Multivariate Quadratic Hash

Jintai Ding, Bo-yin Yang***

University of Cincinnati*
Technical University of Darmstadt*
Institute of Information Science, Academia Sinica**

1. General Introduction
2. General Multivariate Hash
3. Special Implementations
4. Conclusion

1 General Introduction

- It is well-circulated joke that hash functions are the only common cryptological primitives that are getting slower as chips evolve.
- One common feature of the currently used hash functions is that it is indeed an area of art, where the design of the system is based on rather complicated procedures and the security of the system is very difficult to study from the theoretical point of view.
- The work of Wang et al on the collision of some well-known and standard hash functions – SHA-1,MD5.
- Even the work of Wang et al is still not well understood and people are still trying to analyze the methods used in some systematical way, (The work of Sugita etc. using Groebner

basis)

- One direction is to look for provably secure hash functions, whose security relies on well-understood hard computational problems.
- One of the ideas is to use RSA moduli. This idea has the problem that *someone* has to come up with the RSA modulus you are using, which no doubt makes many people uneasy.
- A different solution is `LASH` and similar ideas. These hashes tend to be slower, although they have a saving grace in terms of provable security. In these formulations, the designer seeks a reduction of the security of the compression function to some other intractable problem.
- We propose our own version, which is based on the multivariate quadratic (MQ) problem, and study how well it works.

1.1 The origin of the idea: Multivariate Public Key Cryptosystems

- Cryptosystems based on multivariate functions over a finite field instead of single variable functions.

- The cipher –**the public key** is given as:

$$G(x_1, ..., x_n) = (G_1(x_1, ..., x_n), ..., G_m(x_1, ..., x_n)).$$

Here the G_i are multivariate polynomials over a small finite field k . G can be viewed as a map: $G : k^n \longrightarrow k^m$

- **Encryption**

Any plaintext $M = (x'_1, ..., x'_n)$ has the ciphertext:

$$G(M) = G(x'_1, ..., x'_n) = (y'_1, ..., y'_n).$$

Encryption: Evaluation of the values of polynomials.

- **Decryption**

  To decrypt the ciphertext $(y'_1, ..., y'_n)$, we need to know the hidden structure of G– **the secret key**, so that one can invert the map G to find the plaintext $(x'_1, ..., x'_n)$.

  Decryption relies on the hidden structure of the public key

Multivariate Signature schemes

- To verify, check indeed if the signature and the hash value of the plaintext satisfies the equations given by the public key.

  Document $(y'_1, ..., y'_m)$, signature $(x'_1, ..., x'_n)$, public key $G(x_1, .., x_n)$.

  To verify, we need ro check:

$$G(x'_1, ..., x'_n) \stackrel{?}{=} (y'_1, .., y'_m).$$

- To sign, one need to find one solution of the equation above, or to invert the map G.

Direct attack is to solve the set of polynomial equations:

$$G(x_1, ..., x_n) = (y'_1, ..., y'_m)$$

or

$$(G_1(x_1, ..., x_n), ..., G_m(x_1, ..., x_n)) = (y'_1, ..., y'_m),$$

because G and $(y'_1, ..., y'_m)$ are known.

- **Security Foundation**.

  - Solving a set of n randomly chosen quadratic equations (nonlinear) with n variables over a finite field is NP-complete.

- [MQ Problem:] Solve the system $P_1 = P_2 = \cdots = P_m = 0$, where each P_i is a quadratic polynomial in $x_1, \ldots, x_n$ and coefficients and variables are in $GF(q)$.

- We believe that it is a hard problem on average.

