Introduction to cryptology (GBIN8U16)

✧

Message Authentication Codes, LFSRs

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

2018–02–07

Authentification (in crypto)

Crypto is not all about encrypting. One may also want to:

- Get access to a building/car/spaceship
- Electronically sign a contract/software/Git repository
- Detect tampering on a message
- Detect “identity theft”
- Etc.

$\Rightarrow$ domain of digital signatures and/or message authentication codes (MACs)

A major rule

In the case of a symmetric channel (e.g. on a network):

- It may be fine to only authenticate
- It is *never okay* to only encrypt
  - Recommended reading: *Attacking the IPsec Standards in Encryption-only Configurations* (Degabriele and Paterson, 2007; `https://eprint.iacr.org/2007/125`)

$\Rightarrow$ "Authenticated encryption" (This is hard to do properly.)

Today: MACs (symmetric authentication)

Message authentication code (MAC)

A MAC is a mapping $\mathcal{M} : \mathcal{K}(\times\mathcal{R}) \times \mathcal{X} \to \mathcal{T}$ that maps a key, message (and possibly a (random) nonce) to a *tag*.

- $\mathcal{K}$ is for instance $\{0,1\}^{128}$ (key space, secret)
- $\mathcal{R}$ is for instance $\{0,1\}^{64}$ ("nonce" space, public, either "random" or not)
- $\mathcal{X}$ is for instance $\bigcup_{\ell<2^{64}}\{0,1\}^{\ell}$ (message space)
- $\mathcal{T}$ is for instance $\{0,1\}^{256}$ ("tag" space)

MACs: what do we want?

Given a MAC $\mathcal{M}(k, \cdot)$ with an unknown key, it should be hard to:

- Given m, find t s.t. $\mathcal{M}(k, m) = t$ (*Universal forgery*)
- Find m, t s.t. $\mathcal{M}(k, m) = t$ (*Existential forgery*)
- (Of course, retrieving k leads to those)

UF: ability to forge a tag for **any** message

EF: ability to forge a tag for **some** message

UF $\Rightarrow$ EF

More generally, we want $\mathcal{M}(k, \cdot)$ to be like a "random function"

Attacking a MAC: what complexity?

The complexity of an attack depends (among others) on:

- Its time (T) complexity ("how many operations need to be computed?")
- Its memory (M) complexity ("how much storage do I need?")
  - The memory type: sequential? RAM?
- Its query/data (D) complexity ("how many black box/oracle access are needed?")
  - The query type: known message, chosen message?
- Its success probability (p)

Attacking a MAC: example

Take $\mathcal{M} : \{0,1\}^{128} \times \mathcal{X} \to \{0,1\}^{64}$. One has UF attacks with:

- $\mathsf{T} = 1$, $p = 2^{-64}$
- $\mathsf{T} = 2^{128}$, $\mathsf{M} = 1$, $\mathsf{D} = 3$, $p \approx 1$

And this *generically* (regardless of what $\mathcal{M}$ is)

Generic v. dedicated attacks

Generic attack:

- Unavoidable (in a computational setting)
- Complexity only depends on security params & objectives
- Always work "with some probability"
- Dictates key, block sizes etc. (cf. first lecture)

Dedicated attack:

- What "breaks" a specific scheme (primitive, protocol...)
- (Always) better than the corresponding generic attack

Comments

- An algorithm may have no dedicated attack, but could be too weak against generic ones
  - Example: the Trivium stream cipher (80-bit keys)
- An algorithm may be broken (by a dedicated attack) but could still be used securely in practice. THIS IS HOWEVER STRONGLY ADVISED AGAINST!
  - Example: *preimages* for the MD4 hash function, $T = 2^{95}$ (Zhong & Lai, 2012) instead of 2^{128}

Examples

Generic:

- Guessing the key (of a block cipher, a MAC, etc.)
- Guessing the tag (produced by a MAC)
- Finding collisions in the outputs of CBC encryption
- (Factoring an RSA modulus)

Dedicated:

- (Finding a DES key using linear cryptanalysis)
- (Computing collisions for SHA-1 using differential cryptanalysis)
- (Recovering an RSA private key using continued fractions)

Back to MACs: how to build 'em?

- From scratch
- Using a block cipher in a "MAC mode"
- Ditto, with a *hash function*
- Using a "polynomial" hash function
- Etc.

