Dickson polynomial을 이용한 안전한 해쉬함수의 설계

양 대헌*○ 송 주석*

* 연세대학교 컴퓨터 과학과 정보통신 연구실

The Design of a Secure Hash function using Dickson Polynomial

Nyang Dae Hun* Song Joo Seok*

* Computer and communication lab, computer science department of Yonsei University

요 약

본 논문에서는 디지탈 서명에 사용되는 해쉬함수를 제안해 본다. 현재까지 제안되어온 해쉬함수는 고정된 크기의 블럭들에 대한 복잡한 연산을 통해서 안전성을 확보하고있고, 수학적으로 안전성이 증명되고 있지 않다. 본 논문에서는 permutation polynomial의 일종인 Dickson polynomial에 대해 알아보고, 충돌회피성과 일방향성을 갖는 Dickson polynomial을 이용해서 수학적으로 안전한 해쉬함수를 설계해 본다.

1. 디지탈서명

정보 통신 기술과 컴퓨터의 발달로 이제는 네트웍을 통한 편지나 결재등이 가능해지게 되었다. 아직은 email 수준으로 사용하고 있지만 조금 더 시간이 흐른 후에는 전자결재 시스템, 화상회의 시스템, 스마트 카드등의 등장으로 전송된 메시지의 *validation*과 *authentication*이 요구하게 된다. 현재는 이를 위해서 메시지를 작성한 작성자가 문서에 직접 자신의 서명을 써넣지만, 디지탈 서명은 단순히 0 과 1 의 나열이기 때문에 이런 서명방법을 사용한다면 위조가 가능하다. 따라서 디지탈 서명은 모든 메시지에 대해서 다른 서명을 생성해야 한다. 위의 성질에서 본다면 디지탈 서명은 어떤 비밀 정보를 가지고 있는 서명자에 의해서만 계산되는 값이고, 메시지에 따라서 달라지는 값이다. 이 디지탈 서명에 의해서 *authentication*와 *validation*을 해결할 수 있다. 디지탈 서명을 생성하는데 암호화가 필요하고 메시지가 큰 경우 서명의 생성시간이 길어진다. 따라서 해쉬함수를 통해 메시지를 압축하게되고 안전한 해쉬함수를 설계하는 것이 중요한 문제가 된다.

2. 해쉬 함수

해쉬 함수는 어떤 함수 h 에의해서 다음과 같이 생성된다.

$$H = h(M)$$

여기서, M은 길이가 정해지지 않은 메시지이고, $h(M)$은 고정된 길이의 해쉬값이다. 해쉬값은 송신인이 자신이 작성한 최종 메시지에 덧붙이게 되고, 수신인은 수신된 메시지에 대해서 자신이 계산한 해쉬값과 송신인이 덧붙인 해쉬값을 비교해서 메시지의 변경여부를 확인한다. 이때 해쉬함수 자체에는 비밀성에 대한 보장이 없으므로, 해쉬값만으로는 비밀성을 보장할 수 없다. 대신 공개키 시스템을 이용해서 송신인의 비밀키로 이 해쉬값을 암호화한다.

해쉬함수의 목적은 주어진 파일의 "finger print"를 만드는 것이므로, 이것이 메시지의 인증에 유용하기 위해서는 다음의 조건들을 만족시켜야 한다.

1. 어떤 크기의 데이타 블럭에도 적용가능해야 한다.
2. 고정된 길이의 출력을 생성해야 한다.
3. $h(x)$의 계산이 어떤 주어진 x에 대해서도 계산하기 쉬워야한다.
4. 약 일방향성(weak one-wayness) : 해쉬값 H 로부터 $h(M) = H$되는 메시지 M을 거꾸로 구해내는 일은 계산상 불가능하다.
5. 강 일방향성(strong one-wayness) : 어떤 메시지 M과 그 해쉬값 H가 주어졌을때, $h(M') = H$ 되는 메시지 $M' \neq M$을 찾는 일은 계산상 불가능하다.
6. 충돌 회피성(collision freeness) : $h(M) = h(M')$되는 메시지 $M \neq M'$을 찾는 일은 계산상 불가능하다.

3. Permutation Polynomial

이 절에서는 Permutation polynomial과 이의 일종인 Dickson polynomial에 대해서 알아본다.