- **Quadratic Constructions MPKC.**

  1) Efficiency considerations of key size and computation efficiency lead to mainly quadratic constructions.

$$G_l(x_1, ..x_n) = \sum_{i,j} \alpha_{lij} x_i x_j + \sum_i \beta_{li} x_i + \gamma_l.$$

  2) Mathematical structure consideration: any set of high degree polynomial equations can be reduced to a set of quadratic equations.

$$x^3 = 1,$$

  is equivalent to

$$\begin{aligned} x^2 &= y \quad 0 \\ xy &= 1. \end{aligned}$$

- **The Potentials.**

  I.) We have not yet seen how a quantum computer can be used to attack MPKCs efficiently. (Post-quantum cryptography)

  II.) We have seen the potential to build much more efficient public key cryptosystems. (Passive RFID)

- **MPKCs**

  - Sflash (Matsumoto-Imai-Minus) systems, accepted by NESSIE as a security standard for low cost smart cards.

  -Quartz, HFEv-Minus: NESSIE

  -PMI+, IPHFE

  -Rainbow; TTS, TRMC

- Our general assumption 1 (A1):

  MQ problem is exponential in n when $n = m$.

  If we choose the quadratic coefficients of P_i randomly, then MQ in this case is exponential.

- An even more general assumption 2 (A2):

  MQ problem is exponential in n if $0 < \beta m/n =< \alpha$.

  If we choose the quadratic coefficients of P_i randomly, then MQ in this case is exponential.

2 The general construction

One way function is gives by

$$F(x_1, ..., x_n) = (f_1(x_1, ..., x_n), ..., f_m(x_1, ..., x_n)),$$

where f_i is a randomly chosen quadratic polynomial over $GF(q)$. Here all the coefficients are chosen randomly. We choose the case $m/n = 1/2$ (one can change $1/2$ into possiblely other values), and use this as a compressor in a Merkle-Damgard iterated compression hash function. n is an even number.

Claim 1.

With $m = n/2$, and randomly chosen $F = F(x_1, ..., x_{n/2}, xn{+}1, ..., x_n)$, F is impossible to invert or to find a second image for in polynomial time.

Proof under A1.

We need to solve the equation:

$$F = F(x_1, ..., x_{n/2}, x_{n/2+1}, ..., x_n) = (y'_1, ..., y'_n/2).$$

We can assume that we know already the value of $xn/2 + 1, ..., x_n$, which surely will not make the problem harder.

We need to solve

$$F(x_1, ..., x_{n/2}, x'_{n/2+1}, ..., x'_n) = (y'_1, ..., y'_n/2),$$

where $xn/2 + 1, ..., x_n$ are given constants, but in the evaluation process, the quadratic part of $x_1, ..., x_{n/2}$ stays random. Therefore the equation remains difficult to solve with $A1$.

This shows that it is impossible to invert the hash.

Claim 2. Assume A1, the MQ problem with $2 \geq m/n \geq 1$ is impossible to solve in polynomial time.

Proof

We only need to deal with the case $m = 2n$. Any $m = n$ problem can be changed into a $m = 2n$ problem by guessing $\sqrt{n}$ variables by adding all the quadratic equations derived from these variables' value we guess. If our problem is polynomial, it implies that the $m = n$ case is not exponential, which is impossible.

Claim 3. With $m = n/2$, and randomly chosen $F = F(X, Y)$, where $X = (x_1, ...x_{n/2}), Y = (x_{n/2}, ..., x_n)$, The equation

$$F(X_1, Y_1) = F(X_2, Y_2),$$

is impossible to solve in polynomial time.

Proof with A1

It follows the same argument. Namely we have $n/2$ equations and $2n$ variables, we can here assume that we know the values of the $3n/2$ variables. In this case, it means that at least in one of the polynomial we will have $n/4$ variables unknown, which we can assume to be the right side. We can further assume that the value of the left hand polynomials (or even all the variables) is known. Then this becomes a problem of random polynomials with $n/2$ equations and $n/4$ variables. The proof then follows from Claim 2 above.

This shows the **collision free** property of the hash.

A rough estimate of effort to solve these MQ problems, which is derived from studying MPKCs:

1) 20 equations in 20 $GF(256)$ variables: 2^{80} cycles;

2) 24 equations in 24 $GF(256)$ variables: 2^{91} cycles.

