On rearrangement hashing with Haskell

Roman Maksimovich

Abstract

In this paper I introduce and develop a mathematical method of producing a cryptographic hash of adjustable length, given a public key and a private key. The hashing is done through encoding selections and permutations with natural numbers, and then composing the hash from a set of source strings with respect to the permutations encoded by the keys. The attempts to construct a suitable integer-to-selection mapping leads to interesting mathematical definitions and statements, which are discussed in this paper and applied to give bounds on the reliability of the hashing algorithm. An implementation is provided in the Haskell programming language (source available at https://github.com/thornoar/password-hash) and applied in the setting of password creation. In the paper, the details of the implementation are discussed, as well as the connections between it and the corresponding mathematical model.

March 25, 2024

Contents

1 Introduction

The motivation behind the topic lies in the management of personal passwords. Nowadays, the average person requires tens of different passwords for different websites and services. Overall, one can distinguish between two ways of managing this set of passwords:

- **Keeping everything in one's head.** This is a method employed by many, yet it inevitably leads to certain risks. First of all, in order to fit the passwords in memory, one will probably make them similar to each other, or at least have them follow a simple pattern like "[shortened name of website]+[fixed phrase]". As a result, if even one password is guessed or leaked, it will be almost trivial to retrieve most of the others, following the pattern. Furthermore, the passwords themselves will tend to be memorable and connected to one's personal life, which will make them easier to guess. There is, after all, a limit to one's imagination.
- **Storing the passwords in a secure location.** Arguably, this is a better method, but there is a natural risk of this location being revealed, or of the passwords being lost, especially if they are stored physically on a piece of paper. Currently, various "password managers" are available, which are software programs that will create and store your passwords for you. It is usually unclear, however, how this software works and whether it can be trusted with one's potentially very sensitive passwords. After all, guessing the password to the password manager is enough to have all the other passwords exposed.

In this paper I suggest a way of doing neither of these things. The user will not know the passwords or have any connection to them whatsoever, and at the same time the passwords will not be stored anywhere, physically or digitally. In this system, every password is a cryptographic hash produced by a fixed hashing algorithm. The algorithm requires two inputs: the public key, i.e. the name of the website or service, and the private key, which is an arbitrary positive integer known only to the user. Every time when retrieving a password, the user will use the keys to re-create it from scratch. Therefore, in order to be reliable, the algorithm must be "pure", i.e. must always return the same output given the same input. Additionally, the algorithm must be robust enough so that, even if a hacker had full access to it and its working, they would still not be able to guess the user's private key or the passwords that it produces. These considerations naturally lead to exploring pure mathematical functions as hashing algorithms and implementing them in a functional programming language such as Haskell.

2 The theory

There are many ways to generate hash strings. In our case, these strings are potential passwords, meaning they should contain lower-case and upper-case letters, as well as numbers and special characters. Instead of somehow deriving such symbol sequences directly from the public and private keys, we will be creating the strings by selecting them from a pre-defined set of distinct elements (i.e. the English alphabet or the digits from 0 to 9) and rearranging them. The keys will play a role in determining the rearrangement scheme. With regard to this strategy, some preliminary definitions are in order.

2.1 Preliminary terminology and notation

Symbols A, B, C will denote arbitrary sets (unless specified otherwise). $\mathbb{N}_0$ is the set of all non-negative integers.

By E we will commonly understand a finite set of distinct elements, called a *source*. When multiple sources E_0, E_1, ..., E_{N-1} are considered, we take none of them to share any elements between each other. In other words, their pair-wise intersections will be assumed to be empty. By $|E|$ we will denote the cardinality of a source E, and $E:i$ will represent its i-th element, with the numeration starting from $i = 0$. On the opposite, the expression $E\,!\,i$ will denote the set difference $E \setminus \{E:i\}$.

The symbol "#" will be used to describe the number of ways to make a combinatorial selection. For example, $\#^m(E)$ is the number of ways to choose m elements from a source E with significant order.

