

ITERATION SCHEMA OF PARAMETERIZED HASH ALGORITHM (PHAL)

Przemysław Rodwald, Piotr Mroczkowski
Military Communication Institute
Cryptology Department
05-130 Żegrze, Poland
{p.rodwald, p.mroczkowski}@wil.waw.pl

ABSTRACT

PHAL (Parameterized Hash ALgorithm) is a proposal of a new dedicated hash algorithm designed in answer to weaknesses of MD/SHA family. Recently proposed attacks on well-known and widely used hash functions motivate a design of new hash functions. Some number of its components used by hash functions can be parameterized. By changing parameters the algorithm dynamically changes the way a hash value is computed. This approach makes the hash algorithm more secure against collisions. PHAL consists of two mechanisms: dedicated compression block and new iteration schema. In the paper the iteration schema is discussed.

INTRODUCTION

Last years brought a great progress in cryptanalysis of hash function. Especially the results of Wang et al. have changed the view of security of well-known iterated hash functions. Most hashes of MD/SHA family have succumbed to the Chinese attacks [9]-[13]. Additionally several flaws in MD-type construction were identified.

Taking into account our experience in cryptanalysis [7] and known cryptographic attacks on hash functions, we decided to propose new hash algorithm.

The rest of the paper is organized as follows. In part 1 I proposed important definitions (hash function, required properties). MD iteration schema and weaknesses of this approach was described in part 2. In part 3 I presented other iteration

schema: HAIFA and dithering. Finally, in section 4 I introduced iteration schema of PHAL – Parameterized Hash Algorithm, and I analyzed security of this schema.

1. HASH FUNCTION

Hash function h is computationally efficient function mapping an input m of arbitrary finite bitlength, to an output of fixed bitlength n .

$$h: \{0,1\}^* \rightarrow \{0,1\}^n,$$

$$\text{where } \{0,1\}^* = \bigcup_{i \in \mathbb{N}} \{0,1\}^i$$

For cryptographic hash function, the following properties are required:

preimage resistance (non-invertibility) - it is computationally infeasible to find any input which hashes to that output,

second-preimage resistance (weak collision resistance) - it is computationally infeasible to find any second input which has the same output as any specified input,

collision resistance (strong collision resistance) - it is computationally infeasible to find any two distinct inputs m, m' which hash to the same output.

For an ideal hash function finding preimage or second-preimage require about 2^n operations, and finding collision in birthday attack needs approximately $2^{n/2}$ operations.

2. MD-TYPE CONSTRUCTION

Typical hash function has usually two

components: a compression function and a method of iteration. In 1989 Merkle [5] and Damgård [2] independently proposed a generic construction of a cryptographic hash function. Most of popular hash functions follow this schema. Schema based on the fact that hash function built with the MD construction is as resistant to collisions as its compression function is. For years it was widely believed that MD construction maintains the preimage resistance and the second preimage resistance as well. Unfortunately these beliefs were questioned. Recent attacks showed several undesirable properties and vulnerability to some attacks: multi-block collision attacks, fixed-point attack, multi-collision attack, or time-memory trade-offs.

In the typical MD-type iteration schema, hash function $h:\{0,1\}^* \rightarrow \{0,1\}^n$ is designed from the compression function

$$\varphi:\{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n,$$

where n denotes the size of the chaining value, usually equal to hash value length, and b denotes the block size of the compression function. The original message m is padded with its length (after additional padding to make the message a multiple of the block size b after the final padding), and next the message is divided into k blocks of b -bits each $m = m_1 m_2 m_3 \dots m_k$. An initial chaining value (IV) is set for the hash function. The process of hashing looks as follows:

$$H_0 = IV$$

$$H_i = \varphi(H_{i-1}, m_{i-1}) \quad i = 1, 2, \dots, k$$

$$h(m) = H_k$$

As was said before, recent years gave us few interesting attack on this construction.

It was mentioned in 1999 [6] that, if fix-points of the compression function is easy to find, then the second pre-image attacks on MD hash functions needs only $O(m \cdot 2^{m/2})$ time and $O(m \cdot 2^{m/2})$ memory.

