Introduction to cryptology (GBIN8U16)

✧

Extension fields, Hash functions

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

2018–02–14

Finite fields: prime fields recap

- Motivation: a rich field structure over a finite set
- Idea: 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 (see that another day)
- $\mathbb{F}_{p^d}$:
  - 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 (probably won't see that)

Multiplication in $\mathbb{F}_{2^n}$

- $\alpha \in \mathbb{F}_{2^n}$ 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 $\mathbb{F}_{2^n} = \mathbb{F}_2[X]/Q$

Reduction: 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 $2N > a > N \mod N = a - N$)

Multiplication + reduction: alternative view

$(\alpha_{n-1}, \ldots, \alpha_1, \alpha_0) \times X \mod (Q_n, Q_{n-1}, \ldots, Q_1, Q_0) =$

- $(\alpha_{n-2}, \ldots, \alpha_1, \alpha_0, 0)$ if $\alpha_{n-1} = 0$
- $(\alpha_{n-2} - Q_{n-1}, \ldots, \alpha_1 - Q_2, \alpha_0 - Q_1, -Q_0)$ if $\alpha_{n-1} = 1$
- (or $(\alpha_{n-2} + Q_{n-1}, \ldots, \alpha_1 + Q_2, \alpha_0 + Q_1, Q_0)$ as we're in characteristic two)
- or $(\alpha_{n-2} + Q_{n-1}\alpha_{n-1}, \ldots, \alpha_1 + Q_2\alpha_{n-1}, \alpha_0 + Q_1\alpha_{n-1}, Q_0\alpha_{n-1})$ $\Rightarrow$ the result of one step of LFSR with feedback polynomial equal to $(-)Q$!

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:

- $\alpha = X^5 + X^3 + X$, $\gamma = X^4 + X$ (0x12)
- $\alpha\gamma = X^4\alpha + X\alpha$
  - $X^4\alpha = X(X(X^7 + X^5 + X^3))$
  - $X(X^7 + X^5 + X^3) = (X^8 + X^6 + X^4) + (X^8 + X^4 + X^3 + X + 1) = X^6 + X^3 + X + 1$
  - $X(X^6 + X^3 + X + 1) = X^7 + X^4 + X^2 + X$
- $= X^7 + X^4 + X^2 + X$ (0x96) $+ X^6 + X^4 + X^2$ (0x54) $= X^7 + X^6 + X$ (0xB2)

Why do we care again?

Extension fields (esp. over $\mathbb{F}_2$) are useful to:

- Build polynomial MACs
- Define matrices "over bytes" or *nibbles* (4-bit values)
  - Used e.g. in the AES
- Etc.

They're generally useful when working over binary data

And now for something completely different

Hash function

A hash function is a mapping $\mathcal{H} : \mathcal{M} \to \mathcal{D}$

So it really is just a function...

Usually:

- $\mathcal{M} = \bigcup_{\ell < N} \{0,1\}^{\ell}$, $\mathcal{D} = \{0,1\}^{n}$, $N \gg n$
- N is typically $\geq 2^{64}$, $n \in \{\cancel{128}, \cancel{160}, \cancel{224}, 256, 384, 512\}$

Also popular now: extendable-output functions (XOFs): $\mathcal{D} = \bigcup_{\ell < N'} \{0,1\}^{\ell}$

- Hash functions are *keyless*
- So, how do you tell if one's **good**?

Three classical security properties

1. **First preimage**: given t, find m s.t. $\mathcal{H}(m) = t$
2. **Second preimage**: given m, find $m' \neq m$ s.t. $\mathcal{H}(m) = \mathcal{H}(m')$
3. **Collision**: find $(m, m' \neq m)$ s.t. $\mathcal{H}(m) = \mathcal{H}(m')$

Generic complexity:
1), 2): $\Theta(2^n)$;
3): $\Theta(2^{n/2}) \leftsquigarrow$ "Birthday paradox"

(There's actually more...)

Why do we care? Applications!

Hash functions are useful for:

- Hash-and-sign (RSA signatures, (EC)DSA, ...)
- building MACs (HMAC, ...)
- Password hashing (with a grain of salt)
- Hash-based signatures (inefficient but PQ)
- In padding schemes (OAEP, ...)
- Etc.

$\Rightarrow$ A versatile building block, but only a building block

So, how do you build hash functions?

- Objective #1: be secure
- Objective #2: be efficient
  - Even more than block ciphers!
  - $\Rightarrow$ work with limited amount of memory

So...

- (#2) Build $\mathcal{H}$ from a *small component*
- (#1) Prove that this is okay

$\Rightarrow$ Kind of like a mode of operation!