$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ 이 $q = p^\theta$ 의 차수를 갖는 유한체 F_q위에서 n차의 polynomial이고, $\theta \geq 1$, p가 소수일때 $f : c \rightarrow f(c)$ 가 F_q에서 자신으로의 one to one mapping이라면 $f(x)$를 *permutation polynomial*이라 한다. 이 permutation polynomial에 대해서는 아직 많이 알려지지 않았고, 어떤 polynomial이 permutaion polynomial인지를 검사하는 알고리즘도 아직 많이 알려지지 않았다. 최근에 안전한 데이타의 전송을 위한 cryptosystem에서 permutation polynomial이 많은 관심을 끌게 되었다.

Theorem 1 (Hermite's Criterion) $f(x)$가 F_q를 *permute*한다는 것은 다음과 동치이다.

1. f가 F_q에서 근을 하나만 가지고
2. $1 \leq t \leq q-2$이고 $t \neq 0 \ mod \ p$인 t에대해서, $[f(x)]^t \ mod \ x^q - x$ 의 *degree*가 $q-2$보다 작거나 같다.

1897년에 Dickson은 유한체 위에서 7보다 작은 차수를 갖는 permutation polynomial을 분류했다.

다음은 permutation polynomial의 몇가지 예를 보여준다.

- 모든 일차다항식은 permutation polynomial이다.
- x^k가 F_q를 permute한다는 것은 $(k, q-1) = 1$과 동치이다.

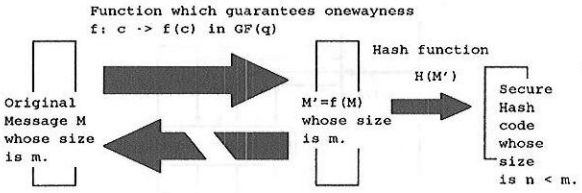

그림 1: 안전성을 위한 onewayness의 이용

- $a \in F_q$ 일때, Dickson polynomial

$$g_k(x,a) = \sum_{j=0}^{\lfloor k/2 \rfloor} \frac{k}{k-j}\left(\frac{k-j}{j}\right)(-a)^j x^{k-2j}$$

가 F_q를 permute한다는 것은 $(k, q^2-1) = 1$과 동치이다.

- $r > 1$ 이 $q-1$과 서로소이고, s가 $q-1$을 나누고, $g(x^s)$가 F_q에서 0이 아닌 근을 갖지 않는다면$(g(x) \in F_q[x])$, $x^r(g(x^s))^{(q-1)/s}$는 F_q를 permute한다.

Definition 1 어떤 정해진 $a \in F_q$ 에 대해, $P(a)$는 F_q의 *permutation polynomial*인 모든 *Dickson polynomial* $g_k(x,a)$의 집합 $P(a)$는 다음과 같다. $P(0) = \{g_k(x,0) : k \in N, gcd(k, q-1) = 1\}$, $P(a) = \{g_k(x,a) : k \in N, gcd(k, q^2-1) = 1\}$ *for* $a \neq 0$.

Theorem 2 $P(a)$가 합성의 연산에 대해 닫혀있다는 것은 $a = 0, 1,$ 또는 -1 과 동치이다.

4. Dickson Polynomial을 이용한 해쉬 함수의 안전성 강화

현재까지 연구되어 오고있는 해쉬 함수는 주로 $h : V_\infty \rightarrow V^k$ 또는 귀납적으로 $f : V^{m+k} \rightarrow V^k$로 정의되었다. 해쉬 함수가 충돌 회피성을 가지려면 그 해쉬함수에 해당하는 기초함수가 충돌 회피성을 가져야하지만, $|m+k| > |k|$ 이므로 충돌 회피성을 가지는 기초함수 $f : V^{m+k} \rightarrow V^k$를 만들기는 쉽지않다. 지금까지 제시되어온 거의 모든 해쉬함수는 고정된 크기의 블럭들에대한 복잡한 연산을 통해서 안전성을 확보하고 있고, 수학적으로 안전성이 증명되어 있지 않은 함수들이다. Permutation polynomial은 $f : c \rightarrow f(c)$ 이므로, 어떤 메시지의 집합 $S = \{M_1, M_2, \cdots, M_N\}$을 이 다항식에 의해 매핑한다면, 같은 크기의 집합 $S' = \{M_1', M_2', \cdots, M_N'\}$로 매핑된다. 이 다항식에의한 매핑은 collision freeness를 만족하고, 만약 이 다항식이 onewayness의 성질을 가지고 있다면, 이를 해쉬 함수로 이용할 수 있다. 그림 1은 이 개념을 보여준다.

