Kxmash : A Novel Parallel Cascading Multihash Algorithm

Abstract

This short paper presents Kxmash (read this as 'keeexmash' for instance) : a novel way of combining multiple hashing algorithms in a single "parallel cascading" hashing function that we believe improves over simpler schemes. The algorithm remains simple and does not impact performance compared to parallel multihashing.

It is used in new V2 implementation of the KeeeX metadata, with a default combining sha2 and sha3 in an attempt to secure hash computation for decades.

Authors: Laurent Henocque and Gabriel Risterucci

Introduction

Hash functions[hash] are functions that convert an input of arbitrary size into a digital fingerprint of fixed size. These fingerprints are mainly used for two purposes: integrity calculation, and substitution in other algorithms requiring fixed or limited-size inputs, such as digital signatures.

One of the key properties of hash functions is their irreversibility, and the significant impact of a minor modification of the input on the construction of the output.

The term cryptographic hash function[cryptohash] is used for hash functions whose properties do not allow the calculation to be reversed more easily than by brute force.

Examples of applications of hashes

In particular, hash functions can be used to substitute data of arbitrary size with data of limited size (32 bytes for SHA2-256) with a high level of confidence in the uniqueness of this fingerprint. This substitution makes it possible to manipulate fingerprints very simply, in the same way as if you were manipulating the initial data in certain cases.

Here are a few examples:

- the git[git] version management system, which enables all versions to be represented by their content fingerprint, which is easier to manipulate in a data structure
- BitTorrent[bittorrent] and IPFS[ipfs] which allow files to be shared in peer-to-peer mode and uses fingerprints to enable a file to be retrieved without having to initially transmit its entire

contents
- recent public blockchains (notably bitcoin[bitcoin]) also use fingerprints to replace the actual content of transactions in Merkle trees
- archival systems use hashes to control the integrity of stored files
- blockchain anchoring systems use hashes to register the existence of files.

These widespread uses are all applications of hash functions that can have far-reaching implications. There is thus a need for improving the guarantee that a hash cannot be brute forced over multiple years, since hashes can be used to protect the integrity of documents expecting lifetime long durability. This is so also when considering the permanent race for computation performance.

Evaluation of security properties

A cryptographic hash function is evaluated on the following properties:

- pre-image resistance: starting from a given fingerprint, it is difficult to find a corresponding input
- second-preimage resistance: starting from a fixed input, it is difficult to find another input with the same fingerprint
- collision resistance: it is difficult to find two arbitrary inputs with the same footprint

Attacks on hash functions aim to compromise the properties described above. Several approaches are currently being considered to compromise hash functions. Among the most commonly put forward hypotheses

- the discovery of a flaw in the algorithm, enabling more direct control of the output bits with certain input manipulations
- weaknesses in the algorithm's construction, enabling it to reduce the impact of part of the input to make the output easier to attack
- increased computing capacity, to reduce the time needed to obtain a brute-force collision

In practical terms, it is impossible to rule out the possibility of the first two points above becoming a reality. Indeed, this kind of flaw has been found in older hash functions (case of MD5 and various other hashatk[hashatk] attacks). It should be noted, however, that the SHA-2 family of functions, still in use today, have not yet had any vulnerabilities discovered. Despite this, the SHA-3 family of functions has already been chosen in preparation for such an evolution. The risk of a brute-force attack deliberately obtaining a defined output by specifying an input value is a matter of probability; however, the space of possible values (in the case of SHA2-256, `2^256` possibilities) considerably reduces this risk.

Finally, cryptographic hash functions (like symmetric encryption functions) are currently considered to be resistant to future quantum computers[postquantic].

How to improve Hash functions?

Commonly used cryptographic hash functions have been subject to cryptanalyses to evaluate their various properties, in order to limit the risk of algorithmic weaknesses and increase the computational complexity for brute-force attacks. Some organizations, notably NIST[nist] organize international[nisthash] competitions aimed at exposing several candidate algorithms to a community of cryptanalysts. Currently, the most widely used hash function algorithms are. the SHA-2 family[sha2]. Other algorithms are already used as successors to SHA-2, including SHA-3[sha3] (evolved from Keccak), specially in the cryptocurrency space.

