Cryptology complementary

✧

Message Authentication Codes, Authenticated Encryption

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

2020–04

Authentication (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

⇒ "Authenticated encryption" (This is hard to do properly.)

Today: MACs (symmetric authentication)

Message authentication code (MAC)

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

- $\mathcal{K}$ is for instance $\{0,1\}^{128}$ (key space, secret)
- $\mathcal{N}$ 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)

$\Rightarrow$ The tag is a "link" between a message and a key

- Note: MACs are not the *only* way to provide authentication

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** messages

UF $\Rightarrow$ EF

MACs: really, what do we want?

More generally, we want $\mathcal{M}(k, \cdot)$ to be like a "variable input-length (pseudo-) random function"
$\rightsquigarrow$ (VIL-) PRF security:

- An adversary has access to an oracle $\mathbb{O}$
- In one world, $\mathbb{O} \xleftarrow{\$} \mathrm{Func}(\mathcal{X}, \mathcal{T})$
- In another, $k \xleftarrow{\$} \mathcal{K}$, $\mathbb{O} = \mathcal{M}(k, \cdot)$
- The adversary cannot tell in which world he lives

Where $\mathrm{Func}(\mathcal{X}, \mathcal{T})$ are the functions from the message to the tag space
$\rightsquigarrow$ Define $\mathbf{Adv}^{\mathrm{PRF}}$ in the same way as $\mathbf{Adv}^{\mathrm{PRP}}$

VIL-PRF $\Rightarrow$ MAC, but the converse is not true (Exercise: can you show why?)

So, how to build a MAC?

- 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
- $\Rightarrow$ Take MAC(m) = LastBlockOf(CBC-ENC(m))
- Not quite secure as is, but overall a sound idea

Advantage:

- "Only" need a block cipher

Disadvantage:

- Not the fastest approach

Polynomials

“Polynomials = vectors”

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

Polynomial evaluation

Let $M \in \mathbb{K}[X]$ be a degree-$(n-1)$ polynomial, the *evaluation* of M on an element of $\mathbb{K}$ 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 $m \in \mathbb{K}^n$ be a "message". The "hash" of $m \equiv M = m_0 X + m_1 X^2 + \ldots \in \mathbb{K}[X]$ for the function $\mathcal{H}_x$ is given by $\text{eval}(M, x)$. (We want a degree-n polynomial here, for the evaluation to "mix" m_0 with the key)

Some properties:

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

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

$\forall a, b \neq a \in \mathbb{K}^n$,

- $\Pr[\mathcal{H}_x(b) = \mathcal{H}_x(a) : x \xleftarrow{\$} \mathbb{K}]$
- $= \Pr[\mathcal{H}_x(b - a) = 0 : x \xleftarrow{\$} \mathbb{K}]$
- $= \Pr[\text{eval}(B - A, x) = 0 : x \xleftarrow{\$} \mathbb{K}] \leq \deg(B - A)/\#\mathbb{K} \leq n/\#\mathbb{K}$

How's that useful?

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

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

Problem: for a MAC, linearity is a weakness!

- 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
Remark: Historically, most polynomial MACs use encryption with a stream cipher. This makes them more vulnerable if the key is reused.

Polynomial Hash functions: implementation

How do you implement eval(M, k) efficiently? A possibility:

- Use Horner's rule:
  $m_0 k + m_1 k^2 + \ldots = k \times (m_0 + k \times (m_1 + k \times (\ldots$
- So only need an efficient multiplication by the constant k:

```
// assuming a field of characteristic two
res = mulK(m[n-1]);
for (int i = n-2; i >= 0; i--)
        res = mulK(m[i] ^ res);
```

- In practice we don't care what message block is what coefficient $\rightsquigarrow$ start the loop from zero

Polynomial Hash functions: field choice

How do you choose $\mathbb{K}$, and how do you implement `mulK` then?

- Prime field option, e.g. $\mathbb{F}_{2^{130}-5} \rightsquigarrow$ use floating-point arithmetic. Good in software. Used in Poly1305 (Bernstein, 2005)
- Binary field option, e.g. $\mathbb{F}_{2^{128}} \rightsquigarrow$ use a precomputed matrix of multiplication or `pclmulqdq`-like instructions. Hardware-friendly, okay in software. Can also use various techniques from the algebraic computation folks. Used in GCM (McGrew & Viega, 2005)
- Don't use a single field but a multi-stage strategy, e.g. VMAC (Krovetz, 2006). Extremely fast in software. (In fact VMAC also relies on *inner-product* H.F. in addition to polynomial ones)

Introducing Authenticated-Encryption

The "modern" view:
If you must never encrypt w/o authentication, why separating the two? $\Rightarrow$ Authenticated-Encryption

- Maybe more efficient (less redundancy)?
- Maybe more secure (no careless combinations)?
- Maybe more complex

$\leadsto$ AEAD (Authenticated-Encryption with Associated Data)

