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 23, 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:

1. **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.

2. **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 recreate 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_1, E_2, ..., E_N 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 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 $[A_1, A_2, ..., A_N]$ as the set of lists $\alpha = [a_1, a_2, ..., a_N]$ of length N where the first element is from A_1, the second from A_2, and so on, until the last one from A_N. Finally, if $\alpha \in [A]$ and $\beta \in [B]$, the list $\alpha \mathbin{+\!\!+} \beta \in [A \cup B]$ 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 $(n \mid m)!$.

2.2 The choice function

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 hashing 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 $f\colon \mathbb{N}_0 \to A$ 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 $\operatorname{spr}(f)$.

Trivially, if $\operatorname{spr}(f) \geqslant n$, 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 $\operatorname{spr}(f) \geqslant n$ 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:

$$f({}^n k_1) = f({}^n k_2), \tag{1}$$

$$g({}_n k_1 + T({}^n k_1)) = g({}_n k_2 + T({}^n k_2)). \tag{2}$$

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:

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

■

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

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

where $T\colon \mathbb{N}_0 \to \mathbb{N}_0$ 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 $\mathcal{C}^1$.

Let us assume that the statement is proven for choice functions of order $m - 1$. Under closer inspection it is clear that the definition of $\mathcal{C}(k, E, m)$ follows the scheme given in proposition 2.3, with $\mathcal{C}^1(E, -)$ standing for f and $\mathcal{C}^{m-1}(E, -)$ standing for g. Thus we can utilize the statement of the proposition:

$$spr$$

■