Message Authentication Codes (MACs)

Tung Chou

Technische Universiteit Eindhoven, The Netherlands

October 8, 2015

About Me

About Me

Tung Chou (Tony)

About Me

Tung Chou (Tony)

- Ph.D. student of Daniel J. Bernstein & Tanja Lange

About Me

Tung Chou (Tony)

- Ph.D. student of Daniel J. Bernstein & Tanja Lange
- Research topics: Post-quantum crypto, ECC, MAC design.

About Me

Tung Chou (Tony)

- Ph.D. student of Daniel J. Bernstein & Tanja Lange
- Research topics: Post-quantum crypto, ECC, MAC design.
- Email: t.chou@tue.nl

Outline

Outline

- Introduction

Outline

- Introduction

- HMAC

Outline

- Introduction
- HMAC
- Universal-hash based MACs
  - Poly1305
  - security issues
  - software implementation issues

Outline

- Introduction
- HMAC
- Universal-hash based MACs
  - Poly1305
  - security issues
  - software implementation issues
- Diffie–Hellman key exchange

What are MACs?

What are MACs?

- On Wikipedia:

> *"a message authentication code (often MAC) is a short piece of information used to authenticate a message and to provide integrity and authenticity assurances on the message. Integrity assurances detect accidental and intentional message changes, while authenticity assurances affirm the message's origin"*

Digital Signatures

Digital Signatures

- Construction:

$$\text{message } (m) \longrightarrow \boxed{\text{hash}} \longrightarrow h \quad \boxed{\text{TP}} \quad \text{signature } (s)$$

$$h \xrightarrow{sk} s, \qquad s \xrightarrow{pk} h$$

Digital Signatures

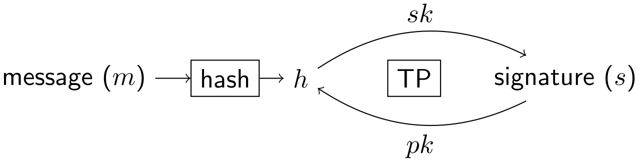
- Construction:

$$\begin{tikzcd} \text{message } (m) \arrow[r] & \boxed{\text{hash}} \arrow[r] & h \arrow[rr, bend left, "sk"] & \boxed{\text{TP}} & \text{signature } (s) \arrow[ll, bend left, "pk"] \end{tikzcd}$$

- Usage:

Digital Signatures

- Construction:



- Usage:
  - S computes h and the $\mathsf{SIGN}_{sk}(h)$.
  - S sends (m, s).

Digital Signatures

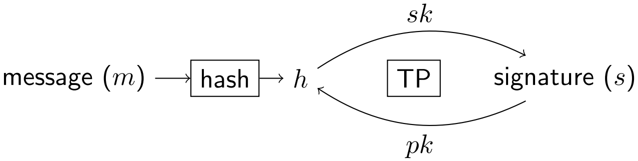
- Construction:

$$
\begin{tikzcd}
\text{message } (m) \arrow[r] & \boxed{\text{hash}} \arrow[r] & h \arrow[rr, bend left, "sk"] & \boxed{\text{TP}} & \text{signature } (s) \arrow[ll, bend left, "pk"]
\end{tikzcd}
$$

- Usage:
  - S computes h and the $\text{SIGN}_{sk}(h)$.
  - S sends (m, s).
  - V gets (m', s').

Digital Signatures

- Construction:



- Usage:
  - S computes h and the $\mathsf{SIGN}_{sk}(h)$.
  - S sends (m, s).
  - V gets (m', s').
  - V computes and check $\mathsf{hash}(m') = \mathsf{VERIFY}_{pk}(s')$.

Digital Signatures

- Construction:

$$\begin{tikzcd} \text{message } (m) \arrow[r] & \boxed{\text{hash}} \arrow[r] & h \arrow[rr, bend left, "sk"] & \boxed{\text{TP}} & \text{signature } (s) \arrow[ll, bend left, "pk"] \end{tikzcd}$$

- Usage:
  - S computes h and the $\text{SIGN}_{sk}(h)$.
  - S sends (m, s).
  - V gets (m', s').
  - V computes and check $\text{hash}(m') = \text{VERIFY}_{pk}(s')$.