Practical implementations

1) Assuming 160-bit hashes (SHA-1), preliminary runs with a generic function F:

- 40 $GF(256)$ variables into 20: 5400 cycles/byte (6.0 cycles/mult)
- 80 $GF(16)$ variables into 40: 19300 cycles/byte (3.2 cycles/mult)

2) Assuming 256-bit hashes (SHA-2), preliminary runs with a generic function F:

- 32 $GF(2^{16})$ variables into 16: 4800 cycles/byte (19 cycles/mult)
- 64 $GF(256)$ variables into 32: 12500 cycles/byte (6.0 cycles/mult)
- 128 $GF(16)$ variables into 64: 53500 cycles/byte (3.2 cycles/mult)

- The coefficients of the map F is taken from the binary expansion of π.
- Multiplication in $GF(16)$ and $GF(256)$ are implemented with tables. In fact, in $GF(16)$, we implement a 4kBytes table with a where we can multiply simultaneously one field element by two others.
- Multiplication in $GF(2^{16})$ is implemented via Karatsuba multiplication over $GF(256)$.

3 Special constructions

3.1 Sparse polynomials

The idea is the same as above but we choose each of the above polynomial to be a sparse one.

Assumption As n and $m = kn$ goes to infinity, for any fixed $0 < \epsilon < 1$, a random sparse quadratic system with a randomly picked ϵ proportion of the coefficients being non-zero (and still random) will still take time exponential in n to solve.

The key problem is that the ratio ϵ of the nonzero terms.

1. How many we choose such that it is fast?

   Answer: no more than maybe one in ten.

2. How many we choose such that it is secure?

   Answer: a fixed percentage.

3. How do we choose the sparse terms?

   Answer: probably randomly.

To store the formula in a sparse form takes extra space and time to unscramble the storage, so it is never worthwhile to have $\epsilon > 1/6$ or so in practice.

In the examples in the following, about 3% of the coefficients are non-zero (one in 30). To give one example, there is around 30 terms per equation in a 40-variable quadratic form.

Assuming 160-bit hashes (SHA-1), preliminary runs with a generic function F:

- 40 $GF(256)$ variables into 20: 450 cycles/byte (6.4 cycles/mult)
- 80 $GF(16)$ variables into 40: 1570 cycles/byte (3.6 cycles/mult)

Assuming 256-bit hashes (SHA-2), preliminary runs with a generic function F:

- 32 $GF(2^{16}$ variables into 16: 1520 cycles/byte (19 cycles/mult)
- 64 $GF(256)$ variables into 32: 3960 cycles/byte (6.4 cycles/mult)
- 128 $GF(16)$ variables into 64: 11420 cycles/byte (3.6 cycles/mult)

There is no longer any proof of security because fixing half the variables in a random quadratic map is still a random quadratic map, or at least a map whose quadratic part is all random, but fixing half the variables at an attacker's discretion in a random sparse map carries no such reassurance.

3.2 Sparse composition factor

The idea is that any quadratic map can be written as $f \circ L$, where f is a certain standard form and L is an invertible linear map. Now we will choose L to be sparse. The obvious standard form for characteristic 2 fields is to start with the standard form ("rank form").

$$f_1(x_1, ..., x_n) = x_1x_2 + x_3x_4 + \cdots x_{n-1}x_n.$$

Let k be a field of characteristic 2. A quadratic form in n variables over k is defined by $Q = \sum_{1 \le i \le j \le n} p_{ij} x_i x_j$, $p_{ij} \in F$.

Any quadratic form over k is equivalent to

$$Q' = \sum_{i=1}^{\nu} x_i y_i + \sum_{j=\nu+1}^{\tau+\nu} \left(a_j x_j^2 + x_j y_j + b_j y_j^2 \right) + \sum_{k=1}^{d} c_k z_k^2$$

with $c_k \neq 0$ and $2\nu + 2\tau + d \le n$.

