchronous calls that do not represent a significant cost. Implementation confirms the theoretical analysis made by the authors that attack time decreases linearly with the increase of clients (resources).

## 5. Delta Encoding

Delta encoding is a data compression technique that consists in calculating the difference between two objects, whether they are files, numbers, strings or something else. There are

numerous applications that benefit from this simple technique (complexity $O(n)$) such as incremental backup [Burns and Long 1997], deduplication systems [Paulo and Pereira 2014] and HTTP frame compression [Mogul et al. 1997].

According to [Mogul et al. 1997] video compression standard MPEG-1 already used the technique of calculating the difference between objects. The MPEG-1 occasionally sends an entirely new frame, and next frames are just information to reconstruct what has changed compared to previous frames.

Naturally the smaller difference between the data, the higher is the performance of the delta encoding. For example, the set $\alpha = \{2, 5, 7, 4, -3, 8, 23, 9, 4\}$ results in $\alpha_d = \{2, 3, 2, -3, -7, 11, 15, -14, -5\}$ such as $\alpha_d = D(\alpha)$ where $D(x)$ is a function that calculates the delta encoding of $x$. The $\alpha$ set needs at least $54$ bits to be stored as the largest element $(23)$ occupies $6$ bits ($5$ bits for the number plus the sign indicator), so $9 * 6 = 54$ bits. On the other hand, the delta encoded set ($\alpha_d$) needs only $45$ bits.

## 6. Proposed Algorithm

In this section, the operation, benefits, and limitations of the proposed algorithm are described in detail. This algorithm consists of applying a round of delta encoding then sorting the resulting elements to remove all duplicate. Through this technique, a new trade-off is created as more duplicate elements are removed, more space is saved, however, more processing is required to recover the original set. The resulting ordered set may undergo another round of delta encoding or other compression technique chosen by the analyst as, for example, bitmap [Lemire et al. 2010].

### 6.1. Delta Encoding of Complexity $O(n^2)$

Given the set $\alpha = \{3, 12, 20, 21, 30\}$ the result of $\alpha_d = D(\alpha)$ is the set $\{3, 9, 8, 1, 9\}$. As explained earlier, it is possible to reassemble $\alpha$ with complexity $O(n)$. Ordering $\alpha_d$ in ascending order through any $S(x)$ function results in $\alpha_s = S(\alpha_d)$ such as $\alpha_s = \{1, 3, 8, 9, 9\}$. Removing duplicate numbers from $\alpha_s$ results in $\beta = \{1, 3, 8, 9\}$ which can again be subjected to the delta encoding function $D(\{1, 3, 8, 9\})$ creating the set $\beta_d = \{1, 2, 5, 1\}$. Set $\beta_d$ must start with the first element of $\alpha$, resulting in $\beta_d = \{3, 1, 2, 5, 1\}$. Set $\beta_d$ can be saved in only $15$ bits as each element occupies $3$ bits while the original set needs $25$ bits, disregarding any compression algorithm.

Recovery of set $\beta$ from $\beta_d$ can be performed with complexity $O(n)$, requiring $4$ operations (ignoring the first element of $\beta_d$ as it is the first element of $\alpha$), obtaining $\beta = \{1, 3, 8, 9\}$. After adding the first element of $\beta_d$ to $\beta$ it is obtained $\beta = \{3, 1, 3, 8, 9\}$. Recovery of set $\alpha$ consists in a problem of higher complexity. One way to recover $\alpha$ would be to select the first element of $\beta$, which is the first element of $\alpha$, iterate the set $\beta$ and at each iteration add the first element of $\alpha$ to each element of $\beta$. Thus, in some interaction, the second element of $\alpha$ is found. When the next element of $\alpha$ is found then the search resumes, summing all elements of $\beta$ to the last element of the set $\alpha$ found. Therefore, all elements of the set $\alpha$ will be found, and the set will be completely recovered.

However, there must be a way to validate whether an element of $\alpha$ was found or not. Evidently, such a method should not be based on the elements of the original set but on some heuristic.

One solution to this problem is to create a function $m(x)$ capable of determining whether an element belongs to $\alpha$ or not. Theorem 1 details the operation of the algorithm using such function $m(x)$, below the theorem, there is a proof confirming that the method described can fully recover $\alpha$.