- Security

Digital Signatures

- Construction:

$$\begin{tikzcd} \text{message } (m) \arrow[r] & \boxed{\text{hash}} \arrow[r] & h \arrow[rr, bend left, "sk"] \arrow[rr, phantom, "\boxed{\text{TP}}"] & & \text{signature } (s) \arrow[ll, bend left, "pk"] \end{tikzcd}$$

- Usage:
  - S computes h and the $\text{SIGN}_{sk}(h)$.
  - S sends (m, s).
  - V gets (m', s').
  - V computes and check $\text{hash}(m') = \text{VERIFY}_{pk}(s')$.

- Security
  - attacker should not be able to forge a valid (m, s) pair

Digital Signatures

- Construction:

$$
\begin{tikzcd}
\text{message } (m) \arrow[r] & \boxed{\text{hash}} \arrow[r] & h \arrow[rr, bend left, "sk"] & \boxed{\text{TP}} & \text{signature } (s) \arrow[ll, bend left, "pk"]
\end{tikzcd}
$$

- Usage:
  - S computes h and the $\text{SIGN}_{sk}(h)$.
  - S sends (m, s).
  - V gets (m', s').
  - V computes and check $\text{hash}(m') = \text{VERIFY}_{pk}(s')$.

- Security
  - attacker should not be able to forge a valid (m, s) pair
  - attacker might have collected many (m, s) pairs

Message Authentication Codes

Message Authentication Codes

- "Keyed hash function":

$$\text{message } (m) \longrightarrow \boxed{\text{MAC algorithm}} \longrightarrow \text{tag/authenticator } (t)$$

$$\uparrow$$

$$\text{shared secret key } (r)$$

Message Authentication Codes

- “Keyed hash function”:

$$\begin{tikzcd} \text{message } (m) \arrow[r] & \boxed{\text{MAC algorithm}} \arrow[r] & \text{tag/authenticator } (t) \\ & \text{shared secret key } (r) \arrow[u] & \end{tikzcd}$$

- Usage:

Message Authentication Codes

- "Keyed hash function":

$$\begin{tikzcd} \text{message } (m) \arrow[r] & \boxed{\text{MAC algorithm}} \arrow[r] & \text{tag/authenticator } (t) \\ & \text{shared secret key } (r) \arrow[u] & \end{tikzcd}$$

- Usage:
  - S computes $t = \mathsf{MAC}_r(m)$ and sends (m, t).

Message Authentication Codes

- "Keyed hash function":

$$\begin{tikzcd}
\text{message } (m) \arrow[r] & \boxed{\text{MAC algorithm}} \arrow[r] & \text{tag/authenticator } (t) \\
 & \text{shared secret key } (r) \arrow[u] &
\end{tikzcd}$$

- Usage:
  - S computes $t = \mathsf{MAC}_r(m)$ and sends (m, t).
  - R gets (m', t').

Message Authentication Codes

- "Keyed hash function":

$$\begin{tikzcd} \text{message } (m) \arrow[r] & \boxed{\text{MAC algorithm}} \arrow[r] & \text{tag/authenticator } (t) \\ & \text{shared secret key } (r) \arrow[u] & \end{tikzcd}$$

- Usage:
  - S computes $t = \text{MAC}_r(m)$ and sends (m, t).
  - R gets (m', t').
  - R computes and checks $\text{MAC}_r(m') = t'$.

Message Authentication Codes

- "Keyed hash function":

$$\begin{tikzcd} \text{message } (m) \arrow[r] & \boxed{\text{MAC algorithm}} \arrow[r] & \text{tag/authenticator } (t) \\ & \text{shared secret key } (r) \arrow[u] & \end{tikzcd}$$

- Usage:
  - S computes $t = \mathsf{MAC}_r(m)$ and sends (m, t).
  - R gets (m', t').
  - R computes and checks $\mathsf{MAC}_r(m') = t'$.
- Security

Message Authentication Codes

- "Keyed hash function":

$$\text{message } (m) \longrightarrow \boxed{\text{MAC algorithm}} \longrightarrow \text{tag/authenticator } (t)$$

$$\uparrow$$

$$\text{shared secret key } (r)$$

- Usage:
  - S computes $t = \mathsf{MAC}_r(m)$ and sends (m, t).
  - R gets (m', t').
  - R computes and checks $\mathsf{MAC}_r(m') = t'$.

