Cryptology complementary

✧

Finite fields — the practical side (1)

Pierre Karpman
pierre.karpman@univ-grenoble-alpes.fr
https://www-ljk.imag.fr/membres/Pierre.Karpman/tea.html

2018–03–15

Bits as field elements

- Digital processing of information $\rightsquigarrow$ dealing with bits
- Error-correcting codes, crypto $\rightsquigarrow$ need analysis $\rightsquigarrow$ maths
- $\Rightarrow$ bits (no structure) $\mapsto$ field elements (math object)

- "Natural" match: $\{0, 1\} \cong \mathbb{F}_2 \equiv \mathbb{Z}/2\mathbb{Z} \equiv$ "(natural) integers modulo 2"
- $\mathbb{F}_2$: two elements (0, 1), two operations ($+$, $\times$)

What's $\mathbb{F}_2$ like?

- Addition $\equiv$ exclusive or (XOR ($\oplus$))
- Multiplication $\equiv$ logical and ($\wedge$)
- $\Rightarrow$ "Boolean" arithmetic

- Fact: any Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be computed using only $\oplus$ and $\wedge$
- Fact 2: ditto, $g : \{0,1\}^n \to \{0,1\}^m$
- Fact 3: ditto, using NAND ($\neg \circ \wedge$)

One bit is nice, but...

- We rather need bit strings $\{0,1\}^n$ than single bits
- Now two "natural" matches:
- $\mathbb{F}_2^n$ (vectors over $\mathbb{F}_2$)
  - Can add two vectors
  - Cannot multiply "internally" (but there's a dot/scalar product)
- $\mathbb{Z}/2^n\mathbb{Z}$ (natural integers modulo 2^n)
  - Can add, multiply
  - Not all elements are invertible (e.g. 2) $\Rightarrow$ only a ring

A third way

- Also possible: $\mathbb{F}_{2^n}$: an *extension* field
  - Addition "like in $\mathbb{F}_2^n$"
  - Plus an internal multiplication
  - All elements (except zero) are invertible
- (Just in a moment)

Why are these useful?

- Allows to perform operations on data
  - E.g. adding two messages together
- Vector spaces ⇒ linear algebra (matrices)
  - Powerful tools to solve "easy" problems
  - (Intuition: crypto shouldn't be linear)
- Fields ⇒ polynomials
  - With one or more variable
  - ⇒ Can compute differentials
- Can mix $\mathbb{F}_2^n$, $\mathbb{Z}/2^n\mathbb{Z}$ to make things "hard"
  - Popular "ARX" strategy in symmetric cryptography (FEAL/MD5/SHA-1/Chacha/Speck/...)

How to build finite fields? Easy case: prime fields

- Just take the integers and reduce modulo N
  - Operations work "as usual"
  - Over a finite set
- Problem: have to ensure invertibility of all elements
  - Necessary condition N has to be prime
  - (Otherwise, $N = pq \Rightarrow p \times q = 0 \mod N \Rightarrow$ neither is invertible)
  - In fact also sufficient: $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ is a field for p prime

Fields ⇒ polynomials

- One can define the polynomials $\mathbb{F}_p[X]$ over a finite field
- One can divide polynomials (e.g. $(X^2 + X)/(X + 1) = X$)
- ⇒ notion of remainder (e.g. $(X^2 + X + 1)/(X + 1) = (X, 1)$
- ⇒ can define multiplication in $\mathbb{F}_p[X]$ modulo a polynomial Q
    - If $\deg(Q) = n$, operands are restricted to a finite set of poly. of $\deg < n$

Finite fields with polynomials

- $\mathbb{F}_p[X]/Q$ is a finite set of polynomials
- With addition, multiplication working as usual (again)
- To make it a field: have to ensure invertibility of all elements
  - Necessary condition: Q is *irreducible*, i.e. has no non-constant factors (Q is "prime")
  - In fact also sufficient: $\mathbb{F}_p[X]/Q$ is a field for Q irreducible *over* $\mathbb{F}_p$
  - Claim: irreducible polynomials of all degrees exist over any given finite field

Quick questions

- How many elements does have a field built as $\mathbb{F}_p[X]/Q$, when $\deg(Q) = n$?
- Describe the cardinality of finite fields that you know how to build
- Let $\alpha \in \mathbb{F}_q = \mathbb{F}_p[X]/Q$. what is the result of $\alpha + \alpha + \ldots + \alpha$ ($p - 1$ additions)?

