Proceedings of the Federated Conference on Computer Science and Information Systems pp. 387–390
DOI: 10.15439/2018F254
ISSN 2300-5963 ACSIS, Vol. 15

Parametric Hash Function Resistant to Attack by Quantum Computer

Sergey Krendelev
Novosibirsk State University,
JetBrains research, Novosibirsk,
Russia
Email: s.f.krendelev@gmail.com

Polina Sazonova
Novosibirsk State University,
JetBrains research, Novosibirsk,
Russia
Email: psazonova@gmail.com

Abstract—**This paper describes an algorithm for creating hash function, resistant for quantum computer. The given approach is based on the problem of solving a system of polynomial equations in integers, where the number of equations is less than the number of unknown parameters. The developed algorithm is parameterized so the result of the hash function depends on several parameters, therefore, it will take considerably longer to select the solution of the task. The avalanche effect is about 50%, collision is impossible because the task to find a solution of the described system of equations with a degree greater than 3 is algorithmically unsolvable. This hash function was developed for blockchain to ensure its integrity, but it can also be used in any application where a hash function is needed.**

I. Introduction

SINCE Peter Shor has been demonstrated the solvability of the problem of discrete logarithm factorization using quantum computer in 1995 [1], there was become actually a post-quantum cryptography. It was necessary to develop such algorithms that could not be solved with the help of quantum computers.

Blockchain technology become popular for different kinds of applications: in banking, gambling, registries and etc. It uses hash function – cryptographically primitive for supporting invariability and consistency of data.

Hashing in blockchain is the process of converting an array of input data of arbitrary length into an output bit string. Hash function uses for making a digest of blocks or some another data, stored not only in blockchain. Hash functions guarantee the "irreversibility" of data.

But inventing quantum computers will force to develop a hash function resistant to the quantum computers.

In developing the hash function algorithm for the blockchain technology, some requirements is important: hash function should be resistance to collisions of first and second kind and it should have a high avalanche effect.

A. Motivation

At present, post-quantum cryptography is based on four approaches that guarantee resistance to quantum computers These are Code-based cryptography, Hash-based Digital Signature Schemes, Multivariate Public Key Cryptography, Lattice-based Cryptography [2].

Our algorithm is based on problem where the number of equations is less than the number of unknown parameters.

B. Algorithm Idea

As already mentioned, post-quantum cryptography is based on algorithmically unsolvable problems. We describe two complexity problems (we call it A and B) that are suitable for us. Our approach is constructed on Problem B, Problem A is its particular case. The work of Aitai [3] is equivalent to Problem A. In this section, we will show the transition from problem A to problem B and justify using of these computational problems.

Problem A. It is needed to find the solution of a system of linear Diophantine equations in integers.

Strongly underdefinished system of equations or a system where the number of equations is substantially less than the number of unknowns is given:

$$\sum_{j=1}^{n} a_{ij}x_j = d_i \; a_{ij}, d_i \in \mathrm{Z},$$

$$i = 1,2,\dots,m, j = 1,2,\dots,n, n > m$$

If there are restrictions, such as $x_j \geq 0,\ j = 1,2,\dots,n$ or $x_j \in \{0,1\}, j = 1,2,\dots,n$ - this task becomes the task of integer programming. Particularly interesting for encryption is the case where $n > m$ (it is a strongly underdefinished system of linear equations). In particular, if $m = 1$ and $x_j \in \{0,1\}, j = 1,2,\dots,n$, then this task is the task of the knapsack problem or subset-sum problem.

The scheme of the hash function, described by M. Aitai in 1996, is a special case of problem A. In the original article it tells about the lattice theory, but we show that the problem on lattices is equivalent to the described problem A.

Let us describe the scheme of the hash function of M. Aitai.

A randomly selected matrix $\mathrm{A} \in \mathrm{Z}_p^{n\times m}$ of dimension $n \times m$ is chosen, where $n < m$. Vector $\mathrm{x} \in \mathrm{Z}_p^m (d < p)$ will be hashed.

For this the system $\mathrm{A}x = mod(p) \in \mathrm{Z}_p^n$ is calculated, where $\mathrm{A}x$ is the hash of the vector x.

Note, that the parameters are set: n, m, q, $d > 1$, $n < m$, $q > d$, $\mathrm{A} \in \mathrm{Z}_p^{n\times m}$.