- Security
  - attacker should not be able to forge a valid (m, t) pair

Message Authentication Codes

- "Keyed hash function":

$$\begin{tikzcd} \text{message } (m) \arrow[r] & \boxed{\text{MAC algorithm}} \arrow[r] & \text{tag/authenticator } (t) \\ & \text{shared secret key } (r) \arrow[u] & \end{tikzcd}$$

- Usage:
  - S computes $t = \mathsf{MAC}_r(m)$ and sends (m, t).
  - R gets (m', t').
  - R computes and checks $\mathsf{MAC}_r(m') = t'$.

- Security
  - attacker should not be able to forge a valid (m, t) pair
  - attacker might have collected many (m, t) pairs

MACs vs Signatures

MACs vs Signatures

	MACs	Signatures
Integrity	yes	yes
Authenticity	yes	yes
Non-repudiation	no	yes
Key	secret-key	public-key

MACs vs Signatures

	MACs	Signatures
Integrity	yes	yes
Authenticity	yes	yes
Non-repudiation	no	yes
Key	secret-key	public-key

"Non-repudiation is about Alice showing to Bob a proof that some data really comes from Alice, such that not only Bob is convinced, but Bob also gets the assurance that he could show the same proof to Charlie, and Charlie would be convinced, too"

MACs vs Signatures

	MACs	Signatures
Integrity	yes	yes
Authenticity	yes	yes
Non-repudiation	no	yes
Key	secret-key	public-key

"Non-repudiation is about Alice showing to Bob a proof that some data really comes from Alice, such that not only Bob is convinced, but Bob also gets the assurance that he could show the same proof to Charlie, and Charlie would be convinced, too"

✍ secret-key crypto is "fast"

HMAC

HMAC

- Build MAC from hash functions

HMAC

- Build MAC from hash functions

- A naive construction:

$$t = H(r \mid\mid m)$$

HMAC

- Build MAC from hash functions

- A naive construction:

$$t = H(r \,||\, m)$$

- Merkle–Damgård construction based hashes (e.g., MD5, SHA1)

$$\begin{tikzcd}
 & m_1 \arrow[d] & m_2 \arrow[d] & & m_\ell \arrow[d] & \\
IV \arrow[r] & \boxed{f} \arrow[r] & \boxed{f} \arrow[r] & \cdots \arrow[r] & \boxed{f} \arrow[r] & h
\end{tikzcd}$$

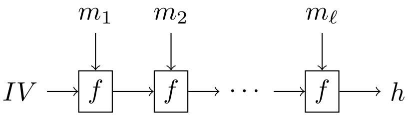
HMAC

- Build MAC from hash functions

- A naive construction:

$$t = H(r \mid\mid m)$$

- Merkle–Damgård construction based hashes (e.g., MD5, SHA1)



- Length extension attack: $h' = f(h, m_{\ell+1})$

HMAC (cont.)

HMAC (cont.)

- Another construction:

$$t = H(m \mid\mid r)$$

HMAC (cont.)

- Another construction:

$$t = H(m \mid\mid r)$$

- HMAC:

$$t = H\left((r \oplus p_o) || H((r \oplus p_i) || m)\right)$$

HMAC (cont.)

- Another construction:

$$t = H(m \mid\mid r)$$

- HMAC:

$$t = H\left((r \oplus p_o) || H((r \oplus p_i)||m)\right)$$

- HMAC-SHA1

HMAC (cont.)

- Another construction:

$$t = H(m \mid\mid r)$$

- HMAC:

$$t = H\left((r \oplus p_o) || H((r \oplus p_i) || m)\right)$$

- HMAC-SHA1
  - widely used in Internet applications

HMAC (cont.)

- Another construction:

$$t = H(m \mid\mid r)$$

- HMAC:

$$t = H\left((r \oplus p_o) || H((r \oplus p_i) || m)\right)$$

- HMAC-SHA1
  - widely used in Internet applications
  - 5.18 Sandy Bridge cycles/byte

HMAC (cont.)