**Theorem 1** *Given a set $\alpha$ of ordered elements and applying the reduction operations $D()$, ordering $S()$, removal and reduction $D()$, in that order, it is possible to retrieve the original $\alpha$ set if, after the last operation the set $\beta_d$ contains, in addition to its elements, the first element of the original set $\alpha$.*

**Axiom 1** *Applying the functions sorting and removal to any set $\alpha_d$, the resulting set $\beta$ maintains a representative of each element of the set $\alpha_d$ and the first element of the set $\alpha$.*

**Axiom 2** *Given the first element of the set $\beta$, the other elements of this set are recoverable from $\beta_d$ in a simple and obvious way.*

**Proof.**

1. Base: The element $\alpha_1$ is in the set $\beta$.
2. Step: Based on axioms 1 and 2, given that $\alpha_n$ is known, $\alpha_{n+1} = \alpha_n + k$, being $\alpha_n + k$ the lowest element resulting from $m(\alpha_n + k) = true$, for all $k \in \beta$.
3. Restriction: The set $\alpha$ is sorted and does not have duplicate elements.

Creating a function $m(x)$ capable of determining whether an element is part of the set $\alpha$ or not can be tough. One possible way of solving this problem is separating the elements of the original set $\alpha$ into smaller subsets through some heuristic, such as a function that maps each element to some subset at random. Such $m(x)$ function should always map an element $x$ for a subset $\alpha_a$, never to another.

Considering that the function $m(x)$ maps elements of a set $\alpha = \{3, 12, 20, 21, 30\}$ into 2 subsets $\alpha_a = \{3, 12, 21\}$ and $\alpha_b = \{20, 30\}$. Applying the algorithm described above in the subset $\alpha_a$ it is obtained $\alpha_{ad} = \{3, 9, 9\}$, $\alpha_{as} = \{3, 9, 9\}$, $\beta_a = \{3, 9\}$ and finally $\beta_{ad} = \{3, 6\}$. Note that the first element of $\beta_{ad}$ is equal to the first element of $\alpha_a$, thus it has not been duplicated.

As noted earlier, it is trivial to recover $\beta_a$, but the recovery of $\alpha_a$ requires some heuristic defined in the function $m(x)$. The first element of $\beta_a$ is always the first element of $\alpha_a$, sum up the last element of the original set found (in the first round is $\alpha_a[0]$) with first of the list, 3, obtaining 6. Applying $m(6)$ it turns out that 6 is not part of the set $\alpha_a$. The last element of the original set found is then added to the second element of the set $\beta_a$, obtaining 12. Applying $m(12)$ returns the subset $a$, indicating that 12 belongs to the original set $\alpha_a$. As a new element of the original set was found, all elements of $\beta_a$ must be added again to the last element of the set $\alpha_a$ found (12) so the third element of the original set can be found. This procedure must be repeated until all elements of the set $\alpha_a$ are recovered.

Disregarding the complexity of function $m(x)$ it is apparent that the complexity of the algorithm which recovers the subsets created from the result of $m(x)$ is $O(n^2)$. The complexity of the generation of subsets is negligible because they are created only once, while the recovery is a task that tends to be repeated multiple times.

### 6.1.1. Considerations regarding the Delta Encoding of Complexity $O(n^2)$

The most important part of the delta encoding algorithm of complexity $O(n^2)$ is the heuristic function that determines the distribution of elements of a set into smaller subsets. This function should be as close as possible to a random function.

Only ordered sets may be used in this algorithm. Another limitation is that it is unable to retrieve sets with duplicate numbers because through the recovery process it is not possible to determine how many times an element exists in the original set, thereby any duplicate element will be lost.

## 6.2. Cryptanalytic Non-Probabilistic Trade-off

Using the delta encoding of complexity $O(n^2)$ algorithm described in the previous section, one can represent numerical sets with many elements in a compact form at the cost of using more processing power. However, only the proposed algorithm can not be considered a trade-off because it is not possible to vary between memory and processing.

