The design of a secure hash function using Dickson polynomial

Dae – Hun Nyang○, *Seung – Joon Park*, *Joo – Seok Song*

Dept. of Computer Science, Yonsei University
134, Shinchon-Dong, Seodaemun-Ku,
Seoul 120-749, Korea

Abstract

Almost all hash functions suggested up till now provide security by using complicated operations on fixed size blocks, but still the security isn't guaranteed mathematically. The difficulty of making a secure hash function lies in the collision freeness, and this can be obtained from permutation polynomials. If a permutation polynomial has the property of one-wayness, it is suitable for a hash function. We have chosen Dickson polynomial for our hash algorithm, which is a kind of permutation polynomials. When certain conditions are satisfied, a Dickson polynomial has the property of one-wayness, which makes the resulting hash code mathematically secure. In this paper, a message digest algorithm will be designed using Dickson polynomial.

1 Hash function

A hash code is generated by a hash function as follows, where M is a variable size message, and $h(M)$ is the fixed size hash code.

$$H = h(M)$$

A hash code is appended to a message and the receiver verifies it by comparing the hash code computed by himself to a received one. Since a hash function has no ability to prevent forging, we cannot work only with a hash function. Instead, using a public key cryptosystem, the hash code is encrypted by the signer's private key.

Let $V = \{0,1\}$. We define $V^1 \cup V^2 \cup V^3 \cup \cdots \cup V^m$ and $V^1 \cup V^2 \cup V^3 \cup \cdots$ as V_m, V_∞, respectively, where V^k means the k-fold Cartesian product of V.

Definition 1 *For a function $h : V_\infty \rightarrow V^k$, if it is computationally infeasible to find $x \neq x'$ which satisfies $h(x) = h(x')$, h is computationally one to one, and a pair of strings such as x, x' is called the colliding pair of h.*

Definition 2 *If a function $h : V_\infty \rightarrow V^k$ satisfies the following conditions, h is a (k-bit) one-way hash function, and the one-way hash function, which is computationally one to one is a collision free one-way hash function(Collision freeness).*

1. *Given a string $x \in V_\infty$, h is efficiently computed.*
2. *Given H, it is computationally infeasible to find x which satisfies $H = h(x)$(Weak one-wayness).*
3. *Given x and $h(x)$, it is computationally infeasible to find the pair $x \neq x'$ which satisfies $h(x) = h(x')$(Strong one-wayness).*

It is not easy to define a collision free one-way hash function. Thus a hash function $h : V_\infty \rightarrow V^k$ is defined recursively as follows, which is normally based on a primitive function $f : V^{m+k} \rightarrow V^k$.

1. Represent a message M as the concatenation of m-bit strings: $M = M_1||M_2||\cdots||M_N$.
2. Initialize $H_0 = I$.
3. $H_i = f(M_i, H_{i-1})(i = 1, 2, 3, \cdots, N)$
4. $H = h(M, I) = H_N$(Hash code).

A primitive function f is defined as $f(X, Y) = E_X(Y)$, where $E_X(Y)$ means the encryption of Y with X. The following theorem represents the relationship between a hash function and its primitive function. It is necessary to make a secure primitive function when making a secure hash function[1].

Theorem 1 *For h whose primitive function is f, h is collision free if and only if f is collision free.*

2 Permutation Polynomials and Dickson Polynomials

When $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ is a polynomial over a finite field F_q and its degree is $q = p^\theta$ for a prime number p, if $f : c \rightarrow f(c)$ is a one to one mapping from F_q to F_q, $f(x)$ is a *permutation polynomial.* The permutation polynomial is not still well known, and it is very hard to determine whether a polynomial is a permutation polynomial or not. Recently, in the area of cryptosystem for the secure transferring of a message, it has drawn much attention. The following theorem shows Hermite's criterion for deciding whether a polynomial is a permutation polynomial or not[2].

Theorem 2 (Hermite's Criterion) *$f(x)$ permutes F_q if and only if*

1. *$f(x)$ has exactly one root in F_q and*
2. *for each integer t with $1 \leq t \leq q-2$ where $t \neq 0$ mod p, the reduction of $[f(x)]^t$ mod $x^q - x$ has degree D ($D \leq q-2$).*

In 1897, Dickson categorized permutation polynomials with a degree below 7 over finite fields. The following is some examples of permutation polynomials.