다음은 해쉬함수의 안전성을 위한 조건과, permutation polynomial을 이용한 안전성 강화를 보여준다.

- **충돌 회피성(Collision freeness)**
  $h(D(M)) = h(D(M'))$ 되는 메시지 $M \neq M'$를 찾으려면, 우선 $D(M) \neq D(M')$인 $D(M')$을 계산해야하고, 만약 위조자가 현재 사용되고 있는 해쉬 함수의 공격에 성공했다 하더라도 실제 필요한 메시지 M'를 찾으려면 $D^{-1}(M')$를 구해야한다. 이것은 D^{-1} 가 계산상 불가능한 permutation polynomial을 선택하므로써 불가능해진다.

  다른 공격방법으로 그림 2에서 처럼 $D^{-1}(M')$의 계산을 거치지 않고 직접 위조된 메시지 M'을 구할 수 있다. 예를 들어 Dickson polynomial 계산 후 적용하는 메시지 압축알고리즘이 위치의 치환에 민감하지 않다면 원래 메시지의 내용중의 일부를 다른 부분과 교환해서 쉽게 충돌이 일어나게 할 수 있다. 하지만 이런 공격방법은 함께 사용하는 메시지 압축알고리즘이 위치의 치환에 민감

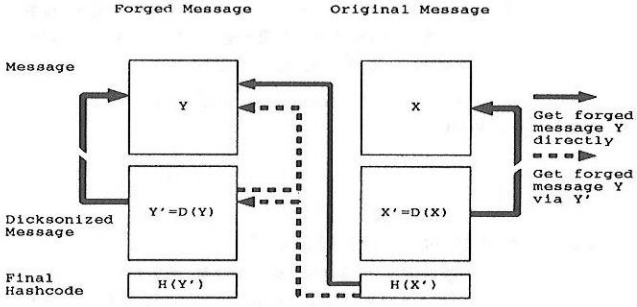

그림 2: 제안한 해쉬함수의 공격방법

한 것같은 어떤 특별한 성질을 갖는다면, 현재 사용되는 해쉬 함수보다 단순하지만 안전한 해쉬함수를 설계할 수 있다.

- **강 일방향성(Strong onewayness)**
  메시지 M과 $H = h(D(M))$ 이 주어졌을때, $h(D(M')) = H$ 인 M'을 찾으려면, $h(D(M')) = H$인 $D(M')$을 찾아야하고, 이는 $D^{-1}(M')$의 계산을 요구한다. 이는 D^{-1}의 계산상 불가능성에의해 안전을 보장 받는다. 앞에서 알아본 다른 공격방법에는 여기서도 적절한 메시지 압축알고리즘에의해 안전을 보장 받을 수 있다.

- **약 일방향성(Weak onewayness)**
  해쉬값 H로부터 $h(D(M)) = H$ 되는 M을 거꾸로 계산하려면, $D(M)$을 거꾸로 계산해야하고, 결국 D^{-1}을 계산해야 한다.

위의 조건들을 살펴보았을때 D^{-1}의 계산이 쉽지 않다면, 좋은 해쉬함수를 구성할 수 있음을 알 수 있다. 3절에서 살펴본 Dickson polynomial은 이 조건에 잘 부합한다. 즉, Dickson polynomial $g_k(M,a)$ 의 a 항을 $0, \pm 1$로 선택하지 않는다면, 정리 2 에의해 $P(a)$는 합성의 연산에 대해 닫혀 있지 않고, 이는 Dickson polynomial $g_k(M,a)$의 역함수를 구하는것이 어렵다는것을 의미한다.

4.1 Dickson polynomial을 이용한 해쉬 함수

다음은 Dickson polynomial을 이용해서 해쉬값을 계산하는 알고리즘이다.

