**Hashes**

Currently, almost no cryptography application is complete without the use of hashing.

Hash functions are functions designed to "compress" an arbitrary message or set of data written, usually in a binary alphabet, into some fixed–length bit combination called convolution. Hash functions have a variety of applications in conducting statistical experiments, testing logical devices, and building algorithms for fast search and checking the integrity of records in databases. The main requirement for hash functions is the uniformity of the distribution of their values when randomly selecting argument values.

A cryptographic hash function is any hash function that is cryptographically secure, that is, it satisfies a number of requirements specific to cryptographic applications. In cryptography, hash functions are used to solve the following tasks:

- building data integrity control systems during transmission or storage,

- authentication of the data source.  
  
A hash function is any function h:X -> Y that is easily computable and such that for any message M, the value h(M) = H (convolution) has a fixed bit length. X is the set of all messages, Y is the set of binary vectors of fixed length.

As a rule, hash functions are built on the basis of the so-called one—step compression functions y = f(x1, x2) of two variables, where x1, x2 and y are binary vectors of length m, n and n, respectively, and n is the length of the convolution, and m is the length of the message block.

To obtain the value of h(M), the message is first divided into blocks of length m (in this case, if the length of the message is not a multiple of m, then the last block is supplemented to the full in some special way), and then the following sequential convolution calculation procedure is applied to the resulting blocks M1, M2,.., MN:

*Ho = v,  
Hi = f(Mi,Hi-1), i = 1,.., N,  
h(M) = HN*

Here *v* is some constant, it is often called the initializing vector. It is chosen for various reasons and can be a secret constant or a set of random data (a sample of date and time, for example).

With this approach, the properties of the hash function are completely determined by the properties of the one-step compression function.

There are two important types of cryptographic hash functions - key and keyless. Key hash functions are called message authentication codes. They make it possible, without additional means, to guarantee both the correctness of the data source and the integrity of data in systems with users who trust each other.

Keyless hash functions are called error detection codes. They make it possible with the help of additional means (encryption, for example) to ensure the integrity of the data. These hash functions can be used in systems with both trusting and non-trusting users.

As I have already said, the main requirement for hash functions is the uniformity of the distribution of their values when randomly selecting argument values. For cryptographic hash functions, it is also important that at the slightest change in the argument, the value of the function changes greatly. This is called the avalanche effect.

The following requirements are imposed on the key hashing functions:

* the impossibility of fabrication,
* the impossibility of modification.

The first requirement means that it is very difficult to select a message with the correct convolution value. The second is the high complexity of selecting another message with the correct convolution value for a given message with a known convolution value.

Keyless functions are subject to the following requirements:

- unidirectionality,

- resistance to collisions,

- resistance to finding a second prototype.

Unidirectionality is understood as the high complexity of finding a message by a given convolution value. It should be noted that at the moment there are no hash functions in use with proven unidirectionality.

Collision resistance refers to the difficulty of finding a pair of messages with the same convolution values. Usually, it is the finding of a way for cryptanalysts to build collisions that serves as the first signal of the algorithm's obsolescence and the need for its rapid replacement.

Resistance to finding a second prototype is understood as the difficulty of finding a second message with the same convolution value for a given message with a known convolution value.

***Hash functions that have been reviewed***

I have considered such hash functions as:

*SHA-1 , FarmHash-32bit, FarmHash-64bit, FarmHash-128bit, Murmur3-32bit, FarmHash-64bit, FarmHash-128bit, XXHash-32bit, XXHash-64bit, XXHash-128bit, std::hash.*

I compared their performance on different file sizes. Both with and without compiler optimization.

As a result, FarmHash works faster on small files, xxHash algorithm has a large margin on large files.

The tests were carried out on the size files: 10,50,100,500,1k,4k,8k,16k,32k,64k.