The expression $[A]$ will denote the set of all ordered lists composed from elements of the set A. The subset $[A]_m \subset [A]$ will include only the lists of length m. Extending the notation, we will define $[A_0, A_1, ..., A_{N-1}]$ as the set of lists $\alpha = [a_0, a_1, ..., a_{N-1}]$ of length N where the first element is from A_0, the second from A_1, and so on, until the last one from A_{N-1}. Finally, if $\alpha \in [A]$ and $\beta \in [B]$, the list $\alpha +\!\!+ \beta \in [A \cup B]$ will be the concatenation of lists α and β.

Let $k \in \mathbb{N}_0$, $n \in \mathbb{N}$. The numbers ${}^N k, {}_N k \in \mathbb{N}_0$ are defined to be such that $0 \leqslant {}^N k < N$ and ${}_N k \cdot N + {}^N k = k$. The number ${}^N k$ is the remainder after division by N, and ${}_N k$ is the result of division.

For a number $N \in \mathbb{N}$, the expression (N) will represent the semi-open integer interval from 0 to N: $(N) = \{0, 1, ..., N-1\}$.

Let $n, m \in \mathbb{N}$, $m \leqslant n$. The quantity $n!/(n-m)!$ will be called a *relative factorial* and denoted by $(n \mid m)!$.

If f is a function of many arguments $a_0, ..., a_{n-1}$, the expression $f(a_0, ..., a_{i-1}, -, a_{i+1}, ..., a_{n-1})$ will represent the function of one argument a_i where all others are held constant.

2.2 Enumerating list selections

The defining feature of the public key is that it is either publicly known or at least very easy to guess. Therefore, it should play little role in actually encrypting the information stored in the private key. It exists solely for the purpose of producing different passwords with the same private key. So for now we will forget about it. In this and the following subsection we will focus on the method of mapping a private key $k \in \mathbb{N}_0$ to an ordered selection from a set of sources in an effective and reliable way.

Definition 2.1. Let E be a source, $k \in \mathbb{N}_0$. The *choice function of order 1* is defined as the following one-element list:

$$\mathcal{C}^1(E, k) = [E:{}^{|E|}k].$$

It corresponds to picking one element from the source according to the key. For a fixed source E, the choice function is periodic with a period of $|E|$ and is injective on the interval $(|E|)$ with respect to k. Injectivity is a very important property for a hash function, since it determines the number of keys that produce different outputs. When describing injectivity on intervals, the following definition proves useful:

Definition 2.2. Let A be a finite set and let $f:\mathbb{N}_0 \to A$ be a function. The *spread* of f is defined to be the largest number n such that, for all $k_1, k_2 \in \mathbb{N}_0$, $k_1 \neq k_2$, the following implication holds:

$$f(k_1) = f(k_2) \implies |k_1 - k_2| \geqslant n.$$

This number exists due to A being finite. We will denote this number by $\mathrm{spr}\,(f)$.

Trivially, if $\mathrm{spr}\,(f) \geqslant n$, then f is injective on (n), but the inverse is not always true. Therefore, a lower bound on the spread of a function serves as a guarantee of its injectivity. Furthermore, if $\mathrm{spr}\,(f) \geqslant n$ and f is bijective on (n), then f is periodic with period n and therefore has a spread of exactly n. We leave this as a simple exercise for the reader.

Proposition 2.3. *Let* $f\colon \mathbb{N}_0 \to A$, $g\colon \mathbb{N}_0 \to B$ *be functions such that* $\operatorname{spr}(f) \geqslant n$ *and* $\operatorname{spr}(g) \geqslant m$. *Define the function* $h\colon \mathbb{N}_0 \to [A, B]$ *as follows:*

$$h(k) = [f({}^n k), g({}_n k + T({}^n k))],$$

where $T\colon \mathbb{N}_0 \to \mathbb{N}_0$ *is a fixed function, referred to as the* ***argument shift function.*** *It is then stated that* $\operatorname{spr}(h) \geqslant nm$.