- All linear polynomials are permutation polynomials.

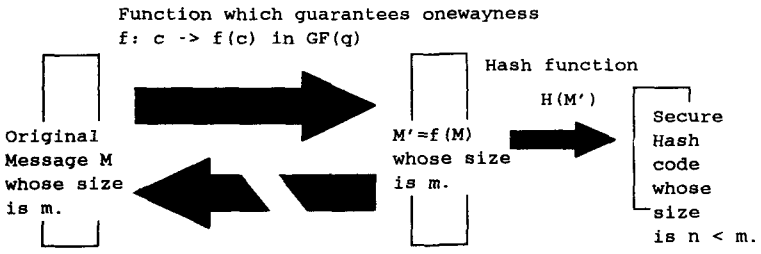

Figure 1: Using the property of one-wayness for security

- x^k permutes F_q if and only if $(k, q-1) = 1$
- For $a \in F_q$, the following Dickson polynomial permutes F_q if and only if $(k, q^2-1) = 1$.

$$g_k(x,a) = \sum_{j=0}^{\lfloor k/2 \rfloor} \frac{k}{k-j} \binom{k-j}{j} (-a)^j x^{k-2j} \tag{1}$$

A Dickson polynomial has interesting properties. It is computationally infeasible to get the inverse of a Dickson polynomial under some conditions. In order to explain it, we define $P(a)$ as follows.

Definition 3 *For fixed $a \in F_q$, the set $P(a)$ of all Dickson polynomials $g_k(x,a)$ that are permutation polynomials of F_q is as follows.* $P(0) = \{g_k(x,0) : k \in N, gcd(k, q-1) = 1\}$, $P(a) = \{g_k(x,a) : k \in N, gcd(k, q^2-1) = 1\}$ *for $a \neq 0$.*

Theorem 3 *$P(a)$ is closed under composition of polynomials if and only if $a = 0, 1$, or -1.*

Refer to [3] for the proof of Theorem 3.

We can purposely choose the integer a which is not $0, \pm 1$ so that there may not exist the inverse $g_D(x,a)$, which satisfies $g_D(g_E(x,a),a) = a$ by Theorem 3.

3 To strengthen the security of a hash function using permutation polynomials

The hash functions are defined as $h : V_\infty \rightarrow V^k$ or $f : V^{m+k} \rightarrow V^k$ recursively. As explained in Section 1, it is necessary for a primitive function to have a collision freeness so that the corresponding hash function can have a collision freeness. However, it is not easy to design the primitive function that has a collision freeness, since $|m+k| > |k|$. Almost all hash functions suggested up till now provide the collision freeness by complicated operations on fixed size blocks. A permutation polynomial is the mapping $f : c \rightarrow f(c)$. The mapping by this polynomial satisfies the collision freeness. If this polynomial has the property of one-wayness, it is suitable for a hash function. Figure 1 demonstrates this concept.

The following shows the conditions for hash functions and the enhancement of the security using a permutation polynomial.

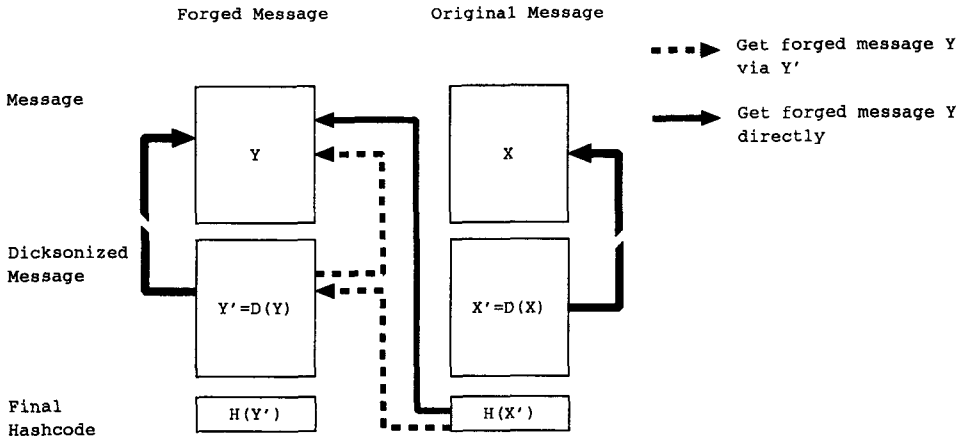

Figure 2: Attack the suggested hash function

- **Collision freeness**
  First of all, a forger should evaluate $D(M')$ which satisfies $D(M) \neq D(M')$ to find the message pair (M, M') which satisfies $M \neq M'$ and $h(D(M)) = h(D(M'))$. Even if he succeeds in finding the message pair of $(D(M), D(M'))$, he would have to compute $D^{-1}(M')$ to obtain M', which has the same hash code as M. This attack can be prevented by choosing a permutation polynomial $D(x)$ with the property of one-wayness.