- Another construction:

$$t = H(m \mid\mid r)$$

- HMAC:

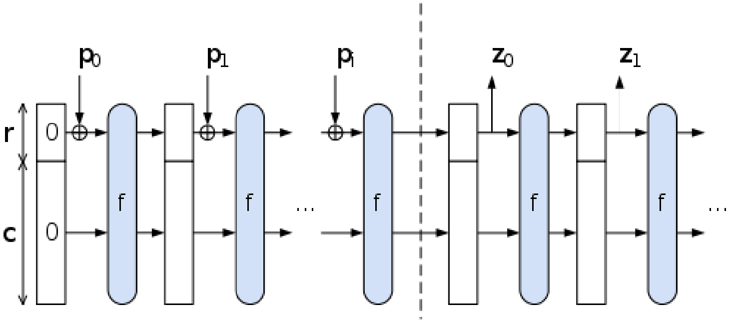
$$t = H\left((r \oplus p_o) || H((r \oplus p_i) || m)\right)$$

- HMAC-SHA1
  - widely used in Internet applications
  - 5.18 Sandy Bridge cycles/byte

✍ Reality: the most commonly used scheme might not be the best

SHA3

The "Sponge" construction:



http://en.wikipedia.org/wiki/SHA-3

The Wegman–Carter construction

The Wegman–Carter construction

- Why?

The Wegman–Carter construction

- Why?
  - provides information theoretic security

The Wegman–Carter construction

- Why?
  - provides information theoretic security
  - usually involves field/ring arithmetic

The Wegman–Carter construction

- Why?
  - provides information theoretic security
  - usually involves field/ring arithmetic
  - better performance than HMAC

The Wegman–Carter construction

- Why?
  - provides information theoretic security
  - usually involves field/ring arithmetic
  - better performance than HMAC

The Wegman–Carter construction

- Why?
  - provides information theoretic security
  - usually involves field/ring arithmetic
  - better performance than HMAC
- Construction

The Wegman–Carter construction

- Why?
  - provides information theoretic security
  - usually involves field/ring arithmetic
  - better performance than HMAC

- Construction
  - "universal" hash function + one-time pad:

$$h_r(m_n) \oplus s_n$$

The Wegman–Carter construction

- Why?
  - provides information theoretic security
  - usually involves field/ring arithmetic
  - better performance than HMAC

- Construction
  - "universal" hash function + one-time pad:

$$h_r(m_n) \oplus s_n$$

  - universal hash: low differential probability

The Wegman–Carter construction

- Why?
  - provides information theoretic security
  - usually involves field/ring arithmetic
  - better performance than HMAC

- Construction
  - "universal" hash function + one-time pad:

$$h_r(m_n) \oplus s_n$$

  - universal hash: low differential probability
  - one-time pad hides all information about the key

Poly1305

Poly1305

- Construction:

$$t = (((m_1 r^{\ell} + m_2 r^{\ell-1} + \cdots + m_{\ell} r) \mod 2^{130} - 5) + s) \mod 2^{128}$$

Poly1305

- Construction:

$$t = (((m_1 r^{\ell} + m_2 r^{\ell-1} + \cdots + m_{\ell} r) \mod 2^{130} - 5) + s) \mod 2^{128}$$

  - $2^{130} - 5$ is a prime
  - r, s are shared secret 128-bit values

Poly1305

- Construction:

$$t = (((m_1 r^\ell + m_2 r^{\ell-1} + \cdots + m_\ell r) \mod 2^{130} - 5) + s) \mod 2^{128}$$

  - $2^{130} - 5$ is a prime
  - r, s are shared secret 128-bit values
  - $m_{i<\ell}$ is the ith 128-bit block of m padded by 1.
  - m_ℓ is the "remainder" of m padded by 1.

Poly1305

- Construction:

$$t = (((m_1 r^\ell + m_2 r^{\ell-1} + \cdots + m_\ell r) \mod 2^{130} - 5) + s) \mod 2^{128}$$

  - $2^{130} - 5$ is a prime
  - r, s are shared secret 128-bit values
  - $m_{i<\ell}$ is the ith 128-bit block of m padded by 1.
  - m_ℓ is the "remainder" of m padded by 1.

- Without proper padding?