One way to make it a trade-off is to change the function $m(x)$ such that a given set of elements can be divided into subsets of different size. In this way, more subsets result in subsets with fewer elements and consequently little processing and lots of memory needed to store them (tending to $M = N$ and $T = 1$). However, fewer subsets result in subsets with more elements, namely very little memory but considerable processing (tending to $T = N$ and $M = 1$). Few subsets lead to low memory use because applying the first step of the algorithm previously described results in many duplicate deltas, which are later removed. On the other hand, many subsets with few elements result in sparse subsets, resulting in less duplicate deltas. Consequently, the total space of the tables will be higher. One can associate the subsets of this method with the tables of Hellman's method.

Since the goal of this algorithm is to store passwords and their hashes, the order of elements is irrelevant. This makes it possible to use delta encoding of complexity $O(n^2)$. Consider $\alpha$ as a set of passwords to be stored. This set must first be sorted and then submitted to the delta encoding algorithm of complexity $O(n^2)$, where $m(x)$ is a reduction function $R(x)$ of the hash function $H(x)$ to be reversed.

It is important to reduce the output of the function $H(x)$ through some reduction function $r(x)$ so that it produces an amount of subsets that satisfies the desired trade-off. For example, a 128-bit hash function would result in $2^{128}$ subsets if no reduction is used. If the number of elements in the set $\alpha$ is less than $2^{128}$ then the trade-off would be a dictionary attack ($M = N$ and $T = 1$).

The subsets of $\alpha$ must be stored along with the identification of which hash and reduction algorithms were used as function $m(x)$.

After all subsets are generated and stored, it is trivial to recover some password only providing its hash, considering that it is in the initial set $\alpha$. To reverse any hash $h$, one has to reduce it to find the subset where its plaintext is stored. After identifying the subset apply the recovery algorithm proposed in the previous section using $r(H(x))$ as $m(x)$. When some element $i$ belonging to the subset $m_{r(h)}$ is found then compare $H(i) = h$, if it is true then $i$ is the plaintext of the hash $h$ produced by function $H(x)$. If the plaintext $i$

**Table 1. Comparison of a set $\alpha$ using a reduction of MD5 as $m(x)$**

| Plaintext | $H(x)$ | $r(H(x))$ |
|---|---|---|
| **1** | **c4ca4238a0b923820dcc509a6f75849b** | **c** |
| **2** | **c81e728d9d4c2f636f067f89cc14862c** | **c** |
| 3 | eccbc87e4b5ce2fe28308fd9f2a7baf3 | e |
| 4 | a87ff679a2f3e71d9181a67b7542122c | a |
| 5 | e4da3b7fbbce2345d7772b0674a318d5 | e |
| 6 | 1679091c5a880faf6fb5e6087eb1b2dc | 1 |
| 7 | 8f14e45fceea167a5a36dedd4bea2543 | 8 |
| **8** | **c9f0f895fb98ab9159f51fd0297e236d** | **c** |
| 9 | 45c48cce2e2d7fbdea1afc51c7c6ad26 | 4 |
| 10 | d3d9446802a44259755d38e6d163e820 | d |
| 11 | 6512bd43d9caa6e02c990b0a82652dca | 6 |
| **12** | **c20ad4d76fe97759aa27a0c99bff6710** | **c** |

is part of the set $\alpha$ then it is guaranteed to be recovered.

The algorithm described is exemplified in Table 1. The Plaintext column represents the set $\alpha$ to be stored, the function $H(x)$ is MD5 [Rivest 1992] and the reduction function $r(x)$ reduces the output of $H(x)$ to its 4 more significant bits or 1 hex character as shown in the third column. In this example, $4$ of the $12$ elements of the set $\alpha$ were reduced to the address $c$: $1, 2, 8, 12$ which after being subjected to delta encoding algorithm of complexity $O(n^2)$ result in: $1, 4, 6$. The address $c$ is composed of the elements $1, 4, 6$.

As a result, the algorithm generates a table relating each $r(H(x))$ to their respective set of elements. In the example of Table 1, $r(H(x)) = c$ points to set $1, 4, 6$. This relationship allows to recover the plaintexts $1, 2, 8, 12$. From this table and a hash $y$ intercepted, a cryptanalyst can recover the plaintext related to $y$ applying $r(y)$ and accessing the list of values encoded with delta encoding algorithm of complexity $O(n^2)$.