1. 우선 q(q는 소수의 멱)를 선택한다.
2. 메시지를 여러 블럭으로 나누고, 각각을 정수로 표현하므로써 메시지 M을 0에서 $n-1$까지의 수로 나타낸다.
3. 선택된 q 와 어떤 Dickson polynomial $g_k(M,a)$ 에 대해서, $(q^2-1, k) = 1$ 을 만족하는 k를 선택한다.
4. $g_k(M,a)$의 a는 $0, \pm 1$이 아닌 임의의 수를 선택한다.
5. 선택된 q, k, a로 다음을 계산한다.

$$g_k(M,a) = \sum_{j=0}^{\lfloor k/2 \rfloor} \frac{k}{k-j}\left(\frac{k-j}{j}\right)(-a)^j M^{k-2j} \ mod\ q$$

6. 앞에서 선택한 q, k, a를 공개한다.

4.2 제안한 해쉬함수의 효율적인 계산 알고리즘

4.2.1 해쉬함수의 효율적인 계산을 위한 방법

이제 앞에서 제안한 해쉬함수를 효율적으로 계산할 수 있는 알고리즘을 알아본다. 우선 효율적인 polynomial의 계산을 위해서 *Horner's rule*을 이용한다.

Rule 1 (Horner's rule)

$$\begin{aligned} u(x) &= u_n x^n + u_{n-1}x^{n-1} + \cdots + u_1 x + u_0, u_n \neq 0 \\ &= ((\cdots(u_n x + u_{n-1})x + \cdots)x + \cdots)x + u_0 \end{aligned}$$

임의의 *polynomial* $u(x)$를 계산하는데 n번의 곱셈과 n번의 덧셈이 필요하고, 만약 계수가 0이라면 그 항만큼의 덧셈은 필요치 않다. 또 중간의 계산값을 임시로 저장할 필요가 없다.

Dickson polynomial에는 combination의 계산이 있고 이는 다음의 식을 이용한다.

$$\begin{pmatrix} a-1 \\ b+1 \end{pmatrix} = \frac{(a-b)(a-b-1)}{a\cdot(b+1)} \cdot \begin{pmatrix} a \\ b \end{pmatrix} \qquad (1)$$

4.2.2 상세한 해쉬계산 알고리즘

다음은 앞의 결과들을 이용해서 해쉬함수를 계산하는 알고리즘을 보인다. Stage 1에서는 Dickson polynomial의 계수를 구하고, Stage 2에서는 Stage 1에서 구해진 계수를 이용해서 Dickson polynomial을 계산한다.

Stage 1: G는 Dickson polynomial의 계수, C는 조합의 계산을 위한 버퍼

- **Step 1:** $C \leftarrow 1, G[0] \leftarrow 1, a \leftarrow 1, j \leftarrow 1$
- **Step 2:**
$$C \leftarrow \frac{(k-2j+2)\cdot(k-2j+1)}{(k-j+1)\cdot j} \; mod\; q, a \leftarrow (-a)\cdot a$$
- **Step 3:** $G[j] \leftarrow a \cdot C \; mod\; q$
- **Step 4:** $j \leftarrow j+1$
- **Step 5:** Repeat Step 2, 3, 4, 5 until $j = \lfloor \frac{k}{2} \rfloor + 1$

Stage 2: Horner's rule을 이용하며, x를 x^2으로 치환해서 계산한다. 마지막 계산 결과는 H에 저장된다.

- **Step 1:** $i \leftarrow n, x \leftarrow x^2$
- **Step 2:** $g_i \leftarrow g_i x^n + g_{i-1} \; mod\; q$
- **Step 3:** $i \leftarrow i-1$
- **Step 4:** Repeat Step 2,3 until $i = 0$
- **Step 5:** $g_1 \Rightarrow H$

5. 제안한 Dickson polynomial과 메시지 압축 알고리즘의 결합

앞 절에서 제시한 permutation polynomial만을 이용한 매핑으로는 $h : V_\infty \rightarrow V^k$처럼 고정된 크기의 해쉬값을 얻을 수 없다. 따라서 이 매핑 후에 적절한 메시지 압축 알고리즘을 이용해서 고정된 크기의 해쉬값을 생성해야 한다. 이 절에서는 이 메시지 압축 알고리즘의 조건을 제시해 보고, 그 조건을 만족하는 메시지 압축 알고리즘을 제안해 본다.

