Polynomials have always occupied a prominent position in mathematics. In recent time their use has become unavoidable in cryptography.

**Part I:** Short excursus on various types of polynomials used in cryptography.

**Part II:** Comments on computing roots, and on evaluating polynomials over finite fields.
Part I

1. Nonlinear transformations over finite fields
2. Rabin and RSA transformations
3. Elliptic curves
4. Secret-sharing schemes
5. Transformations in AES
6. Deciphering in the McEliece scheme
7. Key distribution in consumer systems
8. Error-correcting-codes for bio-imprints
All functions from $GF(q)$ into $GF(q)$ are polynomials

A function $f(x)$ over $GF(2^m)$ is Almost Perfect Nonlinear (APN) if

- $f(x + a) + f(x) + b$ has at most two zeros in the field for every $a \neq 0$, and $b$
- $x \rightarrow f(x + a) + f(x)$ is 2 to 1 in $GF(2^m)$
Nonlinear transformations over finite fields

Until 2006, all known APN functions were monomials or binomials.

Examples:

\[
\begin{align*}
    f(x) &= x^3, \\     f(x) &= x^6 + x^5 \quad \in GF(2^7) \\
    f(x) &= x^{2^k+1} \quad x \in GF(2^m), (k, m) = 1, \quad \text{Gold} \\
    f(x) &= x^{2^{2k}-2^k+1} \quad x \in GF(2^m), (k, m) = 1, \quad \text{Kasami}
\end{align*}
\]
John Dillon (2006) introduced APN functions which were trinomials noting the existing relation between these functions and two-error correcting codes with parity-check matrix:

$$H = \begin{pmatrix}
1 & \alpha & \alpha^2 & \cdots & \alpha^j & \cdots & \alpha^{2m-2} \\
1 & f(1) & f(\alpha) & f(\alpha^2) & \cdots & f(\alpha^j) & \cdots & f(\alpha^{2m-2})
\end{pmatrix}$$

- $\alpha$ a primitive element in $GF(2^m)$
- $H$ parity-check matrix of a $(2m - 1, 2m - 1 - 2m, 5)$ code
- $R$ received vector
- $HR = S$ syndrome vector
System equations for finding the error positions $j$ and $h$

\[
\begin{align*}
\alpha^j + \alpha^h &= S_1 \\
 f(\alpha^j) + f(\alpha^h) &= S_2 \rightarrow f(\alpha^h + S_1) + f(\alpha^h) &= S_2
\end{align*}
\]

Unique solution $\leftrightarrow f(x)$ is an APN function
Nonlinear transformations over finite fields

Examples

\[ f(x) = x^3 \quad \text{on } GF(2^4) \text{ (BCH)} \]
\[ f(x) = x^3 + x^2 + x \quad \text{on } GF(2^4) \text{ (BCH code)} \]
\[ f(x) = x^5 + x^4 + x^3 + x^2 + x \quad \text{on } GF(2^7) \text{ (equiv. to a monomial)} \]
\[ f(x) = \alpha^7 x^{48} + \alpha x^9 + x^6 \quad \text{on } GF(2^6), \alpha^6 + \alpha + 1 = 0 \]

Recently, classes of polynomials with more than three terms have been found

\[ f(x) = b^{2^k} x^{2^k+s+2^k} + bx^{2^k+1} + cx^{2^k+1} + \sum_{i=1}^{k-1} r_i x^{2^i+k+2^i} \]
\[ x \in GF(2^{2k}) \]
\[ (s, 2k) = 1, \quad c \in GF(2^k), \quad b \in GF(2^{2k}), \quad r_i \in GF(2^k) \]
Rabin and RSA transformations

Operations in rings of residues modulo $M = pq$

- $e (= 2)$ divisor of $\phi(M)$

$$f(X) = X^e = a \mod M$$

- To invert the function $f(X)$ and to factor $M$ are equivalent problems

◊ $E$ prime with $\phi(M)$

$$f(x) = x^E = a \mod M$$

◊ Are $f(x)$ inversion and $M$ factorization equivalent problems?
**Power computation**