Poly1305

- Construction:

$$t = (((m_1 r^{\ell} + m_2 r^{\ell-1} + \cdots + m_\ell r) \mod 2^{130} - 5) + s) \mod 2^{128}$$

  - $2^{130} - 5$ is a prime
  - r, s are shared secret 128-bit values
  - $m_{i<\ell}$ is the ith 128-bit block of m padded by 1.
  - m_ℓ is the "remainder" of m padded by 1.

- Without proper padding?
  - $m =$ 'FF', $m' =$ 'FF','00'
  - zero-pad the message obtain a 128-bit block

$$m_1 = m_1' = \texttt{'FF', '00', ..., '00'}$$

Poly1305

- Construction:

$$t = (((m_1 r^\ell + m_2 r^{\ell-1} + \cdots + m_\ell r) \mod 2^{130} - 5) + s) \mod 2^{128}$$

  - $2^{130} - 5$ is a prime
  - r, s are shared secret 128-bit values
  - $m_{i<\ell}$ is the ith 128-bit block of m padded by 1.
  - m_ℓ is the "remainder" of m padded by 1.

- Without proper padding?
  - $m =$ `'FF'`, $m' =$ `'FF','00'`
  - zero-pad the message obtain a 128-bit block

$$m_1 = m'_1 = \texttt{'FF'},\ \texttt{'00'},\ \ldots,\ \texttt{'00'}$$

- Speed: 1.22 Sandy Bridge cycles/byte

Poly1305: avoiding security issue

Poly1305: avoiding security issue

- What is wrong with “real” polynomial evaluation?

$$t = m_1 r^{\ell-1} + m_2 r^{\ell-2} + \cdots + m_\ell + s$$

Poly1305: avoiding security issue

- What is wrong with “real” polynomial evaluation?

$$t = m_1 r^{\ell-1} + m_2 r^{\ell-2} + \cdots + m_\ell + s$$

- The attacker forges a valid message–tag pair easily:

$$t + \Delta = m_1 r^{\ell-1} + m_2 r^{\ell-2} + \cdots + (m_\ell + \Delta) + s$$

Poly1305: avoiding security issue

- What is wrong with “real” polynomial evaluation?

$$t = m_1 r^{\ell-1} + m_2 r^{\ell-2} + \cdots + m_\ell + s$$

- The attacker forges a valid message–tag pair easily:

$$t + \Delta = m_1 r^{\ell-1} + m_2 r^{\ell-2} + \cdots + (m_\ell + \Delta) + s$$

- This does not provide low differential probability

Poly1305: avoiding security issue

Poly1305: avoiding security issue

- What is wrong with using the same pad twice?

$$t = m_1 r^\ell + m_2 r^{\ell-1} + \cdots + m_\ell r + s$$

$$t' = m'_1 r^\ell + m'_2 r^{\ell-1} + \cdots + m'_\ell r + s$$

Poly1305: avoiding security issue

- What is wrong with using the same pad twice?

$$t = m_1 r^\ell + m_2 r^{\ell-1} + \cdots + m_\ell r + s$$

$$t' = m_1' r^\ell + m_2' r^{\ell-1} + \cdots + m_\ell' r + s$$

- The attacker gets information of r by finding roots of

$$t - t' = (m_1 - m_1') r^\ell + (m_2 - m_2') r^{\ell-1} + \cdots + (m_\ell - m_\ell') r$$

Poly1305: avoiding security issue

- What is wrong with using the same pad twice?

$$t = m_1 r^\ell + m_2 r^{\ell-1} + \cdots + m_\ell r + s$$

$$t' = m'_1 r^\ell + m'_2 r^{\ell-1} + \cdots + m'_\ell r + s$$

- The attacker gets information of r by finding roots of

$$t - t' = (m_1 - m'_1) r^\ell + (m_2 - m'_2) r^{\ell-1} + \cdots + (m_\ell - m'_\ell) r$$

- "*nonce-misuse*" issue

Poly1305: avoiding security issue

- What is wrong with using the same pad twice?

$$t = m_1 r^\ell + m_2 r^{\ell-1} + \cdots + m_\ell r + s$$

$$t' = m'_1 r^\ell + m'_2 r^{\ell-1} + \cdots + m'_\ell r + s$$

- The attacker gets information of r by finding roots of

$$t - t' = (m_1 - m'_1) r^\ell + (m_2 - m'_2) r^{\ell-1} + \cdots + (m_\ell - m'_\ell) r$$