5.1 메시지 압축 알고리즘의 조건

앞에서 제시한 Dickson polynomial을 이용하는 해쉬함수는 직접 $H(x)$로 부터 D^{-1}을 거치지 않고 $H(x) = H(y)$인 $y \neq x$를 구할 수 있으므로, 이 공격에 안전해야 한다. 하지만 이 공격 방법은 permutation polynomial을 통해 매핑했다는 사실을 무시하고 원래의 메시지에 대해 공격하는 것이므로 매우 어렵다. 하지만 위치의 치환등의 공격에 안전해야 하고, 해쉬값의 모든 비트가 모든 입력 비트의 함수이어야한다. 또 Dickson polynomail 연산을 거치므로 상대적으로, 빠른 연산시간이 요구된다.

5.2 제안한 Dickson polynomial과 메시지 압축 알고리즘의 결합

여기서는 M_i를 Dickson polynomial의 입력으로해서 계산한 결과를 M_{i+1}과 modulo 덧셈해 나가는 메시지 압축알고리즘을 택했다. 이렇게 했을 경우 위치치환에 sensitive하게되고, Dickson

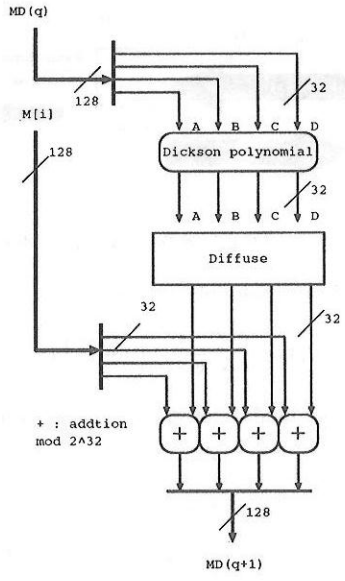

그림 3: 128비트 메시지블럭의 처리

polynomial이 onewayness를 만족하므로, meet-in-the-middle 공격등에 안전하다.

사용될 메시지 압축 알고리즘은 단순히 위치 치환의 공격과 해쉬값의 모든 비트가 입력비트의 함수이기만 하면 되므로 1라운드로 구성된다. 메시지 블럭의 크기는 128비트이고, 각각은 4개의 32비트 서브 블럭으로 나누어진다. 중간의 계산결과를 저장하고 있는 4개의 32비트 버퍼 A,B,C,D 가 필요하며, 이 알고리즘에서 사용하는 모든 덧셈은 2^{32} modulo 연산으로 정의된다. 128비트의 메시지블럭 한개에 대해서 4개의 32비트 서브 블럭이 존재하고, 네개의 32비트 버퍼가 존재하므로 한 스텝 후에 한개의 메시지 블럭이 처리된다. 마지막으로 *diffusion*을 높이기 위해서 다음의 diffusion 함수를 사용한다.

```
A = B + C + D, B = A + C + D
C = A + B + D, D = A + B + C
```

다음은 제안한 메시지 압축 알고리즘이다.

- **Step 1:Padding비트 덧붙임**
메시지의 길이가 0 modulo 128와 *congruent*하게 ($length \equiv 0\; mod\; 128$) 패딩된다. 패딩 비트는 항상 덧붙이는데, 만약 메시지의 길이가 위의 조건을 만족하더라도 128비트를 패딩한다. 따라서 패딩비트의 숫자는 1에서 128 사이이다. 패딩되는 내용은 맨처음 "1" 그리고 필요한 만큼의 "0" 이다.
- **Step 2: Length 덧붙임**
원래 메시지 비트길이(패딩되기 전)의 64비트 표현으로 스텝1의 결과에 패딩된다. 만약 길이가 2^{64}보다 크다면 표현된 길이의 하위 64비트만을 기록한다. 따라서 이 필드는 메시지 길이의 modulo 2^{64}값을 갖고 있게된다.
- **Step 3:버퍼 초기화와 Dickson polynomial의 파라미터 결정**
4개의 버퍼 A,B,C,D는 다음의 값으로 초기화 한다.

```
A = 0 1 2 3 4 5 6 7, B = 8 9 A B C D E F
C = F E D C B A 9 8, D = 7 6 5 4 3 2 1 0
```

유한체 F_q위에서의 Dickson polynomial $g_k(M, a)$의 파라미터는 q, k와 a이고, 각각 다음과 같이 선택한다.