The computation of $X^m$ in any associative domain $D$ needs at most $2 \log_2 m$ products in $D$

$$m = m_0 + m_1 2 + m_2 2^2 + \cdots + m_s 2^s \quad m_i \in \{0, 1\}$$

$$X^m = X^{m_0 + m_1 2 + m_2 2^2 + \cdots + m_s 2^s} = X^{m_0} (X^2)^{m_1} (X^2)^{m_2} \cdots (X^2)^{m_s}$$

The minimum number of products is given by the minimum length $L$ of an *addition chain*

$$a_0, a_1, \ldots, a_L, \text{ with } a_0 = 1 \text{ and } a_j = a_i + a_t \quad i, t < j$$

Example: $m = 47$ \quad min chain length $< 2 \log_2 47 < 11.2$

1) 1, 2, 4, 8, 16, 32, 40, 44, 46, 47 \quad L = 9,

2) 1, 2, 4, 5, 10, 20, 40, 45, 47 \quad L = 8 \text{ minimum}
Elliptic curves

$E[\mathbb{F}_q]$ elliptic curve over a finite field $\mathbb{F}_q$

$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad a_i \in \mathbb{F}_q$

- $Q(x, y)$ point on $E[\mathbb{F}_q]$ $x, y \in E[\mathbb{F}_q]$
- $Q \rightarrow kQ = (k_0 + k_1 2 + k_2 2^2 + \ldots + k_s 2^s)Q$
- Point Doubling $\Rightarrow Q \rightarrow 2Q$
- Point Addition $\Rightarrow P, Q \rightarrow P + Q$
Elliptic curves

Sum and duplication of points

- \( P(x_1, y_1) \), \( Q(x_2, y_2) \) points on \( E[\mathbb{F}] \)
- Addition \( S = P + Q \), Doubling \( 2P = P + P \)

\[
m = \frac{y_2 - y_1}{x_2 - x_1}, \quad m = \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3}
\]

\[
x_3 = m^2 - a_1m - a_2 - x_1 - x_2
\]
\[
y_3 = -a_1x_3 - a_3 - y_1 - m(x_3 - x_1)
\]
A common secret $m$ is "shared" between any group of $k$ subjects out of $n$ subjects.

The secret $m$ is encrypted and $n$ private keys are generated as follows:

- A random polynomial of degree $k$ is selected

$$S(x) = x^k + a_1 x^{k-1} + \cdots + a_{k-1} x + m$$

- $x_i$ Public identifier of a subject
- $S(x_i) = y_i$ Private key for sharing
Secret-sharing

- Recovering polynomial $S(x)$ knowing the value of $k$ pairs $(x_i, y_i)$
  - $S(x)$ is rebuilt using the Lagrange interpolation
    \[
    L(x) = \prod_{i=1}^{k} (x - x_i)
    \]
    \[
    S(x) = \sum_{i=1}^{k} y_i \frac{L(x)}{x - x_i}
    \]

- The common secret $m$ is obtained as
  \[
  S(0) = \sum_{i=1}^{k} y_i \frac{L(0)}{0 - x_i} = (-1)^{k-1} \sum_{i=1}^{k} y_i \frac{\prod_{j=1}^{k} x_j}{x_i}
  \]
The Sub-byte transformation is applied to all rows of the data matrix

- Polynomials over $GF(2^8)$:
- Data matrix row $X_i(x) = X_{i0} + X_{i1}x + X_{i2}x^2 + X_{i3}x^3$
- Encryption polynomial $a(x) = a_0 + a_1x + a_2x^2 + a_3x^3$
- Encrypted row $X_i(x) \Rightarrow X_i(x)a(x) \mod (x^4 - 1)$
Deciphering in the McEliece scheme

**Public key:** binary $n \times k$ matrix $G = PGB$
- $P$ $n \times n$ secret permutation matrix
- $B$ $k \times k$ binary nonsingular secret matrix
- $G$ binary $n \times k$ secret generator matrix of a cyclic or Goppa $(n, k, 2t + 1)$ code over $GF(2)$
- $\alpha$ primitive element of $GF(2^m)$, $n = 2^m - 1$