Proof. Assume that $k_1 \neq k_2$ and $h(k_1) = h(k_2)$. Since h returns an ordered list, the equality of lists is equivalent to the equality of all their corresponding elements:

$$\begin{aligned} f({}^n k_1) &= f({}^n k_2), \\ g({}_n k_1 + T({}^n k_1)) &= g({}_n k_2 + T({}^n k_2)). \end{aligned}$$

Since f is injective on (n), we see that ${}^n k_1 = {}^n k_2$. Consequently, it follows from $k_1 \neq k_2$ that ${}_n k_1 \neq {}_n k_2$ and ${}_n k_1 + T({}^n k_1) \neq {}_n k_2 + T({}^n k_2)$. We can then proceed to utilize the definition of spread for the function g:

$$\begin{aligned} |{}_n k_1 + T({}^n k_1) - {}_n k_2 - T({}^n k_2)| &\geqslant m, \\ |{}_n k_1 - {}_n k_2| &\geqslant m, \\ \left| \frac{k_1 - {}^n k_1}{n} - \frac{k_2 - {}^n k_2}{n} \right| &\geqslant m, \\ \left| \frac{k_1 - k_2}{n} \right| &\geqslant m, \\ |k_1 - k_2| &\geqslant nm, \end{aligned}$$

q.e.d. ∎

With this proposition at hand, we have a natural way of extending the definition of the choice function:

Definition 2.4. Let E be a source with cardinality $|E| = n$, $k \in \mathbb{N}_0$, $2 \leqslant m \leqslant n$. The *choice function of order* m is defined recursively as

$$\mathcal{C}^m(E, k) = [E : {}^n k] +\!\!+ \mathcal{C}^{m-1}(E', k'),$$

where $E' = E \,!\, {}^n k$ and $k' = {}_n k + T({}^n k)$, while $T\colon \mathbb{N}_0 \to \mathbb{N}_0$ is a fixed argument shift function.

Proposition 2.5. *Let* E *be a source with cardinality* n. *Then the choice function* $\mathcal{C}^m(E, k)$ *of order* $m \leqslant n$, *as a function of* k, *has a spread of at least* $(n \mid m)!$.

Proof. We will conduct a proof by induction over m. In the base case, $m = 1$, we notice that $(n \mid m)! = n$, and the statement trivially follows from the definition of $\mathcal{C}^1(E, k)$.

Let us assume that the statement is proven for all choice functions of order $m-1$. Under closer inspection it is clear that the definition of $\mathcal{C}^m(E, k)$ follows the scheme given in proposition 2.3, with $\mathcal{C}^1(E, k)$ standing for f and $\mathcal{C}^{m-1}(E', k')$ standing for g. The application of the proposition is not straightforward, and we encourage the reader to consider the caveats. Thus, we can utilize the statement of the proposition as follows:

$$\operatorname{spr}(\mathcal{C}^m(E, -)) \geqslant \operatorname{spr}\left(\mathcal{C}^1(E, -)\right) \cdot \operatorname{spr}\left(\mathcal{C}^{m-1}(E', -)\right) \geqslant n \cdot ((n-1) \mid (m-1))! = (n \mid m)!,$$

q.e.d. ∎

The preceding result is especially valuable considering the fact that there are exactly $(n \mid m)!$ ways to select an ordered sub-list from a list, meaning that $\mathcal{C}^m(E,k)$ is not only injective, but also surjective with respect to k on the interval $((n \mid m)!)$. This makes it a bijection

$$\mathcal{C}^m(E,-)\colon((n \mid m)!) \to [E]_m,$$

and therefore a periodic function with a spread of exactly $(n \mid m)! = \#^m(E)$.