What kind of small component?

Compression function

A compression function is a mapping $\mathfrak{f} : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$

- A family of functions from n to n bits
- Not unlike a block cipher, only not invertible

Permutation

A permutation is an invertible mapping $\mathfrak{p} : \{0,1\}^n \to \{0,1\}^n$

Yes, very simple

- Like a block cipher with a fixed key, e.g. $\mathfrak{p} = \mathcal{E}(0, \cdot)$

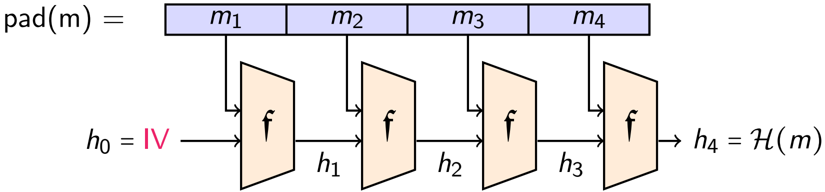
From small to big (compression function case)

Assume a good $\mathfrak{f}$

- Main problem: fixed-size domain $\{0,1\}^n \times \{0,1\}^b$
- Objective: *domain extension* to $\bigcup_{\ell<N} \{0,1\}^\ell$

The classical answer: the Merkle-Damgård construction (1989)

MD: with a picture



That is: $\mathcal{H}(m_1 \| m_2 \| m_3 \| \ldots) = \mathfrak{f}(\ldots \mathfrak{f}(\mathfrak{f}(\mathfrak{f}(\mathrm{IV}, m_1), m_2), m_3), \ldots)$

$\mathrm{pad}(m) \approx m \| 1000 \ldots 00 \langle \text{length of } m \rangle$

MD: does it work?

Efficiency?

- Only sequential calls to $\mathfrak{f}$
- $\Rightarrow$ fine

Security?

- Still to be shown
- Objective: *reduce* security of $\mathcal{H}$ to that of $\mathfrak{f}$
  - "If $\mathfrak{f}$ is good, then $\mathcal{H}$ is good"
- True for collision and first preimage, **false** for second preimage
- Won't see the details, though (in the end, everything is quite fine)

So how to do $\mathfrak{f}$?

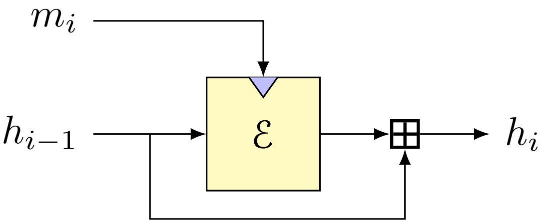
1. Start like a block cipher
2. Add *feedforward* to prevent invertibility

Examples:
"Davies-Meyer": $\mathfrak{f}(h, m) = \mathcal{E}_m(h) \boxplus h$
"Matyas-Meyer-Oseas": $\mathfrak{f}(h, m) = \mathcal{E}_h(m) \boxplus m$

- Systematic analysis by Preneel, Govaerts and Vandewalle (1993). "PGV" constructions
- Then rigorous proofs (in the ideal cipher model) (Black et al., 2002), (Black et al., 2010)

Re: Davies-Meyer

Picture:



Used in MD4/5 SHA-0/1/2, etc.

Re: Re: Davies-Meyer

Why is the “message” the “key”?

- Disconnect chaining value and message length!
- $\mathcal{E}$’s block length: fixed by security level
- $\mathcal{E}$’s key length: fixed by “message” length
- Large “key” $\Rightarrow$ more efficient
- Example: MD5’s “block cipher”: 128-bit blocks, 512-bit keys

DM incentive: use very simple *message expansion* (“key schedules”)

- To be efficient!
- Warning: can be a source of weakness

Major PGV Warning

PGV constructions are proved secure in the *ideal cipher model*,
BUT

- Real ciphers are not ideal!
- Real ciphers *don't have to be ideal* to be okay ciphers
  - IDEA (Lai and Massey, 1991): weak key classes (Daemen et al., 1993)
  - TEA (Wheeler and Needham, 1994): equivalent keys (Kelsey et al., 1996)
- But using them in DM mode is a very bad idea (Steil, 2005), (Wei et al., 2012)!

Symmetric summary

We have seen:

- Block ciphers
- Mode of operations
- MACs
- Hash functions
- Some security properties

All involve similar/distinct techniques

Holidays light personal work:

- Can you define the above objects?
- Do you know how to build some (roughly)?
- Define a good mode of operation (for confidentiality)
- ... and a bad one

What comes next?

After the holidays:

- Some recap
- The beginning of public key primitives