- "*nonce-misuse*" issue
  - In practice s is usually replaced by stream cipher output, e.g., $\mathrm{AES}_k(n)$ for m_n

Poly1305: avoiding security issue

- What is wrong with using the same pad twice?

$$t = m_1 r^\ell + m_2 r^{\ell-1} + \cdots + m_\ell r + s$$

$$t' = m'_1 r^\ell + m'_2 r^{\ell-1} + \cdots + m'_\ell r + s$$

- The attacker gets information of r by finding roots of

$$t - t' = (m_1 - m'_1) r^\ell + (m_2 - m'_2) r^{\ell-1} + \cdots + (m_\ell - m'_\ell) r$$

- "*nonce-misuse*" issue
  - In practice s is usually replaced by stream cipher output, e.g., $\mathsf{AES}_k(n)$ for m_n
  - HMAC does not use nonce

Poly1305: polynomial evaluation

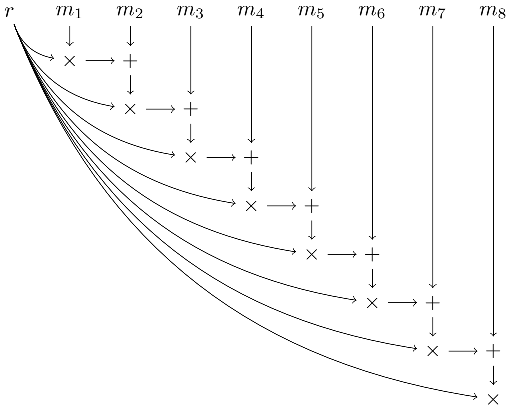
Poly1305: polynomial evaluation

Consider $m_1 r^8 + m_2 r^7 + \cdots + m_8 r$

Poly1305: polynomial evaluation

Consider $m_1 r^8 + m_2 r^7 + \cdots + m_8 r$

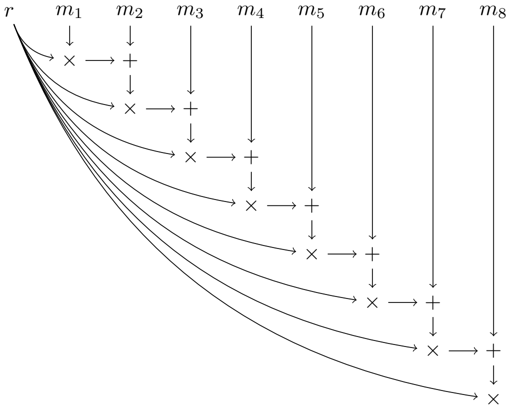
- *Horner's rule*:



Poly1305: polynomial evaluation

Consider $m_1 r^8 + m_2 r^7 + \cdots + m_8 r$

- *Horner's rule*:

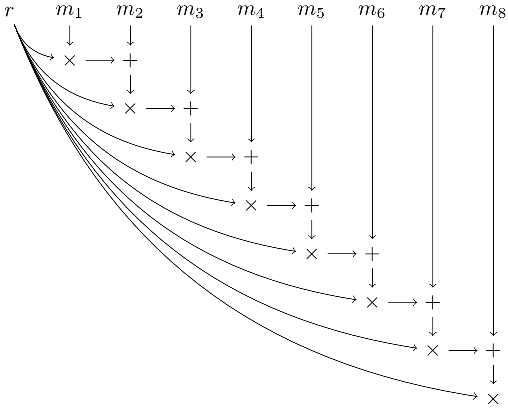


- n multiplications (and $n-1$ additions)

Poly1305: polynomial evaluation

Consider $m_1 r^8 + m_2 r^7 + \cdots + m_8 r$

- *Horner's rule*:



- n multiplications (and $n-1$ additions)
- The issue of being *"on-line"*

GMAC

GMAC

- The NIST-standard *authenticated encryption* scheme GCM

GMAC

- The NIST-standard *authenticated encryption* scheme GCM
  - *Galois Counter Mode*