As $m(x)$ is a hash function used to separate the set $\alpha$ in subsets then the elements shall be part of an arithmetic progression. If the distance between the elements is not constant, the recovery phase may mistakenly create elements that the hash function will consider as part of the subset being recovered. Despite this limitation, hash functions satisfy the conditions imposed on how the function $m(x)$ should work. Notwithstanding, this restriction is not problematic for this type of application as a cryptanalyst will typically use it with large data sets, often encompassing all possibilities. Such combination would be an arithmetic progression with 1 bit interval between each element.

Considering that previously to the second round of delta encoding the elements are unique and sorted, other compression techniques can be used instead of the second series of delta encoding. A straightforward and efficient technique when the elements are not far between and not repeated is the bitmap [Lemire et al. 2010]. In a bitmap numbers are saved in the respective bits, for example, the sequence $1, 4, 7, 8$ would be saved as $010010011$ where each bit set to $1$ represents a number from the sequence. In this case,

the first bit representing the number $0$ is set to $0$, i.e., the number $0$ isn't part of the sequence. However, the second bit, which represents the number $1$ is set, i.e., $1$ is part of the sequence. The same goes for the next bits, as only the fifth, eighth and ninth bit are set then only these numbers are part of the sequence. Using this technique only $9$ bits were needed to represent a sequence of $4$ elements. If the sequence ended with an enormous number, for example, $1'500$ instead of $8$ then it would take $1'501$ bits to represent them. In such case, the bitmap is clearly not advantageous.

In this study, both delta encoding and bitmap were used as the last round of the delta encoding of complexity $O(n^2)$. When few subsets were used the bitmap technique was more efficient.

### 6.3. Theoretical Analysis

As explained earlier, the proposed delta encoding algorithm that recovers $\alpha$ from $\beta$ has temporal complexity of $O(n^2)$. The pseudo-algorithm 1 deepens even further how the recovery of $\alpha$ from $\beta$ works.

As pointed out by Oechslin, the number of iterations to recover an element in a rainbow table is at most $t(t^2-1)/2$, while at most $t^2$ operations are needed to retrieve an element on Hellman's method [Oechslin 2003], i.e. both have temporal complexity $O(n^2)$.

**Algorithm 1** Pseudo-algorithm to recover $alpha$ from $beta$

```
1:  procedure RECOVER α(β)
2:      α[0] ← β[0]
3:      address ← m(α[0])
4:      i ← 1
5:      for all a in β do
6:          for all b in β do
7:              if address = m(a + b)  &  (a + b) > α[i − 1] then
8:                  α[i] ← a + b
9:                  i ← i + 1
10:                 break
11:             end if
12:         end for
13:     end for
14:     return α
15: end procedure
```

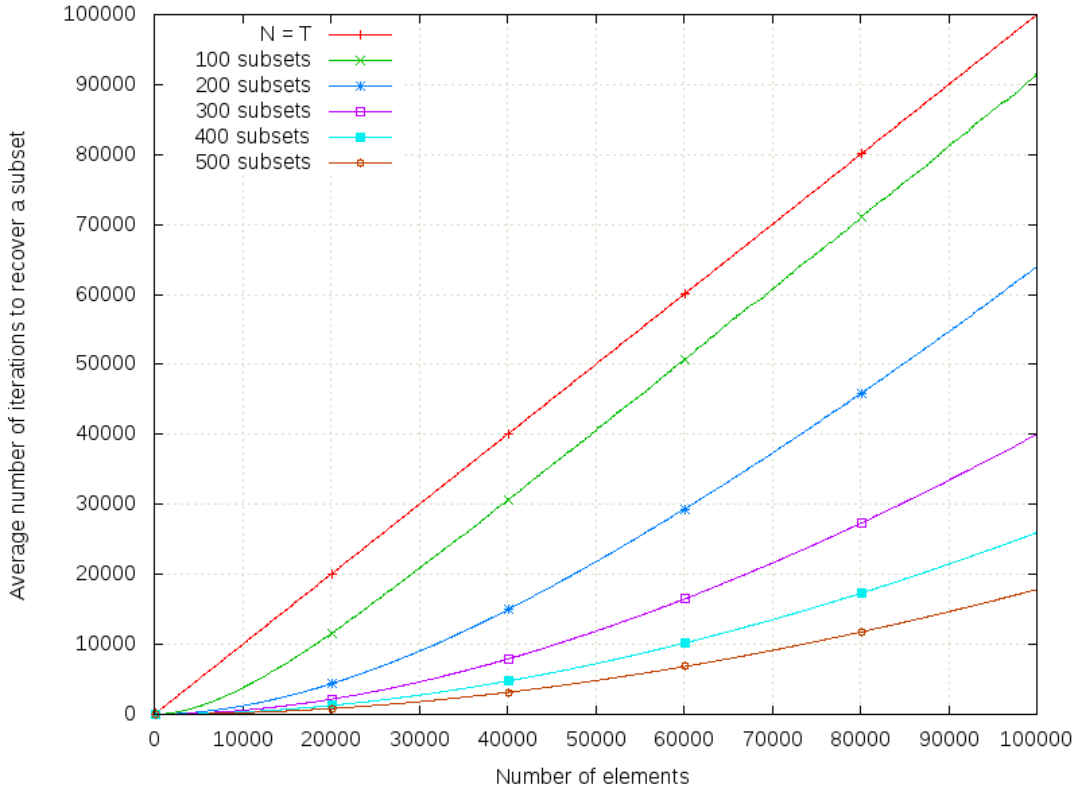
Disconsidering the complexity of the function $m(x)$ it is evident that the complexity of the pseudo-algorithm 1 is $O(n^2)$. The complexity of the function $m(x)$ is not considered in the analysis because it can be implemented in several ways. If using a hash function as implemented in this study, its complexity ($O(1)$) is negligible [Sabharwal and Bratia 1997].