  It may be possible to forge a message in another way, such as one in Figure 2. A forger may directly obtain the forged message M' without evaluating $D^{-1}(M')$, but this method of attack is prevented by a proper diffusion function. It is generally difficult for this kind of attack to succeed because a forger has to attack without prior knowledge of the transformation by a permutation polynomial, i.e. brute-forcely.

- **Strong one-wayness**
  Given a message M and $H = h(D(M))$, finding M' which satisfies $h(D(M')) = H$ requires finding $D(M')$ which satisfies $h(D(M')) = H$ and evaluating $D^{-1}(M')$. The property of impossibility of evaluating $D^{-1}(M')$ guarantees security against this way of attack.

- **Weak one-wayness**
  To compute M which satisfies $h(D(M)) = H$ from a hash code H, one must compute $D(M)$ and finally D^{-1}, which is impossible.

From the above observations, if it is impossible to compute D^{-1}, we can construct a good hash function. The Dickson polynomial in section 2 is a good candidate. If we don't select a of Dickson polynomial $g_k(M, a)$ as $0, \pm 1$, $P(a)$ is not closed under composition by Theorem 3. This means it is difficult to obtain the inverse function of the Dickson polynomial $g_k(M, a)$.

$g_7(0,2)$	0	$g_7(5,2)$	20	$g_7(10,2)$	15	$g_7(15,2)$	10	$g_7(20,2)$	5
$g_7(1,2)$	12	$g_7(6,2)$	7	$g_7(11,2)$	2	$g_7(16,2)$	22	$g_7(21,2)$	17
$g_7(2,2)$	16	$g_7(7,2)$	11	$g_7(12,2)$	6	$g_7(17,2)$	1	$g_7(22,2)$	21
$g_7(3,2)$	4	$g_7(8,2)$	24	$g_7(13,2)$	19	$g_7(18,2)$	14	$g_7(23,2)$	9
$g_7(4,2)$	8	$g_7(9,2)$	3	$g_7(14,2)$	23	$g_7(19,2)$	18	$g_7(24,2)$	13

Table 1: The values of Dickson polynomial when $k = 7, a = 2$

3.1 Permuting a message using Dickson polynomial

The following is the algorithm which permutes a message using Dickson polynomial.

1. Choose q, which is a prime number.
2. Represent a message M with a number from 0 to $q-1$. This can be done by splitting a message into many sub-blocks and then representing each sub-block with a number.
3. For q and Dickson polynomial $g_k(M,a)$, choose k which satisfies $(q^2-1,k) = 1$. This polynomial then permutes F_q, as shown in section 2.
4. Choose a of $g_k(M,a)$ so that it may be the function of the length of a message. a should not be $0, \pm 1$, which makes it impossible to obtain the inverse of the given Dickson polynomial by Theorem 3.
5. For chosen q, k, a, compute Dickson polynomial.

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

6. Publicize q, k, a for receiver to calculate the hash code.

The polynomial that results from after the above procedure is a permutation polynomial, which satisfies the property of one-wayness. The following is an example of permuting a message using a Dickson polynomial.

Let $q = 5^2 = 25$ and $a = 2$. Find k which satisfies $(k, q^2-1) = 1$ to make this polynomial permute F_q, where $q^2 - 1 = 25^2 - 1 = 625 - 1 = 624$

If we choose $k = 7$, then $(q^2-1, k) = (624, 7) = 1$.

$$g_k(x,a) = g_7(x,2) = \sum_{j=0}^{\lfloor 7/2 \rfloor} \frac{7}{7-j} \binom{7-j}{j} (-2)^j x^{7-2j} = x^7 + \frac{7}{6} \times 6 \times -2 \times x^5$$

$$+\frac{7}{5} \times \frac{5 \times 4}{2} \times 4 \times x^3 + \frac{7}{4} \times 4 \times (-2)^3 \times x = x^7 - 14x^5 + 56x^3 - 56x$$

In table 1 we can see that the above polynomial permutes F_q.