These properties make the choice function a fine candidate for a hash mapping. Suppose that the source E is composed from lower-case and upper-case Latin characters, as well as special symbols and digits:

$$E = \texttt{"qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM0123456789!@\#\$\%"}$$

The choice function gives us a way to enumerate all possible ways to select a sub-list from E. What's more, these selections can be made more "random" and unpredictable by means of complicating the argument shift function T. A reasonable practice is to set $T({}^n k)$ to the ASCII value of the character $E\colon{}^n k$. This way, each chosen character will influence the choice of the next, creating what is called a "chaotic system", where its behavior is fully determined, but even small changes to inputs eventually produce large changes in the output. Here is a little input-output table for the choice function of order 10 with the specified source and shift function:

123	`"41BeGs9$Dd"`
124	`"52NgJfZIk7"`
125	`"63MfHs9$Da"`
126	`"740VbDo6@u"`
127	`"851Br469$S"`

There is, however, a serious problem. This selection method does not guarantee that the chosen 10 symbols will contain lower-case and upper-case characters, as well as digits and spacial symbols, all at the same time. Since the choice function is bijective, there is a key that produces the combination `"djaktpsnei"`, which will not be accepted as a password in many places, because it contains only one category of symbols. Fortunately, there is a solution.

2.3 Elevating the choice function

Definition 2.6. Let $\mathfrak{L}$ be the list of pairs (E_i, m_i), where E_i are sources, $|E_i| = n_i$, $m_i \leqslant n_i$, for $i \in (N)$. The *elevated choice function* corresponding to these data is defined for a key $k \in \mathbb{N}_0$ by means of the following recursion:

$$\overline{\mathcal{C}}(\mathfrak{L},k) = [\mathcal{C}^{m_0}(E_0, {}^{n_0}k)] +\!\!+ \overline{\mathcal{C}}(\mathfrak{L}!\,0,\ {}_{n_0}k + T({}^{n_0}k)),$$

where T is an argument shift function. The base of the recursion is given when $\mathfrak{L}$ is empty, in which case $\overline{\mathcal{C}}([],k) = []$. Otherwise, for every key k, its image is an element of

$$\operatorname{cod}\ \overline{\mathcal{C}}(\mathfrak{L},-) = [[E_0]_{m_0}, [E_1]_{m_1}, ..., [E_N]_{m_N}].$$

In this context, the list $\mathfrak{L}$ will be called a *source configuration*.

In other words, the elevated choice function is a "mapping" of the choice function over a list of sources, it selects a sub-list from every source and then composes the results in a list, which we will call a *multiselection*. A trivial application of proposition 2.3 shows that the spread of $\overline{C}(\mathfrak{L}, -)$ is at least

$$\prod_{i=0}^{N-1} \operatorname{spr}\left(\mathcal{C}^{m_i}(E_i, -)\right) = \prod_{i=0}^{N-1} (n_i \mid m_i)! \tag{1}$$

where E_i, n_i, and m_i compose the configuration $\mathfrak{L}$. In fact, due to the rule of product in combinatorics, we see that the expression in (1) directly corresponds to the number of possible multiselections from $\mathfrak{L}$, or $\#^{\overline{C}}(\mathfrak{L})$ for convenience. Therefore, $\overline{C}(\mathfrak{L}, -)$ is bijective on the interval $\left(\#^{\overline{C}}(\mathfrak{L})\right)$ and periodic with period $\#^{\overline{C}}(\mathfrak{L})$.

This solves the problem with lacking symbol categories — now we can separate upper-case letters, lower-case letters, numbers, etc. into different sources and apply the elevated choice function, specifying the number of symbols from each source. However, there are two issues arising:

- The result of the elevated choice function will be something like `"amwYXT28@!"`, which is not a bad password, but it would be nice to be able to shuffle the individual selections between each other instead of lining them up one after another.
- Despite the fact that the argument shift function makes the password selection chaotic, the function is a bijection, which means that it can be reversed. With sufficient knowledge of the algorithm, a hacker can write an inverse algorithm that retrieves the private key from the resulting password. This is a deal breaker for our function, because it defeats the purpose — you may as well have one password for everything. The way to solve this problem it to make the choice function artificially non-injective, or overlapping, in a controlled way. In such case, many different keys will produce the same password, and it will be impossible to know which one of them is the correct one. This violates the common non-collision property of hash functions, but it is necessary given the nature of the function we are developing.