Quick remarks

- Two finite fields of equal cardinality are *unique up to isomorphism*
- But different choices for Q may be possible $\Rightarrow$ different *representations*
- One can build extension towers: extensions over fields that were already extension fields, iterating the same process as for a single extension

How to implement finite field operations?

- $\mathbb{F}_p$:
  - Addition: add modulo
  - Multiplication: multiply modulo
  - Inverse: use the extended Euclid algorithm or Little Fermat Theorem
- $\mathbb{F}_{p^d}$:
  - Addition: add modulo, coefficient-wise
  - Multiplication: multiply polynomials modulo (w.r.t. polynomial division) $\Rightarrow$ Use LFSRs
  - Inverse: use the extended Euclid algorithm or Lagrange Theorem

So what are LFSRs?

LFSR = *Linear Feedback Shift Register*

LFSR (type 1)

An LFSR of length n over a field $\mathbb{K}$ is a map
$\mathcal{L} : [s_{n-1}, s_{n-2}, \ldots, s_0] \mapsto$
$[s_{n-2} - s_{n-1}r_{n-1}, s_{n-3} - s_{n-1}r_{n-2}, \ldots, s_0 - s_{n-1}r_1, -s_{n-1}r_0]$ where the $s_i,\ r_i \in \mathbb{K}$

LFSR (type 2)

An LFSR of length n over a field $\mathbb{K}$ is a map
$\mathcal{L} : [s_{n-1}, s_{n-2}, \ldots, s_0] \mapsto$
$[s_{n-2}, s_{n-3}, \ldots, s_0, s_{n-1}r_{n-1} + s_{n-2}r_{n-2} + \ldots + s_0 r_0]$ where the $s_i,\ r_i$ $\in \mathbb{K}$

Theorem: The two above definitions are “equivalent”

Characterization

An LFSR is fully determined by:

- Its base field $\mathbb{K}$
- Its state/register size n
- Its feedback function $(r_{n-1}, r_{n-2}, \ldots, r_0)$

An LFSR may be used to generate an infinite sequence (U_m) (valued in $\mathbb{K}$):

1. Choose an initial state $S = [s_{n-1}, \ldots, s_0]$
2. $U_0 = S[n-1] = s_{n-1}$
3. $U_1 = \mathcal{L}(S)[n-1]$
4. $U_2 = \mathcal{L}^2(S)[n-1]$, etc.

Some properties

- The sequence generated by an LFSR is periodic (Q: Why?)
- Some LFSRs map non-zero initial states to the zero one (Q: Give an example?)
- Some LFSRs generate a sequence of maximal period (Q: What is it?)
- It is very easy to recover the feedback function of an LFSR from (enough outputs of) its generated sequence (Q: How?)

A simple case: binary LFSRs

We will in fact mostly care about:

- LFSRs of type 1
- Over $\mathbb{F}_2$

$\mathcal{L}$ becomes:

1. Shift bits to the left
2. If the (previous) msb was 1
    1. Add (XOR) 1 to some state positions (given by the feedback function)

Some formalism

The feedback function of an LFSR can be written as a polynomial:

- $(r_{n-1}, r_{n-2}, \ldots, r_0) \equiv X^n + r_{n-1}X^{n-1} + \ldots + r_1 X + r_0$
- $\mathcal{L}$ corresponds to the multiplication by $X \mod$ the feedback polynomial

Example:

