Message Authentication Codes
and Hash Functions

Mike Reiter

A message authentication code is a triple $\Pi = (K, \text{MAC}, \text{VF})$ of efficiently computable algorithms

- $K$ is a randomized “key generation algorithm” that outputs a “key”
  
  $K \leftarrow K()$

  Keys($\Pi$) denotes all keys output by $K()$ with nonzero probability.

- MAC is a deterministic, randomized or stateful algorithm that takes a key $K$ and message $M \in \{0,1\}^*$ as input, and outputs $\bot$ or a “tag”
  
  $Tag \leftarrow \text{MAC}_K(M)$

  $Tag$ is of length $\tau$ (the “tag length”).

  The “message space” is all messages such that $\text{MAC}_K(M) \neq \bot$.

- VF takes a key $K$, message $M \in \{0,1\}^*$, and a tag $Tag$ as input and returns either 1 (accept) or 0 (reject)
  
  $d \leftarrow \text{VF}_K(M, Tag)$

- If $Tag \leftarrow \text{MAC}_K(M)$ and $Tag \neq \bot$, then $\text{VF}_K(M, Tag) = 1$. 
Unforgeability under CMA

- **Definition:** Let $\Pi = (\mathcal{K}, MAC, VF)$ be a message authentication code, and let $A$ be an adversary. Define

  Experiment $\text{Expt}_{\Pi}^{\text{uf-cma}}(A)$

  $K \leftarrow \mathcal{K}()$

  Run $A^{MAC_K(), VF_K()}$

  If $A$ queried $d \leftarrow VF_K(M, Tag)$ where $d = 1$ and $MAC_K(M)$ had not previously been queried then return 1 else return 0

  The *uf-cma advantage* of $A$ is defined as

  $$\text{Adv}_{\Pi}^{\text{uf-cma}}(A) = \Pr\left[\text{Expt}_{\Pi}^{\text{uf-cma}}(A) = 1\right]$$

Unforgeability under CMA

- For any $t, q_s, q_v, \mu_s, \mu_v$ we define the *uf-cma advantage* of $\Pi$ as

  $$\text{Adv}_{\Pi}^{\text{uf-cma}}(t, q_s, \mu_s, q_v, \mu_v) = \max_A \left\{ \text{Adv}_{\Pi}^{\text{uf-cma}}(A) \right\}$$

  where the maximum is over all $A$ having time complexity $t$ and making at most $q_s$ oracle queries of total length $\mu_s$ to $MAC_K$ and at most $q_v$ oracle queries of total length $\mu_v$ to $VF_K$

- Informally, $\Pi$ is “uf-cma secure” if the uf-cma advantage of $\Pi$ is small
MACs from Pseudorandom Functions

- Let $F$ be a function family
- Select $K \leftarrow_R \text{Keys}(F)$
- Define $\text{MAC}_K(m) = F_K(m)$ for $m \in \text{Dom}(F)$
- Define
  \[
  \text{VF}_K(m, t) = \begin{cases} 
  1 & \text{if } F_K(m) = t \\
  0 & \text{otherwise}
  \end{cases}
  \]

Security for PRF-based MACs

- Proposition: Let $F$: Keys$(F) \times \text{Dom}(F) \rightarrow \{0,1\}^L$ be a family of functions and let $\Pi$ denote the PRF-based MAC previously defined. Then

  \[
  \text{Adv}_{\Pi}^{\text{nf-cma}}(t, q_s, \mu_s, q_v, \mu_v) \leq \text{Adv}_{F}^{\text{prf}}(t, q_s + q_v, \mu_s + \mu_v) + \frac{q_v}{2^L}
  \]

  - Proof: Given an attacker $A$ for the MAC scheme that runs with constraints $t$, $q_s$, $q_v$, $\mu_s$, $\mu_v$, we construct a prf-distinguisher $B_A$ for $F$ that works under constraints $t$, $q_s + q_v$, $\mu_s + \mu_v$ such that

  \[
  \text{Adv}_{\Pi}^{\text{nf-cma}}(A) \leq \text{Adv}_{F}^{\text{prf}}(B_A) + \frac{q_v}{2^L}
  \]
Security for PRF-based MACs

Recall that $B_A$ is given an oracle for $f: \text{Dom}(F) \to \{0,1\}^L$.