In order to provide more security than using the best in class algorithm at time t, there exist ways for reusing existing algorithm families in combination. Such combinations called multihashes mainly fall in two broad categories :

- parallel multihashes compute several hashes using different algorithms
- cascading multihashes chain the computation of several hashing algorithms

In practice, it is difficult to find cryptanalysis documentation on the subject, although some research work has looked into the question of hash attacks [hashcoll] on the usual approach of calculating multiple fingerprints in parallel.

a parallel approach

Parallel fingerprinting consists in calculating and storing the fingerprints of a content obtained by several different algorithms. For example, calculating the fingerprints obtained by SHA2-256, SHA3-256 and RIPEMD-160. The "final" fingerprint is simply the concatenation of the fingerprints obtained.

For a message `d` and a final fingerprint `H` :

```
h1=SHA2_256(d)
h2=SHA3_256(d)
h3=RIPEMD_160(d)
H=h1|h2|h3
```

This construction seems to make it impossible to search for a preimage or a second preimage if only of one of the algorithms involved is compromised. It would also seem to complicate the search for a second preimage, even in the event of total compromise. However, some work[hashcoll] seems to show that this intuitive approach is incorrect. It's not clear that this combination significantly increases the strength of the set. It should be noted, however, that this construction doesn't weaken the result either.

On the other hand, it does require all the fingerprints obtained to be retained, which increases the volume of information to be stored or manipulated. A variant allows these different fingerprints to be combined into a single one, by applying a final hash function to the concatenation of the intermediate results. To our knowledge, this variant has not been formally studied in terms of its impact on overall security.

a cascading approach

Cascading is the sequential application of different hash functions to the result of the previous step. The result is a hash that involves several hash functions, but whose size is constrained by the output of the final function. This is used in the Bitcoin codebase for instance, although potentially in the purpose of hiding or shortening a hash by hashing it again with a different algorithm (a cascade of sha256 and ripemd160 is used for instance)

For example, the `H` hash for a `d` message involving three hash functions:

```
H=SHA2_256(SHA3_256(RIPEMD_160(d)))
```

In the absence of a full cryptanalysis, however, there are two problems with this sequence.

Firstly, the hash functions after the first one no longer have the profile `{0,1}^n → {0,1}^m` (where `n` is the size of the input and `m` the size of the output) but an input whose space is limited by the output space of the previous function. Depending on the quality of the statistical distribution of each hash function involved, the final output can be significantly modified.

In addition, all hash function stages following the first level can be considered as a single deterministic black box. A weakness in the first stage of the cascade would therefore be enough to compromise the security properties of the whole.

Kxmash : A novel parallel cascading multi-hash algorithm

KeeeX has developed a method for combining hash functions, making it possible to exploit the individual properties of algorithms while avoiding the known weaknesses of previous methods. The core version of the method can be described as follows :

Kxmash expects as a parameter a string representing the sequence of hash functions to be used. Each hash function is used in sequence with the concatenation of the sequence string and the entire input data. The fingerprint obtained at the output of each calculation stage is added to the input of all subsequent stages. The final fingerprint is the output of the last calculation stage.

For example, using SHA2_512, RIPEMD_160 and SHA3_256 in this order. This sequence is represented as a string `s` : here for instance "SHA3_256<RIPEMD_160<SHA2_512". The order is reversed because :

- it represents a 'recursive' application (despite the fact that the computation can be largely parallelized)
- the name of the first algorithm defines the size of the output.

Then, given a sequence string `s` , an input `d` and a final output `H` (where the '|' operator denotes concatenation), the intermediate fingerprints to be calculated are :

```
h0=SHA2_512 (s | d)
h1=RIPEMD_160 (s | d | h0)
H=SHA3_256 (s | d | h0 | h1)
```

Called "parallel cascading" this approach has the following advantages over previously described combinations :

- a parallel computation is performed but the size of the fingerprint is constrained to the last stage of the cascade of algorithms used
- a cascading computation is performed but the input to each stage of the cascade involves the entire input data, which does not reduce the statistical distribution of their results
- each stage is dependent on the results of the preceding stages, making it impossible to manipulate a single stage
- it is still possible to calculate almost every stage of the cascade in parallel; only the end of the input is affected by the previous stages

To leverage these advantages, it is important to always have at least two calculation stages. For instance the sequence "sha3-256<sha2-256" is used as a default in KeeeX V2.

Comparison with previous methods

It is not the aim of this document to provide a complete cryptanalysis of the kxmash algorithm. Such an analysis remains necessary, but will in any case have to be specific to the algorithms used, or at least limited to certain classes of hash functions.