**Enciphering:** information vector $x$
- error vector with $t$ errors $e$
- encrypted message $r = Gx + e$
Deciphering in the McEliece scheme

Deciphering ⇒ decoding the vector \( \mathbf{r} \), i.e. correction of \( t \) errors:

- Computation of \( \mathbf{R} = \mathbf{P}^{-1} \mathbf{r} \), the modified received vector
- Computation of \( 2t \) syndromes
- Computation of the error locator polynomial \( \sigma(z) \) (Berlekamp-Massey)
- Error location: evaluation of \( \sigma(z) \) in \( n \) points.
Deciphering in the McEliece scheme

- $\mathbf{R} = (R_1, R_2, \ldots, R_n)$ modified received vector
- $R(x) = \sum_{i=0}^{n-1} R_i x^i$ polynomial of degree $n - 1$
- Computation of $2t$ syndromes $S_i = R(\alpha^i)$, $i = 1, \ldots, 2t$
- Construction of $\sigma(z)$ of degree $t$

Vandermonde $\rightarrow$ GPZ $\rightarrow$ Berlekamp-Massey

- Evaluation of $\sigma(z)$ in $n$ points $\alpha^j \in GF(2^m)$ (Chien search): an error is in position $j$ if

$$\sigma(\alpha^j) = 0$$
Key distribution in consumer systems

Parameters:

- $m$ common access key
- $N$ number of users
- $k_u$ private key of user $u$

Broadcast hash function $h(x)$, and polynomial

$$P(x) = \prod_{u=1}^{N} (x - h(k_u)) + m = \sum_{i=0}^{N} P_i x^i$$

User $u$ actions:

- $h(k_u)$ evaluation
- $m = P(h(k_u))$ evaluation of $P(x)$ to get the key $m$
Error-correcting codes for bio-imprints

- To store or distribute bio-imprints keeping the original imprint secret, i.e. it should be difficult to recover the original sample imprint from its stored version.
- Automatically recognizing a claimed identity, which requires fast checking of whether the imprint taken is among a stored set of encrypted sample imprints, given that the imprint taken is corrupted by sensor errors.
Error-correcting codes for bio-imprints

The model

- \( \mathbf{x} \) sample bio-imprint encoded as a binary stream of \( k \) bits
- \( \mathbf{C} \) code word of an \((n, k)\) \( t \)-error correcting code in \( GF(q) \)
- \( t \) has the meaning of a threshold
- \( z = \mathbf{C} + (x, 0) \) encrypted bio-imprint

Checking a bio-imprint is a kind of incomplete decoding of the \((n, k)\)-code with \( n \) very large
Check:

- $y$ $k$-dimensional vector encoding the bio-imprint taken

$$d = (y, 0) \rightarrow R = z + d = e + C$$

- $C$ code word corrupted by $\ell$ errors, i.e. vector $(x - y)$
  - the number of errors $\ell$ is computed and compared with $t$
    if $\ell < t$ test passed, if $\ell > t$ test not passed
  - Operatively $\sigma(z)$ is computed and it is checked whether all
    roots are in $\text{GF}(q)$, i.e.

$$\gcd(\sigma(z), z^q - 1) = \sigma(z)$$

- The most expensive task is the computation of the
  syndromes, and sub-orderly the computation of $\sigma(z)$ via
  Berlekamp-Massey algorithm.
Part II

- Computation of the roots of polynomials in their full splitting finite field. Application to decoding cyclic and Goppa codes.
- Evaluation of polynomials over finite fields: a fast algorithm that admits of asymptotic upper bounds to the number of products and sums respectively equal to

\[ c\sqrt{n}, \quad c' \frac{n}{\log n} \]
Roots of Polynomials over $GF(q)$

Two steps:

- Computation of the roots of $\sigma(x)$, defined over $GF(q)$ and full split in $GF(q^m)$ by means of the Cantor-Zassenhaus algorithm. The roots $\beta$ are expressed in a polynomial basis of $GF(q^m)$
- Computation of the exponential representation $\beta = \alpha^j$, given $\alpha$, primitive in $GF(q^m)$, by means of Shanks’ algorithm.