Algorithm $B_{\mathsf{f}^{(\cdot)}}_A$
\begin{align*}
d & \leftarrow 0 \\
\text{Run } A, \text{ replying to its oracle queries as follows:} \\
\quad \text{When } A \text{ queries MAC} \? (M), \text{ return } f(M). \\
\quad \text{When } A \text{ queries VF} \? (M, \text{Tag}), \\
\qquad \text{if } f(M) = \text{Tag} \\
\qquad \quad \text{if MAC}_A(M) \text{ was not previously queried} \\
\qquad \quad \text{then } d \leftarrow 1 \\
\qquad \quad \text{return 1 to } A \\
\qquad \text{else return 0 to } A \\
\text{Until } A \text{ stops.} \\
\text{Output } d.
\end{align*}

Note that since in the prf-1 experiment, the experiment is exactly the same experiment that $A$ runs.
In addition, since in the prf-0 experiment, the probability that $A$ guesses the tag is $1/2^L$ per verification query.
CBC-MAC

- Historically a very popular method of creating MACs
- Uses CBC with zero initialization vector
  - the last ciphertext block is the tag
- But does it work?

**Proposition:** $\text{F} : \text{Keys}(\text{F}) \times \{0,1\}^l \to \{0,1\}^l$ be a family of functions, and let $\text{CBC}^m[\text{F}] : \{0,1\}^{ml} \to \{0,1\}^l$ denote the CBC-MAC function instantiated with $\text{F}$. Then,

$$\text{Adv}_{\text{CBC}^m[\text{F}]}^{\text{prf}}(t, q, qm) \leq \text{Adv}_{\text{F}}^{\text{prf}}(t', q', q' \ell) + \frac{3q^2m^2}{2^{l+1}}$$

where $q' = qm$ and $t' = t + O(qml)$.

The proof of this builds from the following two lemmas.

**Lemma 1:** For any $t$, $\text{Adv}_{\text{CBC}^m[\text{Func}(l, l)]}^{\text{prf}}(t, q, qm) \leq \frac{3q^2m^2}{2^{l+1}}$.

**Example:** Suppose $l = 128$ bits and we use $\text{CBC}^m[\text{Func}(l, l)]$ to authenticate $q = 2^{30}$ messages of $2^{10}$ blocks each. Then, no adversary, no matter how much time it invests, has advantage larger than $5.4 \times 10^{-15}$ of distinguishing these MACs from purely random strings.

**Lemma 2:** Let $A$ be a distinguisher that makes $q$ oracle queries and has running time $t$. Then there is a distinguisher $B_A$ such that

$$\text{Adv}_{\text{CBC}^m[\text{Func}(l, l)]}^{\text{prf}}(A) \leq \text{Adv}_{\text{F}}^{\text{prf}}(B_A) + \text{Adv}_{\text{CBC}^m[\text{Func}(l, l)]}^{\text{prf}}(A)$$

$B_A$ makes $q' = mq$ oracle queries and runs in time at most $t' = t + O(mq l)$ time.
CBC-MAC

Let’s assume the lemmas (we’ll prove Lemma 2 later), and show how this gives us the proposition.

Proof of proposition: Let \( A \) be a distinguisher that makes \( q \) oracle queries and takes time \( t \). Then,

\[
\text{Adv}^{\text{ref}}_{\text{CBC}}(A) \leq \frac{3q^2m^2}{2^{t+1}}
\]

by Lemma 1.

Now, let \( B_A \) be the distinguisher in Lemma 2. Then,

\[
\text{Adv}^{\text{ref}}_{\text{CBC}}(A) \leq \text{Adv}^{\text{ref}}_{F}(B_A) + \frac{3q^2m^2}{2^{t+1}}
\]

Now, we get

\[
\text{Adv}^{\text{ref}}_{\text{CBC}}(t, q) = \max_A \left| \text{Adv}^{\text{ref}}_{\text{CBC}}(A) \right| \leq \max_A \left\{ \text{Adv}^{\text{ref}}_{F}(B_A) + \frac{3q^2m^2}{2^{t+1}} \right\} \leq \max_B \left\{ \text{Adv}^{\text{ref}}_{F}(B) \right\} + \frac{3q^2m^2}{2^{t+1}}
\]

where \( \max \) is over all \( B \) taking time \( t' \) and making \( q' \) oracle queries.

\[
\leq \text{Adv}^{\text{ref}}_{F}(t', q') + \frac{3q^2m^2}{2^{t+1}}
\]
Now let’s prove Lemma 2. We have to build a distinguisher $B_A$ from the distinguisher $A$.

**Algorithm $B_A^{C(-)}$**

Run $A$.