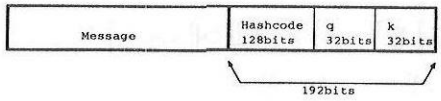

그림 4: 제안한 해쉬함수를 적용한 메시지형태

q는 $2^{16}-1$을 넘지 않는 5자리 소수 들을 선택한 후 그들의 곱으로 정의 한다. k는 $gcd(k, q^2-1)=1$을 만족하는 q를 넘지 않는 10자리 정수를, a는 Step 2에서 계산된 length field의 하위 8비트를 선택해서 a의 값으로 한다. 이때 만약 a가 0 또는 ± 1이라면 $a \leftarrow a+2^4$한다. $k<q$인 k를 선택하는 것은 $k>q$인 경우 $k-q \rightarrow k$가 되므로 차수가 낮은 다항식이 되기 때문이다.

- **Step 4:또 다른 Padding비트 덧붙임**
  Step 3에서 정한 k와 q의 위조를 막기위해 각각 32비트, 모두 64비트를 Step 1,2에서 생성된 메시지에 덧붙인다. a는 메시지 길이의 함수이고, 이미 length비트를 덧붙였으므로 덧붙일 필요가 없다.

  위의 네 스텝이 끝나면, 메시지의 비트 길이는 128의 배수가 된다.

- **Step 5:128비트의 메시지블럭 처리**
  이 단계에서는 4개의 버퍼값을 몇가지 연산을 통해서 결정하고, 각 버퍼의 내용을 저장한다.

  4개의 32비트 버퍼의 Dickson polynomial값을 각각 계산한 후 저장한다.

  이제 각 버퍼에는 이전 라운드에서 계산된 결과의 Dickson polynomial값이 저장되어있고, 각 버퍼의 내용을 앞에서 정의한 연산 $Diffuse$에의해 갱신한다. 이제 이 값과 다음 메시지 서브블럭과의 2^{32} modulo 덧셈값을 각 버퍼에 저장한다. 이 값이 한 단계가 끝난후의 값이며, 다시 Dickson polynomial의 입력으로 주어진다. 이 과정이 모든 메시지 블럭이 처리될때까지 계속된다.

  그림 3은 Step 5의 계산과정을 보여준다.

- **Step 6:출력**
  위의 네 스텝이 끝나면, 4개의 32비트 버퍼 A,B,C,D에는 최종 해쉬값이 남게된다.이것이 128비트의 해쉬값이다.

- **Step 7:서명의 덧붙임**
  Step 3에서 결정된 값은 수신측에서 해쉬 값의 계산을 위해 메시지의 특정부분에 padding되어야 한다. 즉 Step 6까지 계산된 결과 128비트와 q,k 각각 32비트씩 64비트가 원래의 메시지에 덧붙여진다. 원래의 메시지 M에 $M||q||k$와 같이 덧붙인다. 그림 4은 최종 메시지 형태를 보여준다. 이때 서명은 $M||q||k$가 되고 이 부분을 자신의 비밀키로 암호화한다. 따라서 192비트를 암호화 해야한다.

6. 제안한 해쉬함수의 공격방법

6.1 Dickson Polynomial $g_E(x,a)=y \bmod q$ 의 decryption key를 구하는 방법

정리2에서 Dickson polynomial $g_k(x,a)$의 집합 $P(a)$는 $a \neq 0, \pm 1$ 인 경우, 합성의 연산에 대해 닫혀있지 않음을 알 수 있다. 따라서, E 와 a가 공개되어 있고, $g_E(x,a)$로 해쉬값을 계산했을때, 이것의 역함수 $g_D(x,a)$는 존재 하지 않는다. 여기서 D는 $x=g_D(g_E(x,a),a)$를 만족시키는 값이다. 따라서 encryption key E가 알려져 있지만, 이를 decryption하려면 D 뿐만 아니라, b를 알고 있어야 한다. 즉,$g_D(g_E(x,a,),b)=x$ 를 만족하는 b와 D를 구해야하고, 이는 쉽지 않다.