We note that the solution of equation $\mathrm{A}x = mod(p) \in \mathrm{Z}_p^n$ is a problem A, which is guarantees a solution. Consequently, the solution of the system of linear equation where the number of equations is less than the number of unknowns is equivalent to the problem on lattices.

This work was not supported by any organization

Problem B. It is necessary to find a solution of a system of polynomial equations in integers.

$$f_i(x_1, x_2, \dots, x_n) = 0, i = 1,2, \dots, m$$

Problem B is algorithmically unsolvable. In addition, if the degrees of polynomials ≥ 3 and $n > m$, then the problem is algorithmically unsolvable in integers. This conclusion follows from solution of 10th Hilbert problem.

In this paper, we consider a variant of constructing a hash function based on the problem B. In this type of hash function, a set of parameters can be used to enhances the persistence of the hash function. If you build a set of hash functions that depends on a large number of parameters, you get an object of the Universal hash type [4].

II. Algorithm Description

As our algorithm is parametric, first we need to choose parameters. In based version the parameters is: module *p*, size of dimension $m \times n$, set of starting coefficients $\propto_1, \propto_2, \dots, \propto_n$, size of block *b*, rules of forming summands $h_1(x), h_2(x), \dots, h_m(x)$.

Let us consider algorithms parameters. We also have developed requirements for parameters for the better result.

All calculations will be performed on the module *p*. The module should be a sufficiently large prime number.

We will generate some set of vectors according to special rules derived from the parameters $\propto_1, \propto_2, \dots, \propto_n$, $\propto_i \in Z_p^n, i = 1,2, \dots, n$, where n - is an arbitrary integer. The dimension of these vectors is *n*.

Suppose that some hashed document is described by a set of numbers $x = (x_1, x_2, \dots)$. Each number is a certain number of bits, assembled into a conditional block. Our block can be 8, 10, 12, etc. bit. The size of the block in bits *b* is another parameter of our algorithm.

A rule of generation of functions $h_1(x), h_2(x), \dots, h_m(x)$ should be defined as a parameter. It will determine the order of formation of the terms of our hash function.

The computation procedures of the proposed algorithm are illustrated as following.

Step 1: data preparation.

On this step, we prepare a string of decimal integers $x = (x_1, x_2, \dots, x_m)$ according to the input file.

Next, we prepare a matrix A = $(a_1, a_2, \dots, a_m)$, forming on the set of starting coefficients $\propto_1, \propto_2, \dots, \propto_n$. Vector a_i construct as a recurrent sequence according to the formula $a_i = \propto_1 a_1 + \propto_2 a_2 + \dots + \propto_n a_n$

Step 2: constructing a hash function.

Then the following vector will be a hash:

$$H(x) = [a_1 h_1(x) + a_2 h_2(x) + \dots + a_m h_m(x)] mod(p)$$

Functions $h_1(x), h_2(x), \dots, h_m(x)$ in hash function can be implemented as follows, but we can choose any rule for forming $h_i(x)$:

$$H(x) = [a_1 x_1 x_2 + a_2 x_2 x_3 + \dots + a_m x_m x_1] mod(p)$$

The size of the output string of the hash function is $n \times m$.

Step 3: modifications.

On the large file we have a high probability when some terms will be a zero. The main cause of it is a rule of forming a recurrent sequence, when zero in some terms is cumulated. To avoid it in a base version of algorithm we use a cyclic shift. In another version, we can use replacing on zero-component to fixed number which can be a parameter too.

Thus, we have constructed a hash-scheme with parameters, where the parameters are: module *p*, vector dimension *n*, block size *b*, terms generation rules $h_1(x), h_2(x), \dots, h_m(x)$.

III. The Toy Example

On the first step parameters is chosen. It is a simple number p, which will be a module; for example here p = 4049, the dimension of the vector n = 4, m = 4, the size of the block is b = 6 bits, the rule of generating multipliers in the term is $x_i x_{i+1}$, window size is 2, coefficients $\propto_1, \propto_2, \propto_3, \propto_4$ = (3174,3507,860,1294).

On the first step, we preparing a data.

Data from the file represented as decimal integers is x_1, x_2, ..., where each *x* is 6 bits. We separate 32 bits file on block of 6 bit and convert to decimal integers and the result is (34, 16, 23, 63).