For $i = 1 \ldots q$ do

When $A$ queries for $g(M_i)$, return $(CBC''(f))(M_i)$.

When $A$ outputs $b$, return $b$.

First consider that

$$
\text{Adv}_{F}^{\text{prf}}(B_A) = \Pr[\text{Expt}_{F}^{\text{prf},1}(B_A) = 1] - \Pr[\text{Expt}_{F}^{\text{prf},0}(B_A) = 1] \\
= \Pr[B_A^{f} = 1 \mid f \leftarrow_R F] - \Pr[B_A^{f} = 1 \mid f \leftarrow_R \text{Func}(l,l)] \\
= \Pr[A^g = 1 \mid g \leftarrow_R CBC^{m}(F)] - \Pr[A^g = 1 \mid g \leftarrow_R CBC^{m}[\text{Func}(l,l)]]
$$

In addition,

$$
\text{Adv}_{CBC''[\text{Func}(l,l)]}^{\text{prf}}(A) = \Pr[A^g = 1 \mid g \leftarrow_R CBC''[\text{Func}(l,l)]] - \Pr[A^g = 1 \mid g \leftarrow_R \text{Func}(ml,l)]
$$

Adding the two equations gives the result.
CBC-MAC

- Throughout this discussion, we have fixed \( m \), the number of blocks of the input message.
- In fact, CBC-MAC is not secure with variable-length inputs.
  - Work out an example.

Some attempts to “fix” it for variable length inputs:
- Append a block to the message containing the length, and then MAC.
  - Doesn’t work.
- Input-length key separation:
  \[
  \text{CBC}^m[f_K](x) = \text{CBC}^{m-1}[f_{K_{m-1}}](x) \quad \text{where} \quad K_m \leftarrow f_K(m)
  \]
- Map last block:
  \[
  \text{CBC}^m[f_{K_{m-1}}](x) = f_{K_1}(\text{CBC}^{m-1}[f_{K_1}](x))
  \]

Cryptographic Hash Functions

- Cryptographic hash functions map strings of different lengths to short, fixed-size outputs.
  - Examples are MD5, SHA-1, SHA-2.
  - Typically constructed to be “collision resistant”: it’s hard to find two inputs \( x, x' \) such that \( h(x) = h(x') \).
  - Often also constructed to have “randomness-like” properties.
    - Unpredictability of output when part of input is unknown.
    - “Pseudorandomness” and “independence” of input and output.
- Some modern hash functions are built by iterating a “compression” function.
Cryptographic Hash Functions

- Example: In MD5, $b = 512$ and $l = 128$
- Modern hash functions iterate this process

Keying Hash Functions

- Hash functions, as defined, have no keys
- We turn a hash function into a (keyed) function family by replacing the IV with a key
  - Let $f_K$ defined by $f_K(x) = f(K, x)$ be the keyed compression function, where $|K| = l$ and $|x| = b$
  - For any iterated hash construction, define a family $F$ as follows:
    - For $x = x_1 x_2 \ldots x_n$, define $F_K(x) = K_{n+1}$ where $K_i = f_K(x_i)$ for $i = 1 \ldots n+1$, $K_0 = K$, and $x_{n+1} = |x|$
Weak Collision Resistance

**Definition:** Let $F: \text{Keys}(F) \times \{0,1\}^* \rightarrow \{0,1\}^l$ be a family of keyed hash functions, and let $A$ be an adversary. Consider the following experiment:

Experiment $\text{Expt}_{F}^{wcr}(A)$

1. $K \leftarrow \text{R Keys}(F)$
2. $M, M' \leftarrow A^F_K$ (1)
3. If $M \neq M'$ and $F_K(M) = F_K(M')$
   then return 1 else return 0

The wcr-advantage of $A$ is

$$\text{Adv}_{F}^{wcr}(A) = \Pr \left[ \text{Expt}_{F}^{wcr}(A) = 1 \right]$$

Weak Collision Resistance

**For any** $t, q, \mu$, we define the wcr-advantage of $F$ as

$$\text{Adv}_{F}^{wcr}(t, q, \mu) = \max_{A} \left\{ \text{Adv}_{F}^{wcr}(A) \right\}$$

where the maximum is over all $A$ having time complexity $t$ and making at most $q$ oracle queries of total length $\mu$. 
NMAC

- Define the following “nested MAC” function where \( K = (K_1, K_2) \)

\[
\text{NMAC}_K(x) = F_{K_1}(F_{K_2}(x))
\]