We will solve one problem at a time.

2.4 The merge function

Proposition 2.7. *Let $f\colon A\times\mathbb{N}_0 \to B$ and $g\colon B\times\mathbb{N}_0 \to C$ be functions such that $\operatorname{spr}(f) \geqslant n$ and $\operatorname{spr}(g) \geqslant m$, where the spread is taken with respect to the second argument. Assume also that g is **absolutely injective** with respect to the first argument, that is,*

$$\forall (b_1, k_1), (b_2, k_2) \in B \times \mathbb{N}_0 : \quad g(b_1, k_1) = g(b_2, k_2) \implies b_1 = b_2.$$

Define the function $h\colon A \times \mathbb{N}_0 \to C$ by

$$h(a, k) = g(f(a, {}^n k), {}_n k + T({}^n k)),$$

where T is an argument shift function. It is stated that $\operatorname{spr}(h) \geqslant nm$ with respect to k.

Proof. Let an element $a \in A$ and let $k_1, k_2 \in \mathbb{N}_0$ be distinct numbers such that $h(a, k_1) = h(a, k_2)$. That implies,

$$g(f(a, {}^n k_1), {}_n k_1 + T({}^n k_1)) = g(f(a, {}^n k_2), {}_n k_2 + T({}^n k_2)).$$

Since g is absolutely injective, we see that $f(a, {}^n k_1) = f(a, {}^n k_2)$, which means that ${}^n k_1 = {}^n k_2$, since f is injective on (n). Now, since the first argument of g in the equation above is the same, we can use the definition of spread for the function g:

$$\begin{aligned} |{}_n k_1 + T({}^n k_1) - {}_n k_2 - T({}^n k_2)| &\geqslant m, \\ |{}_n k_1 - {}_n k_2| &\geqslant m, \\ \left|\frac{k_1 - {}^n k_1}{n} - \frac{k_2 - {}^n k_2}{n}\right| &\geqslant m, \\ \left|\frac{k_1 - k_2}{n}\right| &\geqslant m, \\ |k_1 - k_2| &\geqslant nm, \end{aligned}$$

q.e.d. ∎

Definition 2.8. Let E_1, E_2 be two sources, m_1, m_2 be numbers such that $m_1 \leqslant |E_1|$ and $m_2 \leqslant |E_2|$. Define the *merge function of order 2*,

$$\mathcal{M}^2 \colon [E_1]_{m_1} \times [E_2]_{m_2} \times \mathbb{N}_0 \to [E_1 \cup E_2]_{m_1+m_2},$$

with the following recursive procedure:

For $\alpha \in [E_1]_{m_1},\ \beta \in [E_2]_{m_2},\ k \in \mathbb{N}_0$ consider two cases:

1. Either α or β is empty, that is, $m_1 = 0$ or $m_2 = 0$. Then set $\mathcal{M}^2(\alpha, \beta, k)$ to be equal to $\alpha +\!\!+ \beta$.

2. Neither α nor β are empty. Then we will assume that the merge function is already defined for $(\alpha\,!\,0,\ \beta,\ -)$ and $(\alpha,\ \beta\,!\,0,\ -)$. Let s_1 be the spread of the function $\mathcal{M}^2(\alpha\,!\,0,\ \beta,\ -)$ and s_2 be the spread of $\mathcal{M}^2(\alpha,\ \beta\,!\,0,\ -)$. Finally, denote the remainder ${}^{(s_1+s_2)}k$ by k'. The merge of α and β with key k is defined as

$$\mathcal{M}^2(\alpha, \beta, k) = \begin{cases} [\alpha : 0] +\!\!+ \mathcal{M}^2(\alpha\,!\,0,\ \beta,\ k' + T(k')), & k' < s_1, \\ [\beta : 0] +\!\!+ \mathcal{M}^2(\alpha,\ \beta\,!\,0,\ k' + T(k')), & \text{otherwise}, \end{cases}$$

where T is an argument shift function.