Next, we need to generate coefficients A from starting coefficients $\propto_1, \propto_2, \dots, \propto_m =$ (3507,860,1294,3174)

$$\begin{pmatrix} \propto_1 \\ \propto_2 \\ \propto_3 \\ \propto_4 \end{pmatrix}, \propto_i \in \mathrm{Z}_{4049}, i = 1,2,3,4$$

Each a_k is calculate using recurrent sequence. On first step it will be $a_k = 3507a_{k-1} + 860a_{k-2} + 1294a_{k-3} + 3174a_{k-4}$.

Let a_i is:

$$a_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}; \; a_{-1} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}; \; a_{-2} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}; \; a_{-3} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Consequently:

$$a_1 = \begin{pmatrix} 3507 \\ 860 \\ 1294 \\ 3174 \end{pmatrix}$$

When we read the elements of a file by 2 items and calculate the product, it can turned to 0, if any one term turns to 0. Therefore, it is necessary to provide a decision of this problem in this case. We make a cyclic shift of numbers in vector a_i on 1 positions.

Next step we need to construct and calculate hash function:

$H(x) = [a_1 x_1 x_2 + a_2 x_2 x_3 + a_3 x_3 x_4 + a_4 x_4 x_1] mod(p)$

Let us construct the first term for the hash $a_1 x_1 x_2$:

$$\begin{pmatrix} 3507 \\ 860 \\ 1294 \\ 3174 \end{pmatrix} \times 34 \times 16 = \begin{pmatrix} 729 \\ 2205 \\ 3459 \\ 1782 \end{pmatrix} mod(4049)$$

Now the next vector according to the recurrence sequence should be calculated:

$$a_2 = 3174\begin{pmatrix}729\\2205\\3459\\1782\end{pmatrix} + 3507\begin{pmatrix}1\\0\\0\\0\end{pmatrix} + 860\begin{pmatrix}0\\1\\0\\0\end{pmatrix} + 1294\begin{pmatrix}0\\0\\1\\0\end{pmatrix} = \begin{pmatrix}1325\\2858\\3321\\3664\end{pmatrix} mod(4049)$$

We calculate the product from the document data to the generated vectors:

$$\begin{pmatrix}1325\\2858\\3321\\3664\end{pmatrix} \times 16 \times 23 = \begin{pmatrix}1720\\3053\\3379\\35\end{pmatrix} mod(4049)$$

Modify the first two terms of hash:

$$\begin{pmatrix}729\\2205\\3459\\1782\end{pmatrix} + \begin{pmatrix}1720\\3053\\3379\\35\end{pmatrix} = \begin{pmatrix}2449\\1210\\2769\\1817\end{pmatrix} mod(4049)$$

And so on by induction. The final result of hash function will be (1679, 1137, 1883, 213).

IV. Possible modifications

The described algorithm can be modified as follows.

Modification 1. The nonlinear case of the formation of terms for the hash function should be considered.

We need to define a rule where the data from the input file will be combined and be distributed according to the hash function.

For example,

$$Ax = [a_1x_1x_2x_3 + a_2x_2x_3x_4 + \cdots + a_mx_mx_1x_2]mod(p)$$

The operations of multiplication are made on modulo p. Thus, we can consider other types of polynomials for the nonlinear case.

Modification 2. The nonlinear case should be considered when multiplications are made according to some multiplication table.

This table can be generated according to the hashed data.

V. Security Proof

C. Theoretical Foundation

For a hash function f will be cryptographically stable, it must satisfy the follow three basic requirements which most hash functions are based in cryptography:

1. Irreversibility or resistance to restoration of the prototype: for a given value of a hash function *y*, a data block *x* for which $f(x) = y$ must not be computed.
2. Resistance to collisions of the first kind or restoration of the second inverse images: for a given message *x* it must be computationally impossible to find another message *z* for which $f(x) = f(z)$.
3. Resistance to collisions of the second kind: it must be computationally impossible to select a pair of messages *x, z* having the same hash.

These requirements are not independent:

1. An invertible function is unstable to collisions of the first and second kind.
2. A function that is unstable to collisions of the first kind is not resistant to collisions of the second kind; the converse is not true.

Let us consider how the collision for our variant of the hash function will look. Let the same hash function be given for two different *x* and *z* documents:

$$H(x) = [a_1h_1(x) + a_2h_2(x) + \cdots + a_mh_m(x)]mod(p)$$

$$H(z) = [a_1h_1(z) + a_2h_2(z) + \cdots + a_mh_m(z)]mod(p)$$

Collision means that if $x \neq z$, but $H(x) = H(z)$.