## 7. Experimental Results and Comparisons

To analyze the spatial and temporal behavior of the algorithm, numerical sets ranging between $1'000$ and $100'000$ elements were generated and applied to the cryptanalytic non-

**Figure 1. Average number of iterations to recover a subset**




probabilistic trade-off using multiple subsets (100, 200, 300, 400 and 500). The input data generated by the experiments are irrelevant; any random sequence gives the same results. The graphs of Figures 1 and 2 aim to demonstrate the spatial and temporal behavior of the algorithm, respectively. Both graphs represent the same data sets and subsets. The $X$ axis of both graphs represent the total number of elements while the lines represent each subset. Lines $N = T$ and $N = M$, however, do not represent any subset but the extremes of the trade-off for informational purposes only. The $Y$ axis of the graph of Figure 1 is the average number of iterations (i.e. hash operations) to recover a subset while $Y$ axis of the graph of Figure 2 represents the average number of deltas saved to disk.

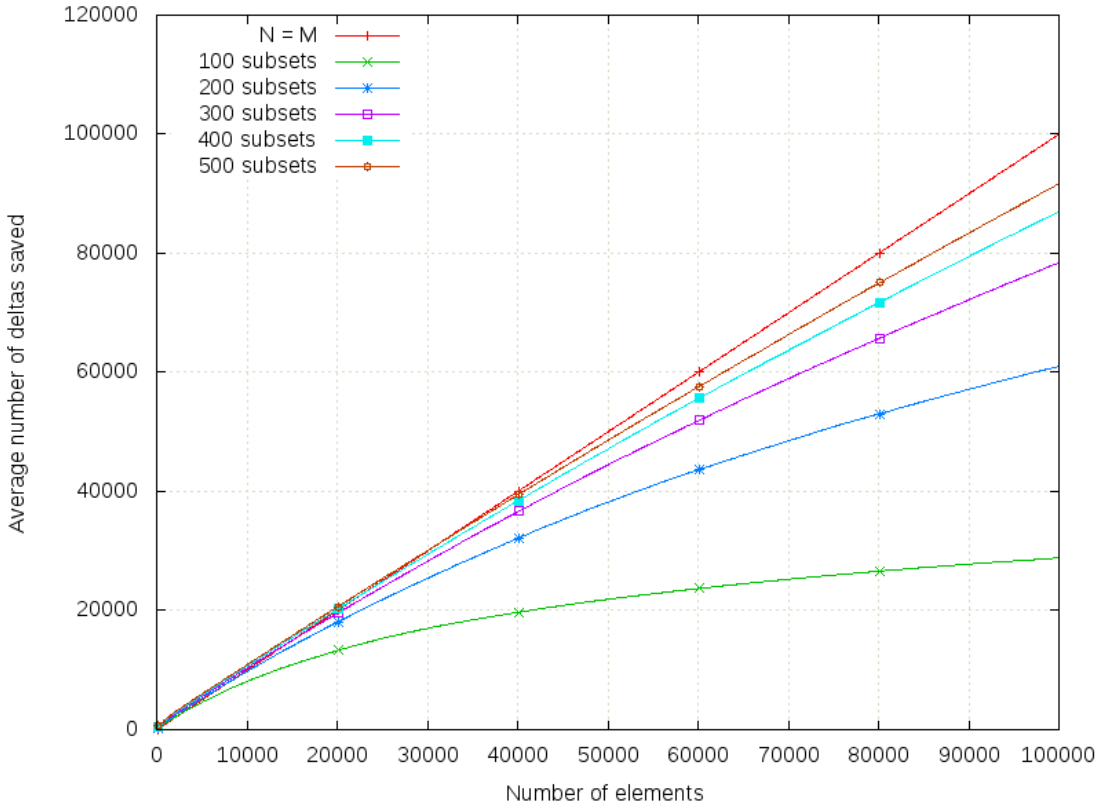
Figure 1 shows that dividing the elements in many subsets (300, 400 and 500) drastically reduces the required number of hash operations (Y-axis) to retrieve the elements of a subset. The fewer subsets are used, i.e., the more elements there are in a subset, more hash operations will be needed to recover it, approximating the trade-off to $N = T$.