Similar conclusion was presented by Kelsey and Schneier [4] in 2005. They proposed the same idea, but without assumption that fix-points can be easily found. This improvement was done by using Joux's [3] ideas for efficiently finding multi-collisions in an

iterative hash functions.

3. OTHERS ITERATION SCHEMAS

In 2006 Biham i Dunkleman [1] proposed new iterative schema - *HASH Iterative Framework (HAIFA)*. Hash function $h:\{0,1\}^* \rightarrow \{0,1\}^n$ is designed using the compression function

$$\varphi:\{0,1\}^n \times \{0,1\}^b \times \{0,1\}^c \times \{0,1\}^s \rightarrow \{0,1\}^n$$

where n is the size of the chaining value, usually equal to hash value length, b is the block size for the compression function, c is the size of *counter* - number of bits hashed so far mod 2^{512} , s is the size of the random value (*salt*). The original message m is padded and divided as previously. The process of hashing looks as follows:

$$H_0 = IV$$

$$H_i = \varphi(H_{i-1}, m_{i-1}, \#bits, salt) \quad i = 1, 2, \dots, k$$

$$h(m) = H_k$$

The number of bits hashed so far (*counter*) was added to increase resistance of hash function to fix-points attacks. The attacker is forced to work harder in order to find fix-points. Random value (*salt*) increases resistance of hash function to generic attacks, where we have table generated in advance (*message - hash value*). When we add salt, this kind of tables should have been generated for every possible value of salt. And this is computationally infeasible.

Other interesting construction, called *dithering*, was presented by Rivest [8] in 2005.

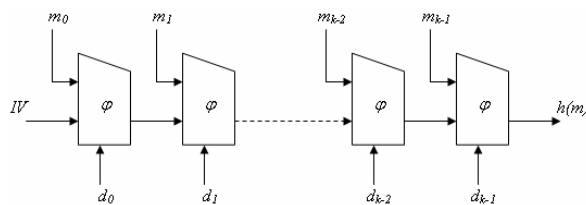


Fig.1. Dithering technique.

The word “dithering” derives from image-processing, where a variety of gray or colored values can be represented by mixing together pixels of a small number of basic shades or colors; this is done in a random or pseudo-

random manner to prevent simple visual patterns from being visible. Authors adapted the term dithering to refer to the process of adding an additional *dithering* input to a sequence of processing steps, to prevent an adversary from causing and exploiting simple repetitive patterns in the input.

Some possible ways of employing dithering:

- dithering with a counter - using the index i as the dither value: $d_i = i$;
- dithering with a pseudorandom sequence - utilizing some pseudo-random sequence r_0, r_1, \dots as the dither value: $d_i = r_i$;
- dithering with alternating 0's and 1's – using a sequence of alternating 0's and 1's:

$$d_i = \begin{cases} 0 & \text{if } i \text{ is even} \\ 1 & \text{otherwise.} \end{cases}$$

The main idea proposed by Rivest is to use a square-free sequence as a dither. A sequence is square-free if it doesn't contain two equal adjacent subwords.

4. PHAL ITERATION SCHEMA

In the proposed schema, hash function $h: \{0,1\}^* \rightarrow \{0,1\}^n$ is designed using the compression function

$$\varphi: \{0,1\}^n \times \{0,1\}^b \times \{0,1\}^c \times \{0,1\}^s \times \{0,1\}^r \rightarrow \{0,1\}^n$$

where n is the size of the chaining value, equal to hash value length, b is the block size for the compression function, c is the size of *counter* - number of bits hashed so far mod 2^{512} , s is the size of the random value (*salt*), r is the size of number of (*rounds*). The original message m is padded and divided as previously. The process of hashing looks as follows:

$$H_0 = IV$$

$$H_i = \varphi(H_{i-1}, m_{i-1}, counter, salt, rounds)$$

$$i = 1, 2, \dots, k$$

$$h(m) = H_k$$

The number of bits hashed so far (*counter*) was added to increase resistance of hash function to fix-points attacks. In this schema finding a fix points is more difficult, because every middle chaining value H_i depends on dynamically changeable value of number of bits hashed so far.

Random value (*salt*) increases resistance of hash function to generic attacks, where we have table generated in advance (*message – hash value*). When we add salt, this kind of tables should have been generated for every possible value of salt. This approach is computationally infeasible.