- Take $\mathcal{L}$ of length 4 over $\mathbb{F}_2$ and feedback polynomial $X^4 + X + 1$
- $\Rightarrow \mathcal{L} : (s_3, s_2, s_1, s_0) \mapsto (s_2, s_1, s_0 \oplus s_3, s_3)$

Back to multiplication: the $\mathbb{F}_{2^n}$ case

- $\alpha \in \mathbb{F}_{2^n}$ is "a polynomial of deg $< n$"
- So $\alpha = \alpha_{n-1}X^{n-1} + \ldots + \alpha_1 X + \alpha_0$
- So we can multiply α by $X \Rightarrow \alpha_{n-1}X^n + \ldots + \alpha_1 X^2 + \alpha_0 X$
- But this may be of deg $= n$, so not in $\mathbb{F}_{2^n}$
- So we reduce the result mod $Q = q_n X^n + q_{n-1}X^{n-1} + \ldots + q_1 X + q_0$, the defining polynomial of $\mathbb{F}_{2^n} = \mathbb{F}_2[X]/Q$
- This can be implemented with an LFSR!

Reduction illustrated: two cases

Case 1: $\deg(\alpha X) < n$

- There's nothing to do

Case 2: $\deg(\alpha X) = n$

- Then $\deg(\alpha X - Q) < n$
- And $\alpha X - Q$ is precisely the remainder of $\alpha X \div Q$
- (Think how $2N > a > N \mod N = a - N$)

Summary

- An element of $\mathbb{F}_{2^n} = \mathbb{F}_2[X]/Q$ is a polynomial
- ...is a state of an LFSR with feedback polynomial Q
- Multiplication by X is done $\mod Q$
- ...is the result of clocking the LFSR once
- Multiplication by X^2 is done by clocking the LFSR twice, etc.
- Multiplication by $\beta_{n-1}X^{n-1} + \ldots + \beta_1 X + \beta_0$ is done "the obvious way"

A note on representation

It is convenient to write $\alpha = \alpha_{n-1}X^{n-1} + \ldots + \alpha_1 X + \alpha_0$ as the integer $a = \alpha_{n-1}2^{n-1} + \ldots + \alpha_1 2 + \alpha_0$

- Example: $X^4 + X^3 + X + 1$ "=" $27 =$ `0x1B`

Examples in $\mathbb{F}_{2^8} \equiv \mathbb{F}_2[X]/X^8 + X^4 + X^3 + X + 1$

Example 1:

- $\alpha = X^5 + X^3 + X$ (0x2A), $\beta = X^2 + 1$ (0x05)
- $\alpha + \beta = X^5 + X^3 + X^2 + X + 1$ (0x2F)
- $\alpha\beta = X^2\alpha + \alpha = X^7 + X^5 + X^3$ (0xA8) $+ X^5 + X^3 + X = X^7 + X$ (0x82)

Examples in $\mathbb{F}_{2^8} \equiv \mathbb{F}_2[X]/X^8 + X^4 + X^3 + X + 1$

Example 2:

- $\alpha = X^5 + X^3 + X$, $\gamma = X^4 + X$ ($\texttt{0x12}$)
- $\alpha\gamma = X^4\alpha + X\alpha$
  - $X^4\alpha = X(X(X^7 + X^5 + X^3))$
  - $X(X^7 + X^5 + X^3) = (X^8 + X^6 + X^4) + (X^8 + X^4 + X^3 + X + 1) = X^6 + X^3 + X + 1$
  - $X(X^6 + X^3 + X + 1) = X^7 + X^4 + X^2 + X$
- $= X^7 + X^4 + X^2 + X$ ($\texttt{0x96}$) $+ X^6 + X^4 + X^2$ ($\texttt{0x54}$) $= X^7 + X^6 + X$ ($\texttt{0xC2}$)

Exercise

1. Implement (in C) the multiplication by X in $\mathbb{F}_{2^8} \equiv \mathbb{F}_2[X]/X^8 + X^4 + X^3 + X + 1$, using a byte (type `uint8_t`) to represent field elements
2. Using the previous function, implement the multiplication of two arbitrary elements