- Proposition: Let \( f : \{0,1\}^l \times \{0,1\}^b \rightarrow \{0,1\}^l \) be a compression function family on messages of length \( b \) bits, and let \( F \) be its keyed iterated hash. Then

\[
\text{Adv}_{\text{NMAC}}^{\text{uf-cma}}(t, q, \mu) \leq \text{Adv}_f^{\text{uf-cma}}(t, q, q^b) + \text{Adv}_F^{\text{wcr}}(t, q, \mu)
\]

Proof: Let \( A \) be an NMAC attacker that runs in time \( t \) and makes \( q \) oracle queries of total length \( \mu \). Consider the attacker \( B_A \) for \( f \) as a MAC defined as follows.

- For a string \( s \) of length \( l \), let \( \langle s \rangle \) denote the result of \( s \) padded to a full block of length \( b \) as specified by the underlying hashing scheme.

\[
\begin{align*}
\text{Algorithm } B_A^{f_A}(\cdot) \\
K_2 &\leftarrow \text{Keys}(F) \\
\text{Run } A. \\
\text{For } i = 1 \ldots q \text{ do} \\
\text{When } A \text{ queries for } \text{NMAC}_i(M_i) \\
z &\leftarrow F_{K_2}(M_i) \\
\text{return } f_{K_1}(\langle z \rangle) \text{ to } A \\
\text{When } A \text{ outputs } (M, N), \text{ output } (\langle F_{K_2}(M) \rangle, N).
\end{align*}
\]
NMAC

Now we have that:

\[ 1 - \text{Adv}_{\text{uf-cma}}^{\text{NM}}(B_A) \]
\[ = \Pr[\text{Exp}_{f}^{\text{uf-cma}}(B_A) = 0] \]
\[ \leq \Pr[\text{Exp}_{\text{NM}}^{\text{uf-cma}}(A) = 0] + \Pr[\exists i : \langle F_{K_2}(M_i) \rangle = \langle F_{K_2}(M) \rangle] \]
\[ = \Pr[\text{Exp}_{\text{NM}}^{\text{uf-cma}}(A) = 0] + \Pr[\exists i : F_{K_2}(M_i) = F_{K_2}(M)] \]
\[ \leq (1 - \text{Adv}_{\text{NM}}^{\text{uf-cma}}(A)) + \text{Adv}_{\text{F}}^{\text{very}}(t, q, \mu) \]

HMAC

- NMAC is a very simple and efficient construction, but does not use hash function as a “black box”
  - Requires access to its compression function
- HMAC is an alternative that uses hash function completely as a “black box”
- HMAC is now a mandatory algorithm for most Internet security protocols
HMAC

- Let $F$ be a hash function (with normal IV)
- The HMAC construction is

$$
\text{HMAC}_K(M) = F(\langle K \rangle \oplus \text{opad} \ || \ F(\langle K \rangle \oplus \text{ipad} \ || \ M))
$$

where (in hexadecimal)
- opad = 36 36 ... 36
- ipad = 5c 5c ... 5c

- What’s the justification for this?

HMAC

- Let $f$ be the compression function of $F$
- If we define
  $$
  \begin{align*}
  K_1 &= f(\text{IV}, \langle K \rangle \oplus \text{opad}) \\
  K_2 &= f(\text{IV}, \langle K \rangle \oplus \text{ipad})
  \end{align*}
  $$
  then

$$
\text{HMAC}_K(M) = \text{NMAC}_{(K_1, K_2)}(M)
$$

- In other words, HMAC is a particular instance of NMAC, where $K_1$ and $K_2$ are “pseudorandomly” derived from $f$ and $K$
  - Strictly speaking, requires an additional assumption about pseudorandomness of $f$ when provided a key as an input
HMAC

- There might be attacks on HMAC but not NMAC, but this would reveal undesirable structural properties in $f$.

- opad and ipad were chosen
  - To be simple
  - To provide a high Hamming distance between themselves

Some MACs to Avoid

- “Append only” MACs
  - $MAC_K(M) = F(M, K)$ where $F$ is an iterated hash function
  - The problem: $F(M) = F(M')$, then for any $K$, $MAC_K(M) = MAC_K(M')$
  - Attack: Use the birthday paradox to find $M, M'$ offline
  - Question: Does $F$ appending $|M|$ help?