6.2 방정식 $g_E(x,a)=y \bmod q$의 해를 구하는 방법

$$g_E(x,a)=\sum_{j=0}^{\lfloor E/2 \rfloor} \frac{E}{E-j}\binom{E-j}{j}(-a)^j x^{E-2j} = y \bmod q \quad (2)$$

n차의 일반적인 polynomial의 해를 계산하는 방법은 여러가지가 존재하고, 그중에 주목할 만한 것으로 Berlekcamp의 알고리즘이 있다. Berlekcamp의 알고리즘은 $O(qn^3)$의 시간 복잡도를 가진다. 하지만 Berlekcamp의 알고리즘은 F_q에서 큰 q에 대해서는 계산이 불가능하다. 이것은 q개의 값에 대한 최대 공약수를 모두 구해야하고 이것은 매우 어려운 일이기 때문이다. 이것을 어느정도 해결한 방법이 있는데, 이것은 d차의 $irreducible$ polynomial $p(x)$가 $x^{p^d}-x$를 나누고, $x^{p^t}-x(t<d)$는 나누지 않는다는 사실을 이용한다. 이 방법을 사용하면 $O(n^3(\log p)^2+n^2(\log p)^3)$의 시간 복잡도를 가지고 이것은 Berlekcamp의 알고리즘보다 우수하다. 또 1981년에는 Cantor와 Zassenhaus가 $Best\ case$에 $O(n^2 \log p)$의 복잡도로 위의 식의 해를 구할 수 있음을 보였다.

이 논문에서 제안한 q와 k는 모두 32비트의 수이므로, Cantor와 Zassenhaus가 제안한 방법으로 $best\ case$에 $\frac{1}{2}^{\log p}$의 확률로 문제가 풀린다고 하더라도 $O((2^{32})^2 \cdot log 2^{32}) = O(2^{69})$ 만큼의 연산을 해야하므로 이는 계산상 불가능하다.

6.3 q,k를 변경하는 방법

q나 k를 변경한다는 것은 일단 해쉬값을 변경한다는 것이고, 이 공격방법으로는 strong onewayness를 공격할 수 없다. 단지 collisioin freeness의 공격이 가능할 수 있지만, 128비트의 해쉬값이 q와 k의 함수이기 때문에 이를 변경하면 해쉬값도 변경된다. 따라서 $H(X)=H(X')$인 $X \neq X'$ 쌍을 찾는것은 불가능하다.

6.4 그 외 다른 공격방법

Meet-in-the-middle공격방법이나 birthday paradox를 이용한 공격방법에 안전하기 위해서는 해쉬값의 길이를 적어도 128비트 이상으로 하면 된다. 제안한 해쉬함수는 onewayness의 성질을 이용하므로 meet-in-the-middle공격방법에 안전하다. 또 만약 제안한 해쉬함수와 함께 사용한 메시지 압축알고리즘이 공격당했다 하더라도, Dickson polynomial의 onewayness에의해 보호 받을 수 있다.

Differential cryptanalysis에는 함께 사용한 메시지 압축알고리즘에 XOR연산보다는 MD5처럼 2^{32} modulo 덧셈연산을 이용했으므로 안전하고, 마찬가지로 이 압축알고리즘이 공격당하는경우에도 제안한 permutation polynomial에의해 보호 받는다. 따라서 이런 종류의 공격에는 안전하다고 할 수 있다.

참고 문헌

[1] Rudolf Lidl and Harald Niederreiter, "Finite fields, Encyclopedia of mathematics and its applications", Vol 20., Cambridge University Press 1984.

[2] R.L. Rivest, A.Shamir, and L. Adleman, "A Method for obtaining digital signatures and public-key cryptosystems", Comm. ACM, Vol 21, No 2, Feb, 1978.

[3] Philip Zimmermann, "A Proposed Standard Format for RSA Cryptosystems", IEEE Computer, Sep, 1983.

[4] Rudolf Lidl and Gary L. Mullen, "When Does a Polynomial over a Finite Field Permute the Elements of the Field?", The American Mathematical Monthly, Vol 95, No 3, Mar, 1988.