The usual method applied in the decoders requires the evaluation of $\sigma(x)$ in $q^m$ points, thus has complexity

$$q^m \times \text{ complexity of } \sigma(\alpha^i) \text{ evaluation}$$

to perform both tasks.
Cantor-Zassenhaus’ Algorithm in characteristic 2

- $\sigma(x)$ polynomial of degree $t$ in $\mathbb{F}_{2^m}$
- $L = \frac{2^{2m}-1}{3}$ random in $\mathbb{F}_{2^{2m}}$
- $\zeta$ primitive cubic root of unity in $\mathbb{F}_{2^{2m}}$
- Compute $a(x) = (x + \omega)^L \mod \sigma(x)$

1 If $a(x) \neq 1, \zeta, \zeta^2$ then $\sigma(x)$ has a common factor with at least one of the following polynomials

$$a(x), \ a(x) - 1, \ a(x) - \zeta, \ a(x) - \zeta^2,$$

with probability greater than $\frac{8}{9}$.

2 All roots are obtained with at most $t$ repetitions.

The largest computational cost is given by the computation of $a(x)$ which entails computing powers of polynomial modulo another polynomial in finite fields.
Shanks’ algorithm for discrete logarithm

Shank’s algorithm:

- The exponent $\ell$ in the equality
  \[ \alpha^\ell = b_0 + b_1 \alpha + \cdots + b_{m-1} \alpha^{m-1} \]
  is written in the form $\ell = \ell_0 + \ell_1 \lceil \sqrt{n} \rceil$.
- A table $\mathcal{T}$ is constructed with $\lceil \sqrt{n} \rceil$ entries $\alpha^{\ell_1 \lceil \sqrt{n} \rceil}$,
- then a cycle of length $\lceil \sqrt{n} \rceil$ is started computing
  \[ A_j = (b_0 + b_1 \alpha + \cdots + b_{m-1} \alpha^{m-1})\alpha^{-j} \quad j = 0, \ldots, \lceil \sqrt{n} \rceil - 1 \]
  and looking for $A_j$ in the Table;
- when a match is found with the $\kappa$-th entry, we set $\ell_0 = j$
  and $\ell_1 = \kappa$, and the discrete logarithm $\ell$ is obtained as
  $j + \kappa \lceil \sqrt{n} \rceil$.
- This algorithm can be performed with complexity $O(\sqrt{n})$.

In our scenario, since we need to compute $t$ roots, the complexity is $O(t\sqrt{n})$. 
Evaluation of a polynomial in the point $\alpha$

$$p(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_m x^m$$

The direct evaluation needs

- Computation of $m$ powers $\alpha^i$
- Computation of $m$ products $p_i \alpha^i$
- Computation of $m$ sums
- Total $2m - 1$ products and $m$ sums

Horner’s Rule

$$p(x) = p_0 + x(p_1 + x(p_2 + \cdots + x(p_{m-1} + xp_m)\cdots))$$

needs $m$ products and $m$ sums

This rule is universal, i.e., it holds in every field (associative ring), and is optimal if the field has an infinite number of elements.
Evaluation of a polynomial in the point $\alpha$

In finite fields it is possible to do better

- The exemplification is restricted to $GF(2)$ and extensions
- Three different problems:
  1. To evaluate a polynomial in a single point
  2. To evaluate a polynomial in $s$ distinct points
  3. To evaluate $f$ polynomials in the same point
Evaluation of a polynomial in the point $\alpha$

Evaluation of $p(x)$ over $GF(2)$ in a single point $\alpha$ in $GF(2^m)$

$$p(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_n x^n \quad p_i \in GF(2) \quad \ell = \left\lfloor \frac{n}{2} \right\rfloor$$

$$p(x) = p_0 + p_2 x^2 + \cdots + p_{2\ell} x^{2\ell} + x(p_1 + p_3 x^2 + p_5 x^4 + \cdots + p_{2\ell+1} x^{2\ell})$$

$$p(x) = p_{00}(x)^2 + xp_{01}(x)^2 \Rightarrow p(\alpha) = p_{00}(\alpha)^2 + xp_{01}(\alpha)^2$$