- “Prepend only” MACs
  - $MAC_K(M) = F(K, M)$ where $F$ is a hash function
  - The problem: If $M$ is an integral number of blocks and $MAC_K(M)$ is known, then $MAC_K(M||M')$ can be computed
  - Question: Does $F$ appending $|M|$ help?
MD5

- MD5 is an iterated hash function of the type anticipated for use in HMAC
- High-level structure
  - Appends padding bits (a “1” bit followed by as many “0” bits as needed) to input so that total length is 448 mod 512
  - Appends a 64-bit representation of the input length in bits (before padding); total length is now an integer multiple of 512 bits
  - A 128-bit buffer (four 32-bit words, labeled A, B, C, D) are initialized to fixed values
  - Each 512 bit block of the (padded, length-appended) input is passed through the compression function, updating the buffer
  - The buffer value at the end is the output value

Here, $H_{MD5}$ denotes the compression function for MD5
MD5 Weaknesses

- Berson 1992: There is an algorithm to find a collision for each of the four rounds individually in reasonable time
- Boer & Bosselaers 1993: There is an algorithm to find a message block on which execution of the MD5 compression function starting from two different values in ABCD will yield the same result
  - This is called a pseudocollision
- Dobbertin 1996: There is an algorithm to produce a collision on the MD5 compression function
- Wang & Yu 2005: Collisions on MD5 in under an hour
  - Attack works for any initial value

SHA-1

- Adopted by the National Institute of Standards and Technology (NIST) in 1995
- Algorithm takes as input a message of length at most $2^{64}$ bits
- Outputs a 160-bit value, processing inputs in 512-bit blocks
- High-level structure
  - Appends padding bits: Same as MD5
  - Appends a 64-bit representation of the input length: Same as MD5
  - A 160-bit buffer (five 32-bit words, labeled A, B, C, D, E) are initialized to fixed values
  - Each 512 bit block of the (padded, length-appended) input is passed through the compression function, updating the buffer
  - The buffer value at the end is the output value
SHA-1 Weaknesses

- Chabaud & Joux 1998: Collisions in full SHA-0 can be found in $\sim 2^{61}$ hash operations
- Biham & Joux 2005: Collisions in full SHA-0 can be found in $\sim 2^{51}$ hash operations
- Wang, Yin & Yu 2005: Collisions in the full SHA-1 can be found in $\sim 2^{69}$ hash operations
- Stevens 2012: Collisions in the full SHA-1 can be found in $\sim 2^{61}$ hash operations

Since SHA-1 ...

- There’s SHA-2, which features longer outputs
  - Variants have 256-bit or 512-bit outputs
  - No effective collision algorithms found yet
  - However, they are algorithmically similar to SHA-1, and so may not be secure for much longer
- SHA-3 was adopted in October 2012 after an open competition
  - 64 entries
  - 51 advanced to first round
  - 14 advanced to second round
  - 5 advanced to third round
  - Dissimilar to SHA-1,2
## Latest Generation of Hash Functions


<table>
<thead>
<tr>
<th>Algorithm and variant</th>
<th>Output size (bits)</th>
<th>Internal state size (bits)</th>
<th>Block size (bits)</th>
<th>Rounds</th>
<th>Operations against collision attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5 (as reference)</td>
<td>128</td>
<td>128 (4 × 32)</td>
<td>512</td>
<td>64</td>
<td>And, Or, Rot, Add (mod 2^9), Or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(collisions found)</td>
</tr>
<tr>
<td>SHA-0</td>
<td>160</td>
<td>160 (5 × 32)</td>
<td>512</td>
<td>80</td>
<td>And, Or, Rot, Add (mod 2^9), Or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(&lt;34 collisions found))</td>
</tr>
<tr>
<td>SHA-1</td>
<td>160</td>
<td>192 (6 × 32)</td>
<td>512</td>
<td>80</td>
<td>And, Or, Rot, Add (mod 2^9), Or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(&lt;63 collisions found))</td>
</tr>
<tr>
<td>SHA-2</td>
<td>SHA-224</td>
<td>224</td>
<td>256</td>
<td>64</td>
<td>And, Or, Rot, Add (mod 2^9), Or</td>
</tr>
<tr>
<td></td>
<td>SHA-256</td>
<td>256</td>
<td>256</td>
<td>64</td>
<td>Or, Rot</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(112)</td>
</tr>
<tr>
<td></td>
<td>SHA-384</td>
<td>384</td>
<td>512</td>
<td>1024</td>
<td>And, Or, Rot, Add (mod 2^9), Or</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>512</td>
<td>512</td>
<td>1024</td>
<td>Or, Rot</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(102)</td>
</tr>
<tr>
<td></td>
<td>SHA-512/256</td>
<td>224</td>
<td>256</td>
<td>64</td>
<td>And, Or, Rot, Add (mod 2^9), Or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Or, Rot, Shr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(112)</td>
</tr>
<tr>
<td>SHA-3</td>
<td>SHA-224</td>
<td>224</td>
<td>256</td>
<td>64</td>
<td>And, Or, Rot, Add (mod 2^9), Or</td>
</tr>
<tr>
<td></td>
<td>SHA-256</td>
<td>256</td>
<td>256</td>
<td>64</td>
<td>Or, Rot</td>
</tr>
<tr>
<td></td>
<td>SHA-384</td>
<td>384</td>
<td>512</td>
<td>1024</td>
<td>And, Or, Rot, Add (mod 2^9), Or</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>512</td>
<td>512</td>
<td>1024</td>
<td>Or, Rot, Shr</td>
</tr>
<tr>
<td></td>
<td>SHA-512/256</td>
<td>224</td>
<td>256</td>
<td>64</td>
<td>And, Or, Rot, Add (mod 2^9), Or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Or, Rot, Shr</td>
</tr>
<tr>
<td></td>
<td>SHA-512/256</td>
<td>224</td>
<td>256</td>
<td>64</td>
<td>And, Or, Rot, Add (mod 2^9), Or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Or, Rot, Shr</td>
</tr>
</tbody>
</table>