When $2\nu + 2\tau + d = n$, the form Q' is regular. The number d is the deficiency of the form and the form q' is completely regular, if Q' is regular and its deficiency is zero, which corresponding the case that the corresponding symmetric form is non-degenerate. A randomly chosen is in general expected to completely regular.

We will use this to give a general characterization of a quadratic function. Any quadratic function $f(x_1, ..., x_{2n})$ can be written in the form

$$f(x_1, .., x_n) = \sum_{1 \leq i \leq j \leq 2n} a_{ij} x_i x_j + \sum_{1 \leq i \leq 2n} b_i x_i + c$$

where a_{ij}, b_i, c are in k. We know that through a linear transformation of the form $L(x_i) = x_i + d_i$, if the quadratic part is non degenerate, we can have that

$$f(L_1(x_1, .., x_n)) = \sum_{1 \leq i \leq j \leq 2n} a'_{ij} x_i x_j + c'.$$

From the theorem above we know that there is a linear map L_2 such that

$$f((L_2 \circ L_1)(x_1, .., x_n)) = \sum_{1 \leq i \leq n} x_{2i-1} x_{2i} + \sum_{i \in S} x_i^2 + c',$$

where S is a set consisting of pairs in the form of $(2j - 1, 2j)$.

The simplest form this kind of function is surely

$$f((L_2 \circ L_1)(x_1, .., x_n)) = \sum_{1 \leq i \leq n} x_{2i-1} x_{2i} + c',$$

and its difference from others in some sense are something of the linear nature, which can be neglected in some way. From this we conclude that a general quadratic function can be represented as: $F \circ L$, where

$$F = \sum_{1 \leq i \leq n} x_{2i-1} x_{2i} + c',$$

which is the basis we will use to build our hash function.

In this particular instance, there is something that leaps out at us. starting with $X_1 := (x_1, ..., x_n)$, we compute $f_1(X_1)$, then transform $X_1 \mapsto X_2 := L_2(X_1)$, where L_2 has three randomly chosen entries in each row, and $f_2(X) := f_1(X_2)$. Continue in this vein and do $X_2 \mapsto X_3 := L_3(X_2)$, $f_3(X) := f_1(X_3)$, and so on and so forth.

Assuming 160-bit hashes (SHA-1), preliminary runs with a generic function F:

- 40 $GF(256)$ variables into 20: 890 cycles/byte (6.4 cycles/mult)
- 80 $GF(16)$ variables into 40: 1570 cycles/byte (3.6 cycles/mult)

Assuming 256-bit hashes (SHA-2), preliminary runs with a generic function F:

- 64 $GF(256)$ variables into 32: 3960 cycles/byte (6.4 cycles/mult)
- 128 $GF(16)$ variables into 64: 11420 cycles/byte (3.6 cycles/mult)

3.2.1 Other Attacks

There are many specialized attacks in multivariate public key cryptography, that one may think to use to attack our systems. But one should realize that due to the property of random polynomials, from what we can see, all but one of them is now inapplicable to attack our hash.

There is a special multivariate attack to solve under-defined systems of equations (Courtois,Goubin,Meier,Tacier) that applies to this situation where there is a lot many more variables than equations, but for fields other than $q = 2$ it has proved to be rather useless if we just plug in the numbers into the formulas.

There are the usual attacks of linear and differential cryptanalysis to think of, but quadratic equations are so far removed from linearity that it is hard to imagine such attacks working.

3.3 The challenge

How to study the security of the special constructions?

4 Conclusion

We present the idea of using randomly polynomials, and randomly polynomials with sparse property to build hash functions.

For the case of randomly polynomials, we present the provable security of the system and for the case of sparse construction, we present the main security assumptions and the theoretical challenge in its provable security.

Our work mainly is to point out a new direction in developing hash function whose security relies on a clear hard problems and therefore easier to study and understand, our work is just the beginning of this new direction and much work need to be done. We believe the multivariate hash has a very strong potential in practical applications.

Thanks and questions?