When fewer subsets are used, greater is the number of elements in each subset. Consequently, the first round of delta encoding result in more duplicate elements than using more subsets (and therefore fewer elements per subset). Since duplicate deltas are later removed, less memory is required to save all subsets as evidenced in graph of Figure 2. However, if many subsets are used, then few duplicate deltas will be generated causing more deltas to be written to disk, approximating the trade-off to $N = M$.

It is possible to decrease the size of saved deltas through a data compression algorithm. The ZPAQ algorithm [Mahoney 2013] was able to reduce the size of subsets

**Figure 2. Average number of deltas saved in disk**




such that compressed deltas occupy less space than when arranged in $100$ subsets without compression, as can be seen in the graph of Figure 3.

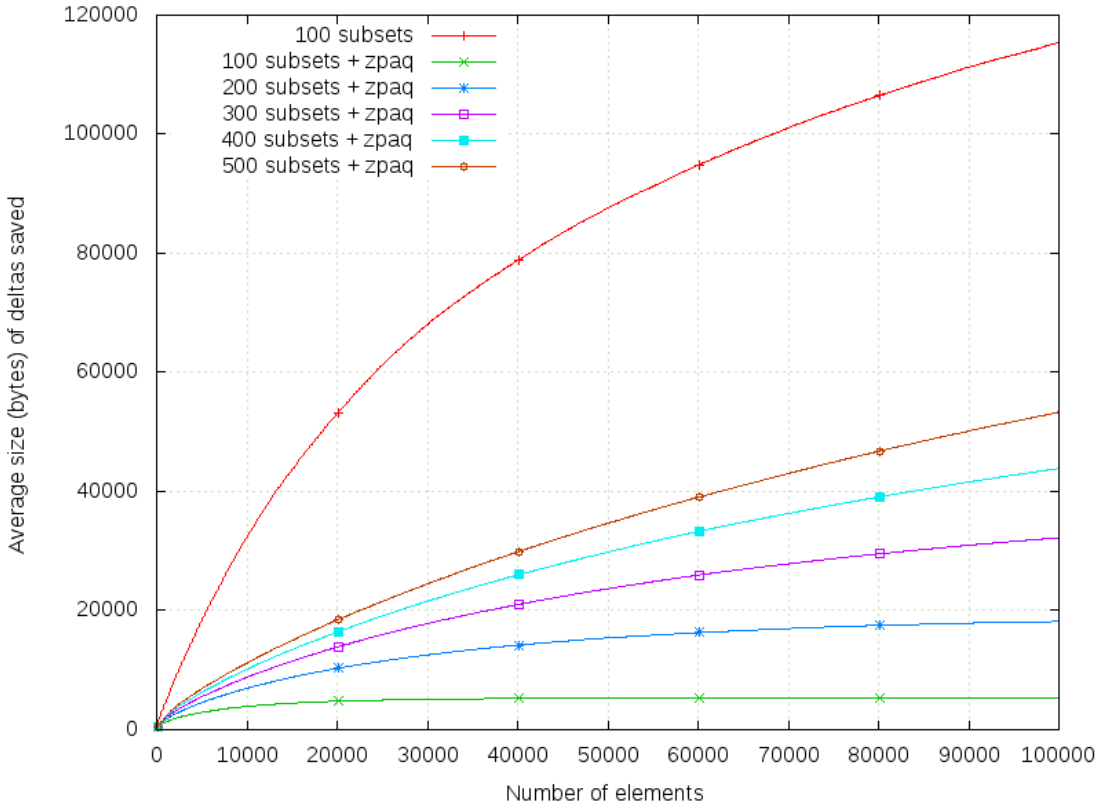
Oechslin created $23'330$ Hellman's tables with dimensions of $m = 7'501$ lines and $t = 4'666$ columns and $5$ rainbow tables with dimensions $t_r = 4'666$ rows and $m = 35'000'000$ columns both using the same set $N$ of $2^{37}$ different hashes of $8$ bytes each. The probability of finding a hash of this group was estimated at $99.9\%$ both in rainbow table and in Hellman's tables. Oechslin recovered $500$ random elements in the rainbow table and found out that it took on average of $7.4$ million hash operations to recover an element versus $90.3$ million using original tables. The two sets of tables were designed to occupy $1.4$ GB each.