GMAC

- The NIST-standard *authenticated encryption* scheme GCM
  - *Galois Counter Mode*
  - Special hardware support for AES-GCM in high-end CPUs

GMAC

- The NIST-standard *authenticated encryption* scheme GCM
  - *Galois Counter Mode*
  - Special hardware support for AES-GCM in high-end CPUs

- Polynomial evaluation MAC:

$$t = (m_1 r^\ell + m_2 r^{\ell-1} + \cdots + m_\ell r) + s$$

GMAC

- The NIST-standard *authenticated encryption* scheme GCM
  - *Galois Counter Mode*
  - Special hardware support for AES-GCM in high-end CPUs

- Polynomial evaluation MAC:

$$t = (m_1 r^\ell + m_2 r^{\ell-1} + \cdots + m_\ell r) + s$$

- Based on arithmetic in

$$\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(x^{128} + x^7 + x^2 + x + 1)$$

GMAC

- The NIST-standard *authenticated encryption* scheme GCM
  - *Galois Counter Mode*
  - Special hardware support for AES-GCM in high-end CPUs

- Polynomial evaluation MAC:

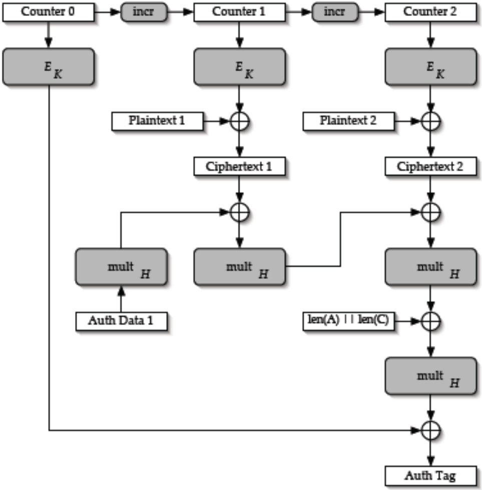
$$t = (m_1 r^\ell + m_2 r^{\ell-1} + \cdots + m_\ell r) + s$$

- Based on arithmetic in

$$\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(x^{128} + x^7 + x^2 + x + 1)$$

✍ Binary fields: better in hardware

GCM



http://en.wikipedia.org/wiki/Galois/Counter_Mode

GMAC: speeds

reference	platform	`PCLMUQDQ`	cycles per byte
Käsper–Schwabe 2009	Core 2	no	14.40
	Sandy Bridge	no	13.10
Krovetz–Rogaway 2011	Westmere	yes	2.00
Gueron 2013	Sandy Bridge	yes	1.79
	Haswell	yes	0.40

Auth256*

Auth256*

- Construction

Auth256*

- Construction
  - a *pseudo-dot-product* MAC:

$$t = (m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots + s$$

Auth256*

- Construction
  - a *pseudo-dot-product* MAC:

$$t = (m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots + s$$

  - base field $\mathbb{F}_{2^{256}} = \mathbb{F}_{2^8}[x]/(\phi)$. Tower field construction for $\mathbb{F}_{2^8}$.

Auth256*

- Construction
  - a *pseudo-dot-product* MAC:

$$t = (m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots + s$$

  - base field $\mathbb{F}_{2^{256}} = \mathbb{F}_{2^8}[x]/(\phi)$. Tower field construction for $\mathbb{F}_{2^8}$.
- Compared to GMAC

Auth256*

- Construction
  - a *pseudo-dot-product* MAC:

$$t = (m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots + s$$

  - base field $\mathbb{F}_{2^{256}} = \mathbb{F}_{2^8}[x]/(\phi)$. Tower field construction for $\mathbb{F}_{2^8}$.
- Compared to GMAC
  - higher security level

Auth256*

- Construction
  - a *pseudo-dot-product* MAC:

$$t = (m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots + s$$

  - base field $\mathbb{F}_{2^{256}} = \mathbb{F}_{2^8}[x]/(\phi)$. Tower field construction for $\mathbb{F}_{2^8}$.
- Compared to GMAC
  - higher security level
  - 0.5/1 multiplications per block

Auth256*

- Construction
  - a *pseudo-dot-product* MAC:

$$t = (m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots + s$$

  - base field $\mathbb{F}_{2^{256}} = \mathbb{F}_{2^8}[x]/(\phi)$. Tower field construction for $\mathbb{F}_{2^8}$.