MACs from block ciphers: CBC-MAC example

Observation:

- The last block of CBC-ENC(m) "strongly depends" on the entire message
- ⇒ Take MAC(m) = LastBlockOf(CBC-ENC(m))
- Not quite secure as is, but overall a sound idea (cf. TD)

Advantage:

- "Only" need a block cipher

Disadvantage:

- Not the fastest approach

⇒ Alternative: polynomial MACs

Polynomials

"Polynomials = vectors"

Let $m = \begin{pmatrix} m_0 & m_1 & \dots & m_{n-1} \end{pmatrix}$ be a vector of k^n, one can interpret it as $M = m_0 + m_1 X + \ldots + m_{n-1} X^{n-1}$, a degree-$(n-1)$ polynomial of $k[X]$.

Polynomial evaluation

Let $M \in k[X]$ be a degree-$(n-1)$ polynomial, the *evaluation* of M on an element of k is given by the map $\mathrm{eval}(M, \cdot) : x \mapsto m_0 + m_1 x + \ldots + m_{n-1} x^n$.

Polynomial hash functions

Polynomial hash function

Let $m \in k^n$ be a "message". The "hash" of $m \equiv M \in k[X]$ for the function $\mathcal{H}_x$ is given by $\mathrm{eval}(M, x)$.

Some properties:

$\mathcal{H}_x$ is linear (over k)

- $\mathcal{H}_x(a + b) = \mathcal{H}_x(a) + \mathcal{H}_x(b)$

$\forall n \in \mathbb{N}^*,\ \forall x \in k,\ \forall a \in k^n,$

- $\Pr[\mathcal{H}_x(b) = \mathcal{H}_x(a) : b \stackrel{\$}{\leftarrow} k^n]$
- $= \Pr[\mathcal{H}_x(b - a) = 0 : b \stackrel{\$}{\leftarrow} k^n]$
- $= \Pr[\mathrm{eval}(B - A, x) = 0 : B \stackrel{\$}{\leftarrow} k[X]] \leq (n-1)/\#k$

How's that useful?

W.h.p., $\neq m \Rightarrow \neq \mathcal{H}_x(m)$

- E.g. take $\#k \approx 2^{128}$, $n = 2^{32}$, the "collision probability" between two messages is $\leq 2^{-96=32-128}$
- This is "optimum"

Problem: for a MAC, linearity is a weakness! (cf. TD)

- One way to solve this: encrypt the result of the hash with a block cipher!

Polynomial MACs

Toy polynomial MAC

Let $\mathcal{H} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a polynomial hash function family, $\mathcal{E} : \mathcal{K}' \times \mathcal{Y} \to \mathcal{Y}$ be a block cipher. The MAC $\mathcal{M} : \mathcal{K} \times \mathcal{K}' \times \mathcal{X} \to \mathcal{Y}$ is defined as $\mathcal{M}(k, k', m) = \mathcal{E}(k', \mathcal{H}_k(m))$.

(Remark: not randomized)

Advantage of polynomial MACs:

- Fast
- Good and “simple” security
    - But still rely on block ciphers and friends!

Examples: UMAC; VMAC; Poly1305-AES; NaT (more sophisticated variant of the above), NaK, HaT, HaK

Re: finite fields

Polynomial hash functions $\Rightarrow$ need large finite fields (for good sec.)
Two options:

- *Prime fields* $\mathbb{Z}/p\mathbb{Z}$ for a large prime p (e.g. $p = 2^{130} - 5$)
- *Extension fields* $\mathbb{F}_q$, $q = p^n$ for a prime p (e.g. $q = 2^{128}$)

So:

- How do you "build" extension fields?

$\Rightarrow$ Let's see *Linear Feedback Shift Registers* (LFSRs) first

LFSRs

LFSR (type 1)

An LFSR of length n over a field 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, $r_i \in k$

LFSR (type 2)

An LFSR of length n over a field 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, $r_i \in k$

Theorem: The two above definitions are "equivalent"

Characterization

An LFSR is fully determined by:

- Its base field k
- Its state 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 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
- For simplicity: "just a notation"

Example:

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

Why should I care about those?

- Useful as a basis for stream ciphers (in the olden times, mostly)
- One way to define/compute with extension fields (e.g. example from previous slide; see more next week)
- (Notice that the structure is kind of like a Feistel)
- It's beautiful?

Next week

- Hash functions (not linear ones this time)
- Extensions of $\mathbb{F}_2$