The merge function takes two lists and combines them together in one, in such a way that the order of elements in each of the two lists is not disturbed. For example, the merge of $[1, 2, 3]$ and $[a, b, c]$ with a certain key could be $[1, a, b, 2, c, 3]$. We will now derive some properties of $\mathcal{M}^2$. If s_1 and s_2 are what they are in the above definition, we immediately see that $\mathcal{M}^2(\alpha, \beta, -)$ is periodic with period $s_1 + s_2$, since it depends only on $k' = {}^{(s_1+s_2)}k$. Moreover, it is clear from the definition that $\mathcal{M}^2(\alpha, \beta, -)$ is injective on the interval $(s_1 + s_2)$, which means that its spread is equal exactly to $s_1 + s_2$:

$$\operatorname{spr}\left(\mathcal{M}^2(\alpha, \beta, -)\right) = \operatorname{spr}\left(\mathcal{M}^2(\alpha\,!\,0,\ \beta,\ -)\right) + \operatorname{spr}\left(\mathcal{M}^2(\alpha,\ \beta\,!\,0,\ -)\right). \tag{2}$$

From this recursive relationship we can derive this useful proposition:

Proposition 2.9. *Let α and β be elements of $[E_1]_{m_1}$ and $[E_2]_{m_2}$ respectively. Then the spread of the corresponding merge function, with respect to the key k, is equal to $(m_1 + m_2)!/(m_1! \cdot m_2!)$.*

Proof. We will conduct a proof by induction over the sum $m_1 + m_2$. The base case is provided by the situation when either m_1 or m_2 is zero. Now assume that $m_1, m_2 \neq 0$ and for all similar pairs with sum $m_1 + m_2 - 1$ the statement is proven. Recall equation 2, which we can now transform due to the induction hypothesis:

$$\operatorname{spr}\left(\mathcal{M}^2(\alpha,\beta,-)\right) = \frac{((m_1-1)+m_2)!}{(m_1-1)!\cdot m_2!} + \frac{(m_1+(m_2-1))!}{m_1!\cdot(m_2-1)!} =$$

$$= \frac{m_1\cdot(m_1+m_2-1)! + m_2\cdot(m_1+m_2-1)!}{m_1!\cdot m_2!} = \frac{(m_1+m_2)!}{m_1!\cdot m_2!},$$

voila. ∎

Using some combinatorial logic, we can see that the number $(m_1 + m_2)!/(m_1! \cdot m_2!)$ corresponds to the number of all possible ways to merge the lists α and β, denoted by $\#^{\mathcal{M}}(\alpha,\beta)$. It means that the function $\mathcal{M}^2(\alpha,\beta,-)$ is in fact surjective, and therefore bijective, on the interval from zero to its spread. Now we will, as usual, expand its definition beyond only two lists.

Definition 2.10. Let $\overline{\alpha} = [\alpha_0, \alpha_1, ..., \alpha_{N-1}]$ be a list of lists, where $N \geqslant 2$ and $\alpha_i \in [E_i]_{m_i}$. Define the *merge function of order* N recursively as follows:

$$\mathcal{M}^N(\overline{\alpha},k) = \begin{cases} \mathcal{M}^2(\alpha_0,\alpha_1,k), & N=2, \\ \mathcal{M}^2(\alpha_0,\ \mathcal{M}^{N-1}(\overline{\alpha}!\,0,\ {}^sk),\ {}_sk + T({}^sk)), & N>2, \end{cases}$$

where s is the spread of the function $\mathcal{M}^{N-1}(\overline{\alpha}!\,0,\ -)$, and T is an argument shift function.