Given a message stream $x = 100, 3, 6, 21, 70$, this is mapped into $g_7(x,2) = 0, 4, 7, 17, 5$ by this Dickson polynomial.

3.2 How to evaluate the suggested polynomial efficiently

3.2.1 Some basic considerations

In this section, we'll explain the algorithm that can evaluate a polynomial and combinations efficiently . For computing a polynomial efficiently, we will use *Horner's rule.*

Rule 1 (Horner's rule)

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

In computing a polynomial $u(x)$, n multiplications and n additions are needed. If there are coefficients that are 0, then an equal number of additions are saved. Furthermore, this algorithm does not need to store the partial results.

Dickson polynomial needs the evaluation of combinations and can be efficiently evaluated as follows.

$$\begin{pmatrix} a-1 \\ b+1 \end{pmatrix} = \frac{(a-1)!}{(b+1)!\cdot(a-b-2)!} = \frac{(a-b)(a-b-1)}{a\cdot(b+1)} \cdot \begin{pmatrix} a \\ b \end{pmatrix} \quad (2)$$

Using equation(2), we can evaluate the coefficients of Dickson polynomial, while evaluating this polynomial using Horner's rule.

3.2.2 Evaluation of combinations and a polynomial

Using the previous section's observation, we suggest an algorithm which evaluates Dickson polynomial.

Evaluation of a polynomial: G is the coefficient of Dickson polynomial, and C is the buffer for the evaluation of combinations. In equation(2), a and b correspond to $k-j$, j each. Horner's rule is used and x is replaced with x^2. The last result is stored in D.

- **Step 1:** $C \leftarrow 1, D \leftarrow 1, ta \leftarrow 1, j \leftarrow \lfloor \frac{k}{2} \rfloor, x = M^2$
- **Step 2:**
$$C \leftarrow \frac{(k-2j+2)\cdot(k-2j+1)}{(k-j+1)\cdot j} \cdot C \bmod q, ta \leftarrow ta \cdot (-a)$$
- **Step 3:**
$$G \leftarrow \frac{k}{k-j} \cdot C \cdot ta \bmod q$$
- **Step 4:** $D \leftarrow (D \cdot x + G) \bmod q$
- **Step 5:** $j \leftarrow j - 1$
- **Step 6:** Repeat Step 2, 3, 4, 5 until $j = 0$

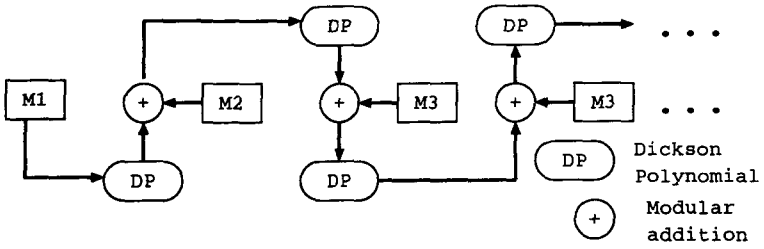

Figure 3: Message Digest Algorithm

4 Combining a Dickson polynomial and a message digest algorithm

If only the mapping using a permutation polynomial is used, we cannot get a fixed size hash code such as $h : V_\infty \rightarrow V^k$. Thus, we need a proper message digest algorithm to produce a fixed size hash code.

4.1 A proper message digest algorithm and its conditions

A forger can directly obtain $y \neq x$ which satisfies $H(x) = H(y)$ without computing D^{-1}. For example, if the message digest algorithm applied after mapping by a permutation polynomial is not sensitive in replacing the position of some message blocks, a forger can cause a collision easily by changing the position of some message blocks. However, this method of attack is prevented by a proper diffusion function. It is difficult for this kind of attack to succeed because the transformation of a message is related with a permutation polynomial. Still, there are a few conditions to be considered.

A message digest algorithm should satisfy the following conditions.

- The message digest algorithm must be computed efficiently, since the evaluation of a Dickson polynomial is needed.
- All bits of a hash code should be dependent on all bits of the input and its order.

If a message digest algorithm satisfies the above conditions, it is secure despite its simplicity.

We suggest the message digest algorithm which does modulo addition of the result of the Dickson polynomial evaluation of M_i and M_{i+1}. This algorithm is sensitive in replacing positions. It is also very fast. Figure 3 shows the suggested message digest algorithm.