Number of rounds (*rounds*) was added to make this function more flexible. There is a trade-off between performance and security. Small number of rounds should be used in systems where performance is most important. When we need higher security we can increase number of rounds. When we look at the history of cryptanalysis, we can observe that very often hash function are broken gradually. This means that, for example, at the beginning someone finds some weaknesses or collision for reduced variant of iterated hash function, let's say for m rounds from n rounds, where $m < n$. Some time later collisions are found for k rounds, where $m < k < n$. So cryptographers need time to find collision in full version of particular hash function. When we add to the function number of rounds as a parameter, we can take active part in the race between hash function designers and those who want to break them. When some weaknesses of reduced version of hash function will be found, then we could recommend new, increased value of number of rounds.

PHAL algorithm should support different size of the hash value. As minimum it supports 224, 256, 384, and 512-bit hash values (NIST's draft minimum requirements). When we ensure this, then a hash function can be used in a number of different applications. Additionally this makes implementation more flexible for wide range usage.

5. CONCLUSION

In this article an iterative schema of PHAL – new Parameterized Hash Algorithm was presented. This new approach is similar in some sense to Biham's idea of HAIFA, but taking into account number of rounds as a parameter. This flexible approach give the designer of a cryptosystem possibility of making some trade-off. Trade-off between security and computational complexity.

BIBLIOGRAPHY

- [1] E. Biham, O. Dunkelman, *A Framework for Iterative Hash Functions - HAIFA*, The Second Cryptographic Hash Workshop, Santa Barbara, USA, 2006
- [2] I. Damgård, *A design principle for hash functions*, Advances in Cryptology, Crypto'89, LNCS 435, Springer-Verlag, 1989
- [3] A. Joux, *Multicollisions in Iterated Hash Functions*, Advances in Cryptology, Crypto'04, LNCS 3152, Springer-Verlag, 2004
- [4] J. Kelsey, B. Schneier, *Second Preimages on n -Bit Hash Functions for Much Less than $2n$* , Advances in Cryptology, Eurocrypt'05, LNCS 3494, Springer-Verlag, 2005
- [5] R. Merkle, *One way hash function and DES*, Advances in Cryptology, Crypto'89, LNCS 435, Springer-Verlag, 1989
- [6] R. D. Dean, *Formal Aspects of Mobile Code Security*, Ph.D. dissertation, Princeton University, 1999
- [7] P. Rodwald, *Cryptanalysis of Petra's family hash functions*, Military Communications and Information Systems Conference, Gdynia, 2006
- [8] R. Rivest, *Abelian square-free dithering for iterated hash functions*, The First Cryptographic Hash Workshop, Gaithersburg, USA, 2005
- [9] X.Wang, D.Feng, X.Lai, H.Yu, *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*, Cryptology ePrint Archive, <http://eprint.iacr.org/2004/199>, 2004
- [10] X.Wang, X.J.Lai, D.Feng, H.Chen, X.Yu, *Cryptanalysis of the Hash Function MD4 and RIPEMD*, Advances in Cryptology, Eurocrypt'05, LNCS 3494, Springer-Verlag, 2005
- [11] X.Wang, H.Yu, Y.Yin, *Efficient Collision Search Attacks on SHA-0*, Advances in Cryptology, Crypto'05, LNCS 3621, Springer-Verlag, 2005
- [12] X.Wang, A.L.Yin, H.Yu, *Finding Collisions in the Full SHA-1*, Advances in Cryptology, Crypto'05, LNCS 3621, Springer-Verlag, 2005
- [13] X.Wang, A.Yao, F.Yao, *New Collision search for SHA-1*, Rump Session Crypto'05, 2005

BIOGRAPHY

Przemysław Rodwald was born in Lębork, Poland, in 1977. He was graduated from the Cybernetics Faculty of the Military University of Technology in Warsaw in 2001. Since 2002 he has been working at the Military Communication Institute. He is currently working in the NATO Internet Protocol Security Task Force group. His main objects of interest: hash functions and cryptanalysis.

Piotr Mroczkowski was born in Ilża, Poland, in 1975. He was graduated from the Cybernetics Faculty of the Military University of Technology in Warsaw in 2000. Since 2000 he has been working at the Military Communication Institute. He is interested in cryptology and computer science.