Evaluation of $p(\alpha)$ requires

1. The evaluation of $p_{00}(\alpha)$ and $p_{01}(\alpha)$ of degree $n/2$
2. The computation of 2 squares
3. The computation of 1 product $\alpha p_{01}(\alpha)^2$
4. The computation of 1 sum
The evaluation of $p_{00}(a)$ and $p_{01}(\alpha)$ of degree $n/2$ can be done with

- $n/2$ multiplications
- $n - 1$ additions

The total number of operations for obtaining $p(\alpha)$ is

- $3 + n/2$ multiplications
- $n$ additions

The procedure can be re-applied iteratively to every $p_{ij}(\alpha)$ and their descendants
At each iteration the number of polynomials is doubled and their degrees are halved

<table>
<thead>
<tr>
<th>st</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th># des</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p(x)</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>p_0^0(x)</td>
<td></td>
<td>p_1^0(x)</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>p_0^1(x)</td>
<td>p_1^1(x)</td>
<td>p_2^1(x)</td>
<td>p_3^1(x)</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>p_0^L(x)</td>
<td>p_1^L(x)</td>
<td>p_2^L(x)</td>
<td>...</td>
<td>p_s^L(x)</td>
<td>2^L</td>
<td></td>
</tr>
</tbody>
</table>

The reconstruction starts from the bottom level (L) and ends with p(\alpha) after L steps

Notational remark: p_j^i(x) = p_{ij}(x)
Evaluation of a polynomial in the point $\alpha$

Computational complexity

- After $L$ steps we have $2^L$ polynomials of degree $\left\lfloor \frac{n}{2^L} \right\rfloor$
- Number of operations
  1. $\left\lfloor \frac{n}{2^L} \right\rfloor$ powers of $\alpha$
  2. $n$ additions for producing $2^L$ polynomials $p_{Lj}(\alpha)$
  3. $2^{L+1} - 2 = 2^L + \cdots + 2$ squares of the polynomials $p_{ij}(\alpha)$
  4. $2^L - 1 = 2^{L-1} + \cdots + 1$ additions for reconstructing $p(\alpha)$
  5. $2^L - 1 = 2^{L-1} + \cdots + 1$ products for reconstructing $p(\alpha)$

- Total number of arithmetic operations
  1. $3 \cdot 2^L - 3 + \left\lfloor \frac{n}{2^L} \right\rfloor$ products
  2. $n + 2^L - 1$ additions
Evaluation of a polynomial in the point $\alpha$

Optimal value of $L$

$$3 \cdot 2^L \approx \frac{n}{2^L}$$

$$2^L \approx \sqrt{\frac{n}{3}}$$

The total number of products is approximately $2\sqrt{3n}$

The total number of sums can be reduced to about

$$\frac{n}{\ln(n)}$$

re-utilizing sums in the evaluations of $2^L$ polynomials at level $L$
Evaluation of a polynomial in the point $\alpha$

**Polynomial with coefficients in $GF(2^s)$**

The computation is reduced to the evaluation of $s$ polynomials with coefficients in $GF(2)$

$$p(x) = p_0(x) + \alpha p_1(x) + \alpha^2 p_2(x) + \cdots + \alpha^s p_s(x)$$

Typical cases $n = 2^m$ or $2^m - 1$

Asymptotic number of multiplications

$$O(\sqrt{n \ln(n)})$$
Open Problems

1. Find an upper bound to the multiplicative complexity necessary to evaluate a polynomial of degree $n$ over finite fields (over infinite fields Horner’s rule is optimal, according to Borodin and Munro)

2. Can Berlekamp-Massey algorithm be improved when both $t$ and $n$ are large? (the complexity is $t^2 \log(t)$ according to von zur Gathen)
Open Problems

1. Find the minimum number of additions necessary to evaluate a polynomial of degree $n$ over finite fields (over infinite fields the Horner’s rule is optimal, according to Borodin and Munro)

2. Find the constant $c(p)$ such that $c(p)\frac{n}{\ln(n)}$ is a tight upper bound to the additive complexity for evaluating a polynomial of degree $n$ over finite fields of characteristic $p$. 
References


References