We can, however, highlight a number of areas in which this combination method should reinforce the security properties of the whole by reducing the impact of a flaw on the algorithms used.

comparing with parallel multihash

As with simple parallel combination, each stage of the cascade involves the initial data in its input. In the simple parallel approach, it was "sufficient" to obtain a second pre-image for each algorithm in order to be able to substitute one datum for another (this task being already considered very difficult). In the case of kxmash, it is necessary to do the same thing, but without having total control over the inputs to each stage, which are also dependent on the output of the earlier stages. Empirically, this aspect of the construction multiplies the difficulty of the task by adding constraints on the handling of the input of each hash function.

The same can be said of the use of a final hash function to "merge" the different results of the simple parallel combination into a single fingerprint. Although an attack might ignore this final stage, the final multihash stage involves by construction all the previous stages and does not allow an attacker to manipulate its input freely.

comparing with cascade multihash

The problem with the cascade multihash is that it propagates the statistical weaknesses of each stage to the next. At the same time, the stages following the first don't bring any significant improvement in the security properties of the hash functions, since a preimage found on the first stage will obtain the same output at the end of the cascade.

Kxmash instead retains the idea of the propagation of each stage to the next(s) but, by involving the entire input data at each stage, it no longer allows to consider the stages following the first as a simple deterministic black box.

In this way, compromising the first stage in order to obtain a second pre-image must also work with all subsequent stages to obtain a second preimage on the output.

What's more, even a compromise (in terms of security properties) of all the algorithms involved would require a favorable combination of all stage outputs, whose input cannot be fully manipulated. A brute-force enumeration or the application of heuristics meeting all these criteria at the same time seems hardly realistic at the the date of writing.

performance considerations

By construction, the algorithms used to calculate each stage of kxmash multi-hashing are kept as they are. This effectively implies multiplying the fingerprint calculation time by the number of algorithms used (we can reasonably ignore the additional time required to concatenate the previous fingerprints, whose size is generally negligible compared to the useful data).

However, this impact on performance is moderated by several factors.

First of all, for most of the computation, it is possible to run the several stages in parallel of each other over the same buffer.

Secondly, the computation speed of the hash functions is usually much more constrained by read performance than by computation time. This read time is shared between the different hash functions used.

These considerations mean that, in practice, the measurable impact of kxmash stays close to parallel multihash.

Conclusion

We believe that kxmash is both robust in terms of resistance to attacks on hash functions, and at the same time futureproof. Robust because a flaw discovered in one algorithm does not allow an attack on the combination, and futureproof as the multi-hash construction itself imposes no constraints on the number or nature of the underlying algorithms.

Kxmash enables the strengths of the algorithms used to be cumulated without weakening the expected security properties, and without significantly impacting performance in terms of the the longevity of the fingerprints produced. It is used by KeeeX to provide proofs of authenticity and integrity over the long term to obey for lifelong archival and proof requirements.

References

- hash : https://fr.wikipedia.org/wiki/Fonction_de_hachage
- cryptohash : https://fr.wikipedia.org/wiki/Fonction_de_hachage_cryptographique
- nist : https://fr.wikipedia.org/wiki/National_Institute_of_Standards_and_Technology
- nisthash : https://fr.wikipedia.org/wiki/NIST_hash_function_competition
- sha2 : https://fr.wikipedia.org/wiki/SHA-2
- sha3 : https://fr.wikipedia.org/wiki/SHA-3
- git : https://fr.wikipedia.org/wiki/Git
- bittorrent : https://fr.wikipedia.org/wiki/BitTorrent
- ipfs : https://fr.wikipedia.org/wiki/InterPlanetary_File_System
- bitcoin : https://fr.wikipedia.org/wiki/Bitcoin
- postquantic : https://fr.wikipedia.org/wiki/Cryptographie_post-quantique
- hashcoll: https://link.springer.com/chapter/10.1007/978-3-540-28628-8_19
- hashatk: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.3746&rep=rep1&type=pdf

Authors Laurent Henocque and Gabriel Risterucci, KeeeX.

This file should either be the keeexed html original with idx xerif-culym-muvon-tyhof-sefih-pyseg-matik-lyhyh-hukyr-teful-gimov-zybeh-godeg-dafik-sopop-segoh-zexux or a keeexed exported pdf from the original

It is signed by 1NkZmqDcTKmAWJaJM957HJo84CFHe5JXZn

Il should be verified on https://s.keeex.me/verify