Following the criterion of using at most $1.4$ GB, tables of the proposed algorithm were generated to cover the same amount of $2^{37}$ different hashes of $8$ bytes each. $27$ thousand subsets were generated, and $500$ elements were randomly recovered using on average $4$ billion hash operations to retrieve each element. If compression is available, as discussed previously, the number of subsets to fulfill $1.4$ GB would be $60$ thousand subsets. In this scenario each of the $500$ random numbers need on average $3.5$ billion hash operations to be recovered.

Unlike probabilistic methods in which the elements are distributed randomly among all lines of all tables, the proposed algorithm randomly distributes the elements between the tables progressively and steadily, starting from the first element to the last. Resulting in each table being composed of random elements arranged in ascending order, thus

**Figure 3. Average size of subsets after being compressed with ZPAQ**




recovering an element will depend on its position in the table. If an element is at the beginning of a table few hash operations are required to retrieve it, but if it is at the end of the table many hash operations will be necessary.

In the scenario of 27 thousand tables, the number of saved elements in each subset is on average equivalent to $3.09\%$ of the total elements before applying the delta encoding algorithm of complexity $O(n^2)$. On the other hand, the scenario of 60 thousand tables they were equivalent to on average $11.03\%$. Thus, with less duplicate elements less hash operations were required to recover each element. In both 27 and 60 thousand tables scenarios, the last delta encoding operation was replaced by bitmap because on average all sets possessed elements close enough and in sufficient quantity for the technique to be more advantageous than the second round of delta encoding. Only when using more than about 550 thousand subsets the second delta encoding was more efficient than the bitmap technique.

Table 2 summarizes this comparison, showing that although the average number of hash operations required by the proposed method is far superior to Hellman's and Oechslin's methods, the algorithm needs to make fewer read operations on disk to retrieve an element since only the subset to which it belongs must be loaded from disk into RAM at the beginning of execution. Meanwhile, Hellman's and Oechslin's methods need to either load all the tables at once or do random searches on disk when applying the reduction functions to the hash being recovered as it is necessary to search in all lines of all tables based on the result of the reductions. Depending on the architecture in which attacks are

**Table 2. Comparison between the Hellman's and Oechslin's methods with the proposed algorithm**

| | Hellman | Rainbow Table | Proposed Method | Proposed Method with Compression |
|---|---|---|---|---|
| Input Range | $2^{37}$ | $2^{37}$ | $2^{37}$ | $2^{37}$ |
| Size | 1.4 GB | 1.4 GB | 1.4 GB | 1.4 GB |
| Hash Operations | 90.3 M | 7.4 M | 4 B | 3.5 B |
| Advantages | • Few hash operations | | • Little disk access<br>• Disk access is serial<br>• Success is guaranteed<br>• Easily parallelizable | |

conducted and settings chosen for the trade-off, the average execution time of probabilistic methods quoted may be higher than the proposed method.