Suppose for our algorithm the source document is known. Then, taking into account that the vectors $a_1, a_2, \dots, a_m$ are formed according to the parameters and the special rules to the function $h_i(x), i = 1,2, \dots, m$ calculations, the attacker is aware of the following information: vectors $a_1, a_2, \dots, a_m$, $\alpha_i = h_i(x), i = 1,2, \dots, m$ and $v = \sum_{j=1}^{m} \alpha_i a_j \, mod(p)$. The vector *v* is a hash of the document.

We need to solve equation $H(x) = d$ to find collisions, but this problem is equivalent to problem B, described in the section I.B of this article.

Thus, we demonstrated that a collision is theoretically possible. However, we affirm that there is no sense to find a collision for our algorithms, since the problem is algorithmically unsolvable if there are polynomials in the system of equations with a degree greater than 3.

For cryptographic hash functions it is also important that with the slightest change in the argument, the value of the function changes greatly (avalanche effect). In particular, the value of a hash should not give a leak of information, even about individual bits of the argument. This requirement is the key to the crypto-stability of algorithms for hashing user passwords to obtain keys.

D. Implementation Details

Describing algorithm was implemented in Python 3.3 for testing; measurements were made on a computer with an Intel Core i5-4210U of 2 cores, operating at 2.4Ghz. The PC contains 8 Gb RAM.

For testing avalanche effect, we calculated the hash function from the source file, changed an arbitrary bit in the source file and calculated the hash function from the modified file. Then a bitwise comparison was made. In the case of any documents of any size, when changing 1 bit in the source file, the hashes of the primary and modified files coincide only by 47-50% with a bitwise comparison.

Moreover, the best parameters at which the maximum number of discrepancies is reached is a sufficiently large prime number and the large dimension of the vector K is about 100.

The speed of the algorithm is about 0.007 sec for a 1 kb file, an average of 70 seconds for a 500 kb file, an average

500 seconds for a 1 mb file. The algorithm works both with text data, and with photo, video and audio content. Obviously, realization of this algorithm should be optimized to reduce the processing speed of the file.

VI. Conclusion

In this article is proposed an algorithm of hash function resistant to quantum computer. This algorithm uses algorithmically unsolvable problem of finding a solution to a system of polynomial equations in integers. Our algorithm is parametrized, which increases the decision-making time. It is resistant to collisions, because the problem on which the algorithm is built is algorithmically unsolvable (in the case where the degree of the polynomial is greater than 3). The avalanche effect is about 47-50% with a bitwise comparison. The algorithm can work both with text data, with photo, video and audio contents.

This algorithm was developed for blockchain technology to increase its resistance to attacks by quantum computer. It can also be used in any application where a hash function is needed.

VII. References

[1] Shor P.W. “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer”, SIAM J. Com., 26:5, 1997, pp. 1484-1509.

[2] Bernstein D. J., Buchmann J., Dahmen E. “Post-Quantum Cryptography”. Springer-Verlag Berlin Heidelberg, 2009.

[3] M. Ajtai. “Generating Hard Instances of Lattice Problems”. In: 28th ACM Symposium on Theory of Computing, ACM, Philadelphia, USA, 1996, pp. 99–108.

[4] L. Carter and M. Wegman. “Universal Classes of Hash Functions”. In: J. Computer and System Sciences, Vol. 18(2), 1979, pp. 143–154.

[5] O. Goldreich, H. Krawczyk and M. Luby. “On the existence of pseudorandom generators”. In: SIAM J. on Computing, Vol. 22-6, 1993, pp. 1163–1175.

[6] J. Hastad, R. Impagliazzo, L.A. Levin and M. Luby. “A Pseudorandom Generator from any One-way Function”. In: SIAM J. on Computing, Vol. 28 (4), 1999, pp. 1364–1396.

[7] A.K. Lenstra, H.W. Lenstra, L. Lov´asz. “Factoring Polynomials with Rational Coefficients”. In: Mathematische Annalen, vol. 261(4), 1982, pages 515–534.

[8] C.P. Schnorr. “A more efficient algorithm for a lattice basis reduction”. In: Journal of Algorithms, Vol. 9, 1988, pages 47–62.