Preimage Attack on Hashing with Polynomials proposed at ICISC'06

Donghoon Chang

Center for Information Security Technologies(CIST),
Korea University, Korea
dhchang@cist.korea.ac.kr

Abstract. In this paper, we suggest a preimage attack on Hashing with Polynomials [2]. The algorithm has n-bit hash output and n-bit intermediate state. (for example, $n = 163$). The algorithm is very simple and light so that it can be implement in low memory environment. Our attack is based on the meet-in-the-middle attack. We show that we can find a preimage with the time complexity $2^{n-t} + 2^t * (n + 1/33)$ and the memory 2^t even though the recursive formula H uses any f. We recommend that hash functions such as Hashing with Polynomials should have the intermediate state size at least two times bigger than the output size.

Keywords : Hash Function, Polynomial, Preimage Attack.

1 Introduction.

Nowadays, since MD4-style hash function were broken, new style hash functions are studied intensively. On the other hand, new style hash functions have little security analysis. So, probably secure hash algorithms are more important. Since hash functions are used practically, new hash functions must be efficient also. This paper describe a preimage attack on Hashing with Polynomials which is totally different from MD4-style hash functions. This means that the algorithm must be modified. Frankly speaking, even though the algorithm is modified, we can not have a confidence that the modified algorithm is secure. This means that provably secure hash function is required.

2 Hashing with Polynomials

$R = \mathbb{F}_{2^n} = \mathbb{F}_2[x]/(p(x))$ where $p(x)$ is an irreducible polynomial of degree n. α is a root of $p(x)$. $P(\alpha)$ and $Q(\alpha)$ are two elements of R. For a bit '0' or '1', $H(0) = P(\alpha)$ and $H(1) = Q(\alpha)$. $u_i(\alpha)$, $v(\alpha)$ and $v_i(\alpha)$ are fixed elements of R. In [1], $\mathsf{f}(\mathsf{x},\mathsf{y}) = \mathsf{s} = \mathsf{x} \circ \mathsf{y} = (\mathsf{x} + u_1(\alpha)) \cdot (\mathsf{y} + u_2(\alpha)) + \mathsf{x}\cdot, v_1(\alpha) + \mathsf{y} \cdot v_2(\alpha)$. In [2], $\mathsf{f}(\mathsf{x},\mathsf{y}) = \mathsf{s} = \mathsf{x} \circ \mathsf{y} = \mathsf{x} \cdot \mathsf{y} + \mathsf{x}^2 \cdot u_1(\alpha) + \mathsf{y}^2 \cdot u_2(\alpha) + v(\alpha)$. (In this paper, we show that we can find a preimage even when f of any degree is used.) The recursive formula H is defined by $H(\mathcal{S}_1\mathcal{S}_2) = \mathsf{f}(H(\mathcal{S}_1), H(\mathcal{S}_2))$. Then, Hashing

with Polynomials is defined like Fig. 1 and Fig. 2. For example, the author [1, 2] suggests particular polynomials for Hashing with Polynomials as follows: $p(x) = x^{163} + x^7 + x^6 + x^5 + x^4 + x + 1$, $P(\alpha) = \alpha^7 + 1$, $Q(\alpha) = \alpha^8 + 1$, $u_1(\alpha) = \alpha^2$, $u_2(\alpha) = \alpha$, $v(\alpha) = v_1(\alpha) = 1$ and $v_2(\alpha) = \alpha$. However, our attack does not depend on $p(x)$ and f, i.e. we can find a preimage for any polynomial $p(x)$ of any degree and f of any degree.

$\underline{\mathsf{HashPoly}}(M) = O$ (n-bit hash output)
M is any bit length (at least 2 bits) message for which $M = m_0||m_1||\cdots||m_{t-1}$ and each m_i is 32-bit for $0 \leqslant i \leqslant t-2$ and m_{t-1} has 32 bits at most.

1.Compute the hash h_i of each block m_i independently, going left to right bit by bit and using the recursive formula H.
2.Compute the hash of M inductively, going left to right block by block and using the recursive formula H.
3. Output O which is the final n-bit value.

Fig. 1. Hashing with Polynomials.

.

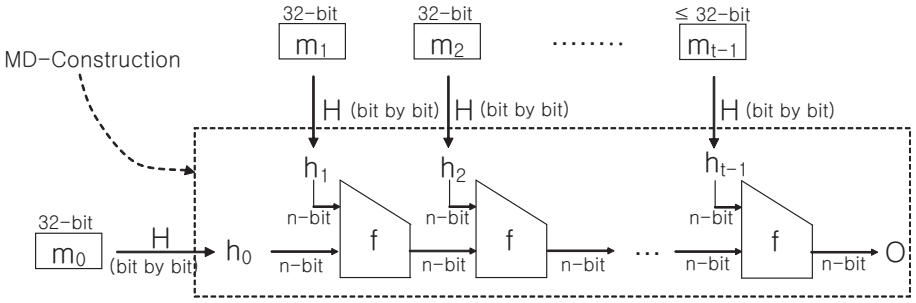

Fig. 2. Hashing with Polynomials.