## 8. Conclusion

Hash functions are widely used to store passwords securely, but the simple application of a hash function may not be enough to protect them. Different attacks can be applied in hashes such as brute force, where passwords are generated, their hashes calculated and compared with the hashes being reverted. If any hash matches, the password that originated this hash is returned. Another possibility is to pre-compute a range of passwords and save them in a structure called dictionary and then search in this database for hashes to be reverted. In brute force attack nothing is stored, all hashes are re-calculated each time the attack is executed. In the dictionary attack, the password range is calculated and hashed only once, without the need to perform repeated hash operations when an attack is applied. On the other hand, hashes are fully saved to disk, consuming a lot of disk space. The brute force attack is one extreme of the trade-off: $M = 1$ and $N = T$ while the dictionary attack is the other extreme: $M = T$ and $N = 1$.

Hellman proposed a probabilistic method able to recover $N$ elements with $N^{2/3}$ operations using $N^{2/3}$ memory words. Oechslin changed Hellman's method to use a different reduction function for each column of the table and obtained practical results of about $12$ times fewer hash operations to retrieve an element in comparison to Hellman's method. Unlike Hellman's and Oechslin's methods, the algorithm proposed in this paper is not probabilistic, ensuring that any element of the range used will be found. This algorithm uses the delta encoding technique of complexity $O(n^2)$, also proposed and analyzed in this paper.

Despite the fact that the average number of hash operations required to retrieve each element of the proposed algorithm is much higher than the results obtained by Hellman and Ochslin, the proposed method only requires to read the subset on disk on which the hash being sought belongs, while Hellman's and Oechslin's methods need to do random reads on their tables. In practice, the high number of hash operations required by the proposed method is somewhat offset by the low amount of disk access. The gain obtained

with low disk access relies heavily on the type of hardware where the algorithm runs, so future studies are needed to evaluate whether in practice the proposed algorithm is more efficient at run time than current methods and which hardware architectures it can obtain speed advantages. In conclusion, the proposed algorithm has advantages over the current methods but also has disadvantages that can be circumvented with future improvements, demonstrating that it has potential for use if improved.

## References

Burns, R. C. and Long, D. D. (1997). Efficient distributed backup with delta compression. In *Proceedings of the fifth workshop on I/O in parallel and distributed systems*, pages 27–36. ACM.

Hellman, M. E. (1980). A cryptanalytic time-memory trade-off. *Information Theory, IEEE Transactions on*, 26(4):401–406.

Lemire, D., Kaser, O., and Aouiche, K. (2010). Sorting improves word-aligned bitmap indexes. *Data & Knowledge Engineering*, 69(1):3–28.

Mahoney, M. (2013). The zpaq open standard format for highly compressed data - level 2. `http://mattmahoney.net/dc/zpaq202.pdf`. [Online; accessed 10-May-2015].

Mogul, J. C., Douglis, F., Feldmann, A., and Krishnamurthy, B. (1997). Potential benefits of delta encoding and data compression for http. In *ACM SIGCOMM Computer Communication Review*, volume 27, pages 181–194. ACM.

Oechslin, P. (2003). Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology-CRYPTO 2003*, pages 617–630. Springer.

Paulo, J. and Pereira, J. (2014). A survey and classification of storage deduplication systems. *ACM Computing Surveys (CSUR)*, 47(1):11.

Percival, C. and Josefsson, S. (2015). The scrypt password-based key derivation function.

Provos, N. and Mazieres, D. (1999). A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track*, pages 81–91.

Rivest, R. (1992). The md5 message-digest algorithm. `https://www.ietf.org/rfc/rfc1321.txt`. [Online; accessed 02-May-2015].

Sabharwal, C. L. and Bratia, S. K. (1997). Image databases and near-perfect hash table. *Pattern Recognition*, 30(11):1867–1876.

Sleator, D. D. and Tarjan, R. E. (1985). Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686.

Stalings, W. (2005). *Cryptography and Network Security*. Upper Saddle River, NJ: Prentice Hall.

Zhang, W., Zhang, M., Liu, Y., and Wang, R. (2010). A new time-memory-resource trade-off method for password recovery. In *Communications and Intelligence Information Security (ICCIIS), 2010 International Conference on*, pages 75–79. IEEE.