AEAD

AEAD

An AEAD scheme is a pair of mappings $(\mathcal{E}, \mathcal{D})$ with:
$\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{X} \to \mathcal{C}$
$\mathcal{D} : \mathcal{K} \times \mathcal{C} \to \mathcal{X} \cup \{\perp\}$

- $\mathcal{E}$ encrypts a message from $\mathcal{X}$ with a key and a nonce, and authenticates it together with associated data from $\mathcal{A}$
- $\mathcal{D}$ decrypts a ciphertext and returns the message if authentication is successful, or $\perp$ ("bottom") otherwise
- Security is typically analysed w.r.t. IND-CPA (for confidentiality) and IND-CTXT (for integrity)

AEAD designs

An AEAD scheme can be built in many ways:

- By combining a BC mode w/ a MAC (e.g. GCM: CTR mode + a polynomial MAC)
- As a single BC mode (e.g. OCB)
- From a permutation/sponge consruction (e.g. Keyak)
- From a hash function (e.g. OMD)
- From a variable input-length wide-block block cipher (e.g. AEZ)
- Etc.

AEAD: A quick hash function example

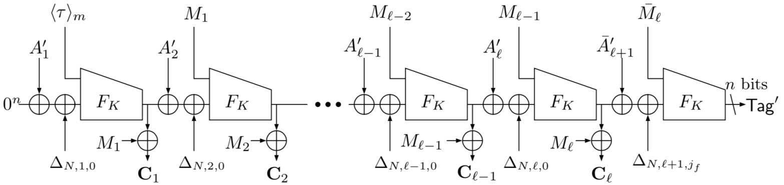

Figure: The p-OMD mode (excerpt; source: p-OMD specifications)

pure Offset Merkle-Damgård (Reyhanitabar et al., 2015), based on a keyed hash function (e.g. SHA-256 w/ semi-secret message)

AE: A quick VIL-WBC idea

If $\mathcal{E} : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ is a block cipher, one can encrypt *and* authenticate any message m of fixed length $b < n$ by:

- Computing $c = \mathcal{E}(k, m \| 0^{n-b_1} \| r)$, where r are $b_2 := b - b_1$ random bits
- Decrypting c to m iff. $\mathcal{E}^{-1}(k, c) = m \| 0^{n-b_1} \| *$

If $\mathcal{E}$ is a "good" SPRP, it is "hard" for an adversary to forge $\hat{c}$ s.t. $\mathcal{E}^{-1}(\hat{c})$ has $n - b_1$ zeroes at specific positions (roughly: success prob. $\approx 2^{b_1 - n}$)

$\rightsquigarrow$ Good paradigm, but very limited if $\mathcal{E}$ has typical block size $n \leq 256$

VIL-WBC

(VIL)-[W]BC

A *Variable input-length wide block cipher* is a family $\mathcal{W} = \{\mathcal{E}^\ell\}$ of mappings $\mathcal{E}^\ell : \mathcal{K} \times \mathcal{X}_\ell \to \mathcal{X}_\ell$ s.t. for all ℓ, $\mathcal{E}^\ell$ is a block cipher, where $\ell \in \mathcal{S} \subseteq \mathbb{N}$

- One can for instance take $\mathcal{X}_\ell = \{0,1\}^\ell$, $\ell \in [2^7, 2^{64}]$
- The SPRP security of $\mathcal{W}$ is defined as the $\min_\ell$ SPRP security of $\mathcal{E}^\ell$
- ¿The notion of VIL-WBC is (different and in some way) stronger than IND-CPA/CCA symmetric encryption ‽
  - Exercise: Why isn't encryption with CBC mode w/ a fixed IV a good VIL-WBC?

VIL-WBC constructions

Some various strategies have been proposed to build VIL-WBC

- Sequential two-pass (e.g. CBC-MAC feeding CTR, Bellare and Rogaway, 1999; CBC forward and backward, Houley; Matyas, 1999)
- Wide Feistel (e.g. Naor and Reingold, 1997 $\rightsquigarrow$ Mr Monster Burrito, Bertoni et al., 2014, and several others)
- Parallel Feistel (e.g. AEZ, Hoang et al., 2014)

Maybe not the easiest/fastest way, but conceptually beautiful

Conclusion

- Authentication is essential
- Most of the time, both encryption and authentication are needed
- The "modern" way: do both at the same time
- Still an active research topic (cf. the perpetual CAESAR competition $\rightsquigarrow$ `https://competitions.cr.yp.to/caesar.html`)

Appendix: a list of existing MACs

AMAC, BMAC, CMAC, DMAC, EMAC, FMAC, GMAC, HMAC, IMAC, JMAC, KMAC, LMAC, MMAC, NMAC, OMAC, PMAC, QMAC, RMAC, SMAC, TMAC, UMAC, VMAC, WMAC, XMAC, YMAC, ZMAC, PelicanMAC, SandwichMAC (see Karpman & Mennink, CRYPTO RUMP 2017 for a review)