- Compared to GMAC
  - higher security level
  - 0.5/1 multiplications per block
  - larger key size

Auth256*

- Construction
  - a *pseudo-dot-product* MAC:

$$t = (m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots + s$$

  - base field $\mathbb{F}_{2^{256}} = \mathbb{F}_{2^8}[x]/(\phi)$. Tower field construction for $\mathbb{F}_{2^8}$.
- Compared to GMAC
  - higher security level
  - 0.5/1 multiplications per block
  - larger key size
  - very different field construction for low bit operation count

Wegman–Carter construction: security

- "δ-xor-universal hash": For all distinct (m, m') and Δ, we have

$$Pr\left(\mathsf{Hash}_r(m) = \mathsf{Hash}_r(m') \oplus \Delta\right) \leq \delta$$

Wegman–Carter construction: security

- "δ-xor-universal hash": For all distinct (m, m') and Δ, we have

$$Pr\left(\mathsf{Hash}_r(m) = \mathsf{Hash}_r(m') \oplus \Delta\right) \leq \delta$$

- The one-time pad hides all information about the key r.
- The best strategy for the attacker is to guess.

Auth256: Security Proof

Hash values:

$$\begin{aligned} h =& (m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots + (m_{2\ell-1} + r_{2\ell-1})(m_{2\ell} + r_{2\ell}), \\ h' =& (m'_1 + r_1)(m'_2 + r_2) + (m'_3 + r_3)(m'_4 + r_4) + \cdots + (m'_{2\ell-1} + r_{2\ell-1})(m'_{2\ell} + r_{2\ell}). \end{aligned}$$

Auth256: Security Proof

Hash values:

$$\begin{aligned} h =& (m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots + (m_{2\ell-1} + r_{2\ell-1})(m_{2\ell} + r_{2\ell}), \\ h' =& (m'_1 + r_1)(m'_2 + r_2) + (m'_3 + r_3)(m'_4 + r_4) + \cdots + (m'_{2\ell-1} + r_{2\ell-1})(m'_{2\ell} + r_{2\ell}). \end{aligned}$$

Then $h = h' + \Delta$ if and only if

$$\begin{aligned} & r_1(m_2 - m'_2) + r_2(m_1 - m'_1) + r_3(m_4 - m'_4) + r_4(m_3 - m'_3) + \cdots \\ & \quad = \Delta + m'_1 m'_2 - m_1 m_2 + m'_3 m'_4 - m_3 m_4 + \cdots . \end{aligned}$$

Auth256: Security Proof

Hash values:

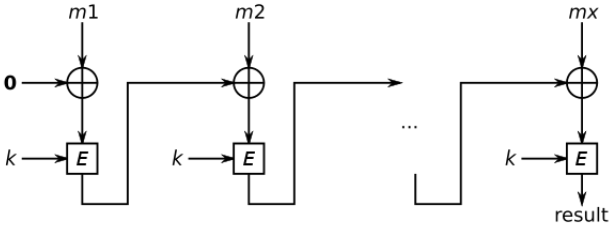
$$\begin{aligned} h =& (m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots + (m_{2\ell-1} + r_{2\ell-1})(m_{2\ell} + r_{2\ell}), \\ h' =& (m'_1 + r_1)(m'_2 + r_2) + (m'_3 + r_3)(m'_4 + r_4) + \cdots + (m'_{2\ell-1} + r_{2\ell-1})(m'_{2\ell} + r_{2\ell}). \end{aligned}$$

Then $h = h' + \Delta$ if and only if

$$\begin{aligned} r_1(m_2 - m'_2) + r_2(m_1 - m'_1) + r_3(m_4 - m'_4) + r_4(m_3 - m'_3) + \cdots \\ = \Delta + m'_1 m'_2 - m_1 m_2 + m'_3 m'_4 - m_3 m_4 + \cdots . \end{aligned}$$

$m \neq m'$ implies that there are at most $|K|^{2\ell-1}$ solutions for r.

CBC-MAC



http://en.wikipedia.org/wiki/CBC-MAC