Exercise 2.11. *Using induction and proposition 2.7, prove the following result for the spread of the merge function:*

$$\operatorname{spr}\left(\mathcal{M}^N(\overline{\alpha},-)\right) = \frac{\left(\sum_{i=0}^{N-1} m_i\right)!}{\prod_{i=0}^{N-1} m_i!}, \tag{3}$$

where $\overline{\alpha} = [\alpha_0, \alpha_1, ..., \alpha_{N-1}] \in [[E_0]_{m_0}, [E_1]_{m_1}, ..., [E_{N-1}]_{m_{N-1}}]$.

2.5 Merging the multiselection

Definition 2.12. Let $\mathfrak{L}$ be a source configuration of pairs (E_i, m_i) for $i \in (N)$, and let k be a number from $\mathbb{N}_0$. We define the *merged choice function*, corresponding to $\mathfrak{L}$, as follows:

$$\mathcal{MC}(\mathfrak{L},k) = \mathcal{M}^N\left(\overline{\mathcal{C}}(\mathfrak{L},{}^nk),\ {}_nk + T({}^nk)\right),$$

where n is the spread of the elevated choice function $\overline{\mathcal{C}}(\mathfrak{L},-)$, while T is a fixed argument shift function.

In other words, the merged choice function selects N lists from the configuration via $\overline{\mathcal{C}}$, and then merges them together using $\mathcal{M}^N$. We have therefore solved our first problem — the password resulting from this new hash function will have different categories of characters mixed together. What is more important, there is no information lost in the process, which is illustrated by the following proposition:

Proposition 2.13. *Let* $\mathfrak{L}$ *be a configuration of length* N*. The following lower bound takes place for the spread of the merged choice function:*

$$\operatorname{spr}\left(\mathcal{MC}(\mathfrak{L},-)\right) \geqslant \operatorname{spr}\left(\overline{\mathcal{C}}(\mathfrak{L},-)\right)\cdot\operatorname{spr}\left(\mathcal{M}^N(\overline{\alpha},-)\right) = \prod_{i=0}^{N-1}(n_i \mid m_i)!\cdot\frac{\left(\sum_{i=0}^{N-1} m_i\right)!}{\prod_{i=0}^{N-1} m_i!}, \tag{4}$$

where $\overline{\alpha}$ *is the multiselection arising from the application of* $\overline{\mathcal{C}}$*, while* n_i *are the lengths of sources* E_i *in the configuration* $\mathfrak{L}$*.*

Proof. Since the merge function $\mathcal{M}^N$ preserves the order of the lists it merges, we can see that it is absolutely injective with respect to its first argument, $\overline{\alpha}$. Indeed, no matter how the multiselection $\overline{\alpha}$ is merged, all elements of, say, E_0, can be read from the resulting merged list in the order that they were originally. In other words, the list $\alpha_0 \in \overline{\alpha}$ can be reconstructed from the output of $\mathcal{M}^N(\overline{\alpha}, k)$ for all k. And so can α_1, α_2, etc. Here we are actively using the fact that all elements across all sources $E_0, E_1, ..., E_{N-1}$ are distinct. Now that the absolute injectivity of $\mathcal{M}^N$ has been established, the present statement immediately follows from an application of proposition 2.7. ∎

It can once again be shown using combinatorics, that the expression in (4) is, in fact, the total number of ways to choose a multiselection from $\mathfrak{L}$ and merge it. Therefore, we conclude that the function $\mathcal{MC}(\mathfrak{L}, -)$ is bijective on the interval from zero to its spread, and therefore periodic with the period of its spread.

2.6 Double encryption

However, we still have one problem left: our current hash function is bijective, and it can be reverse-engineered relatively easily.