4.2 Combining a Dickson polynomial and a message digest algorithm

In this section, we will combine the message digest algorithm of section 4.1 and the Dickson polynomial that satisfies the properties of one-wayness and impossibility of the composition. The message digest algorithm needs only to satisfy the conditions laid out

in section 4.1. Therefore, the combined message digest algorithm is composed of just one round.

The size of a message block is 128 bits, and the block is divided into four 32 bit sub-blocks. Four 32 bit buffer A,B,C,D are needed to store the partial results. All additions in this algorithm are defined as 2^{32} modulo addition. We use the following function to diffuse a message:

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

The entire message digest algorithm is described below.

- **Step 1: Appending the padding bits**
  The message is padded so that its bit length is congruent to 16 modulo 128 ($length \equiv 16$ *mod* 128). Padding is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 128. The padding consists of a single 1-bit followed by the necessary number of 0-bits.

- **Step 2: Appending length**
  A 64-bit representation of the length in bits of the original message (before the padding) is appended to the result of step 1. If the original length is greater than 2^{64}, then only the low-order 64 bits of the length are used. Thus, the field contains the length of the original message, modulo 2^{64}.

- **Step 3: Initializing buffers and Dickson polynomial's parameters**
  Four 32-bit registers(A,B,C,D) are initialized to the following hexadecimal values (low-order octets first):

  ```
  A = 01234567, B = 89ABCDEF, C = FEDCBA98, D = 76543210
  ```

  The parameters of Dickson polynomial $g_k(M, a)$ over the finite field F_q are q,k, and a , which are chosen in the following way:

  q is a randomly chosen prime number whose length is 32 bit. k is a 16 bit number which satisfies $gcd(k, q^2 - 1)$. The most significant bits of q and k are always "1". a is the lower 8 bits of the length field which is evaluated in step 2. If a is 0 or ± 1, do $a \leftarrow a + 2^4$.

- **Step 4: Appending another padding bits**
  To prevent the forgery of k and q chosen in step 3, k and q are appended to the message made in steps 1 and 2. Since a is dependent on the length of the message, and the length bits have already been appended, it is unnecessary to append it.

- **Step 5: Processing 128 bit message blocks**
  Update the contents of four buffers by the *diffusion function* defined previously. Add (modulo 2^{32}) the next message sub-block to the updated values of buffers. Evaluate Dickson polynomial for four 32 bit buffers and put them into four buffers. This is one round, and the result after one round is again put into Dickson polynomial. This procedure is iterated until it uses up all the message blocks. Figure 4 shows this process.

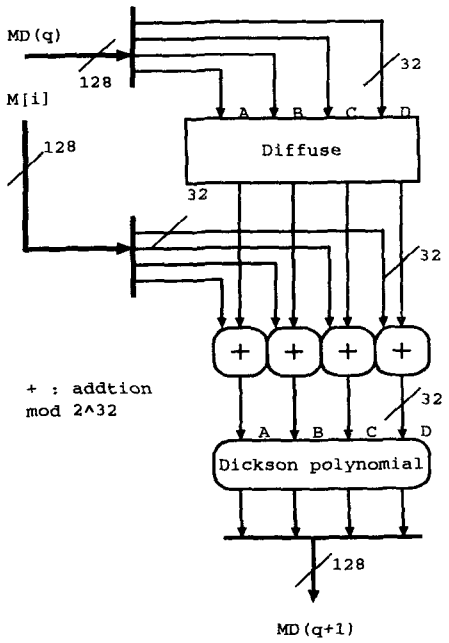

Figure 4: Processing 128 bit message blocks

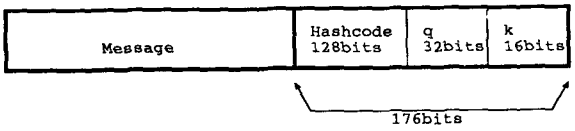

Figure 5: The final message form appended by the suggested hash code

- **Step 6: Appending the digital signature**
  The values chosen in step 3 should be revealed for a receiver to evaluate the hash code by padding them in the specific region of a message. The 128 bit value after step 6, 32 bit value q, and 16 bit value k are appended to the original message. Figure 5 shows the final form of a message.