3 Preimage Attack on Hashing with Polynomials

This section, we show a preimage attack on Hashing with Polynomials [1, 2].

Easy to Invert Function f in [1]

Here, we show that it is easy to invert f.

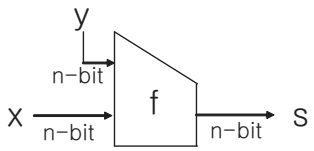

Fig. 3. In [2], $\mathsf{f}(\mathsf{x},\mathsf{y}) = \mathsf{s} = \mathsf{x} \circ \mathsf{y} = (\mathsf{x} + u_1(\alpha)) \cdot (\mathsf{y} + u_2(\alpha)) + \mathsf{x} \cdot v_1(\alpha) + \mathsf{y} \cdot v_2(\alpha)$.

$$\mathsf{f}(\mathsf{x},\mathsf{y}) = \mathsf{s} = \mathsf{x} \circ \mathsf{y} = (\mathsf{x} + u_1(\alpha)) \cdot (\mathsf{y} + u_2(\alpha)) + \mathsf{x} \cdot v_1(\alpha) + \mathsf{y} \cdot v_2(\alpha)$$

Given any value s, we choose a y and then we can easily get x satisfying above equation because all terms are fixed values except x.

Easy to Invert Function f in [2]

$$\mathsf{f}(\mathsf{x},\mathsf{y}) = \mathsf{s} = \mathsf{x} \circ \mathsf{y} = \mathsf{x} \cdot \mathsf{y} + \mathsf{x}^2 \cdot u_1(\alpha) + \mathsf{y}^2 \cdot u_2(\alpha) + v(\alpha)$$

Given any value s, we choose a y and then we want to get x satisfying above equation. Since operations used in above equation are defined over $\mathbb{F}_2[x]/(p(x))$, above equation is a linear transformation on x when y is fixed. So, above equation can be expressed as the following matrix operation where $\mathsf{x} = (x_0||x_1||x_2||\cdots||x_{n-1})$ and c_i is 0 or 1 and M is a $n \times n$ matrix whose element is 0 or 1.

$$(x_0||x_1||x_2||\cdots||x_{n-1})\mathsf{M} = (c_0||c_1||c_2||\cdots||c_{n-1})$$

Therefore, we can find x with bit operation complexity about n^3 by using Gaussian elimination method. If y is random, we can expect that we can find a solution on average. Now we compute the complexity required to make a matrix M and constants c_i's. We define the complexity 1 as the time of simulating compression function including one f and one H processing one 32-bit message block. H have to use f thirty two times for one 32-bit message block. So, we can make a matrix M and constants c_i's with complexity 1/33. And f requires n^2 bit operation complexity at least because f uses the multiplication. Therefore, we can use the Gaussian elimination method with complexity n at most.

Easy to Invert Function f of any degree

We want to generalize our attack for any f of any degree. For any f, when y are fixed, f is a linear transformation on x. So we can express f as the matrix operation. So, we can apply above result similarly.

the Meet-in-the-Middle Attack

Hashing with Polynomials [1, 2] uses Merkle-Damgård-construction and use a function f easy to invert. And the size of intermediate value is same as that of the hash output value. This means we can apply the meet-in-the-middle attack as follows.

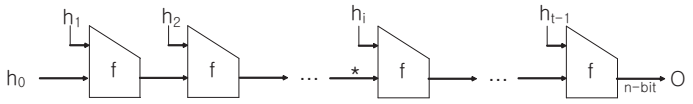

Fig. 4. Merkle-Damgård Construction : Meet-in-the-Middle Attack.

Given an output O, we choose any $m_i \sim m_{t-1}$ and get $h_i \sim h_{t-1}$ corresponding to them by the recursive function H. Then we can know the value in $*$ in Fig. 4 and store it in a table. Like this, we repeat to choose $m_i \sim m_{t-1}$ 2^t times and repeat to store the value in $*$ in Fig. 4 by inverting f as explained above. Then we choose randomly $m_0 \sim m_{i-1}$ and get the value in $*$ and then check if the value is in the table. If there is the value in the table, we can know a preimage of the hash output O. According to the birthday attack, it is enough to repeat to choose $m_0 \sim m_{i-1}$ 2^{n-t} times. Therefore, we can find a preimage with the time complexity $2^{n-t} + 2^t * (n + 1/33)$ and the memory 2^t.

4 Conclusion

In this paper, we show a preimage attack on Hashing with Polynomials. Our attack does not depend on f of any degree. We comment that the size of the intermediate value should be at least two times bigger than that of the hash output. Even though the algorithm is modified, more careful analyses on it are required.

Acknowledgement

We thank prof. Jaechul Sung for his valuable comment.

References

1. V. Shpilrain, *Hashing with Polynomials*, http://www.sci.ccny.cuny.edu/ shpil/hashpoly.pdf.
2. V. Shpilrain, *Hashing with Polynomials*, ICISC'06, LNCS 4296, pp. 22-28, 2006.