5 How to attack the suggested hash function

5.1 Blind search of the colliding pair

If a forger tries to find the message which has the same hash code as the original one, he should evaluate the hash function for each message of m of q field elements. It takes mn operations to evaluate a message with m sub-blocks, where n is the degree of a polynomial. So $O(\binom{q}{m} \cdot m! \cdot mn)$ operations are needed. This, however, is computationally infeasible because it requires the following number of operations, even when the message size is 128

bits (the smallest one):

$$\binom{2^{32}}{4} \cdot 4 \cdot 4! \cdot 2^{18} = \frac{2^{32}(2^{32}-1)(2^{32}-2)(2^{32}-3)}{4 \cdot 3 \cdot 2 \cdot 1} \cdot 2^{24} \approx 2^{150}$$

5.2 Finding the other roots of $g_E(x,a) = y \ mod \ q$

If there are the other roots of $g_E(x,a) = y \ mod \ q$, a forger can attack the hash function by replacing partial results with the other roots and modifying the original message. However, because $g_E(x,a) = y \ mod \ q$ has a unique root $x \in GF(p^m)$ for any $b \in GF(p^m)$, this attack is impossible[5].

5.3 Finding the decryption key of $g_E(x,a) = y \ mod \ q$

It has been shown that $P(a)$, the set of Dickson polynomials $g_k(x,a)$, is not closed under the operation of composition in the case of $a \neq 0, \pm 1$. Thus, $g_D(x,a)$ which satisfies $g_D(g_E(x,a),a) = x$ does not exist.

5.4 Solving $g_E(x,a) = y \ mod \ q$

A forger may be able to obtain a forged message by colliding the hash code of a permuted message and then solving the equation for the colliding pair. D.G. Cantor and H. Zassenhaus suggested the algorithm which can solve the equation in $O(n^3 + n^2 \log q)$[4]. The suggested hash function uses the 16 bit number k and the 32 bit number q. It takes $O(\binom{32m}{2})$ operations to find the colliding pair of the message digest algorithm, where m is the number of sub-blocks of the message and it takes $O((2^{16})^3 + (2^{16})^2 \log 32)$ operations to find the root for the colliding pair. So a forger should do $O(2^{64m+14})$ operations to find a forged message from the given hash code. This attack is computationally infeasible.

5.5 What else?

To be secure against the birthday attack, we chose the hash code with the length of 128 bits. Because the suggested hash function uses the property of one-wayness, it is secure against a meet-in-the-middle attack.

6 Conclusion

We suggest the hash function through which security is guaranteed mathematically using Dickson polynomial. Dickson polynomial permutes the finite field when some conditions are satisfied. Furthermore, it has the property of one-wayness when we choose a special parameter for it. Using these two special properties, we suggest the hash function which satisfies a weak one-wayness, a strong one-wayness, and a collision freeness. We also observe various cryptanalysis and conclude that the suggested hash function is secure mathematically.

References

[1] S.G. Hwang, H.H. Cho, "Digital signature and hash function," *KIISC Review*, Vol. 2, No. 1, Mar, 1992.

[2] L.Carlitz and Jo Ann Lutz, "A Characterization of Permutation Polynomials over a Finite Field," *The American Mathematical Monthly*, Vol. 85, No. 9, Nov, 1978.

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

[4] D.G. Cantor and H. Zassenhaus, "A new algorithm for factoring polynomials over finite fields," *Mathematics of Computation*, Vol. 36, No. 154, Apr, 1981.

[5] Kenneth S. Williams, "Note on Dickson's Permutation polynomials," *Duke Mathematical Journal*, Vol. 38, No. 4, Dec, 1971.

[6] Donald E. Knuth, "Seminumerical algorithms, The art of computer programming," *Addison-wesley publishing company*, Vol. 2, 1969.

[7] William Stallings, "Network and internetwork security," *IEEE press*, 1995.

[8] 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.

[9] Selim G.Akl, "Digital Signatures: A Tutorial Survey," *IEEE Computer*, Vol. 16, No. 2, Feb, 1983.

[10] Ernest F. Brickell and Andrew M. Odlyzko, "Cryptanalysis: A survey of Recent Results," *IEEE Computer*, Vol. 21, No. 5, May, 1988.

[11] 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.

[12] D.R. Hayes, "A Geometric Approach to Permutation Polynomials over a Finite Field," *Duke Mathematical Journal*, Vol. 34, No. 2, Jun, 1967.