## Combining Encryption and Authentication

- We’ve now seen secure encryption and security message authentication (via message authentication codes)
- If you want to do both, then you’ll need to make two passes over the data, which is expensive
- A popular alternative today is to achieve encryption and authentication with a single primitive
  - and, notably, a single pass over the data
- The most widely used such mechanism is Galois/Counter Mode (GCM)
Inputs and Outputs of GCM Encryption

- **Inputs to the GCM encryption algorithm**
  - A secret key $K$ for use with an underlying block cipher
  - An initialization vector $IV$ of length between 1 and $2^{64}$
    - For a fixed key $K$, each $IV$ used in encryption must be distinct
  - A plaintext $P$ of length up to $2^{39} - 256$
  - Additional data $A$ to be authenticated (but not encrypted), of length up to $2^{64}$

- **Outputs from the GCM encryption algorithm**
  - A ciphertext $C$ of length the same as $P$
  - An authentication tag $T$ of length $t$, $0 \leq t \leq 128$

- **Note:** If $|P| = 0$, then GCM is just a MAC on $A$, called “GMAC”

Inputs and Outputs of GCM Decryption

- **GCM decryption algorithm takes as inputs**
  - Secret key $K$
  - Initialization vector $IV$
  - Ciphertext $C$
  - Authenticated data $A$
  - Authentication tag $T$

- **GCM decryption algorithm outputs**
  - Plaintext $P$ or
  - A failure symbol $\perp$

- **If** $(C, T) \leftarrow E_K(IV, P, A)$ then $P \leftarrow D_K(IV, C, A, T)$
GHASH

- Built using a function called $\text{GHASH}(H, A, C)$
- Suppose $H \in \{0, 1\}^{128}$ and $A \in \{0, 1\}^{128m}$ and $C \in \{0, 1\}^{128n}$
  - Let $A = A_1 A_2 \ldots A_m$ and $C = C_1 C_2 \ldots C_n$ with each $A_i, C_i \in \{0, 1\}^{128}$
- Let $\times$ and $\oplus$ denote multiplication and addition in $GF(2^{128})$
- $\text{GHASH}(H, A, C)$ returns $X_{m+n+1}$ where

\[
X_i = \begin{cases} 
0 & \text{for } i = 0 \\
(X_{i-1} \oplus A_i) \times H & \text{for } i = 1 \ldots m \\
(X_{i-1} \oplus C_{i-m}) \times H & \text{for } i = m + 1 \ldots n \\
(X_{m+n} \oplus (\text{len}(A) || \text{len}(C))) \times H & \text{for } i = m + n + 1
\end{cases}
\]

GCM Encryption Algorithm

Algorithm $E_K(IV, P, A)$

\[
\begin{align*}
H & \leftarrow F_K(0^{128}) \\
Y_0 & \leftarrow \text{GHASH}(H, \emptyset, IV) \\
\text{for } i & \leftarrow 1 \ldots n \\
C_i & \leftarrow P_i \oplus F_K(Y_0 + i) \\
C & \leftarrow C_1 || C_2 || \ldots || C_n \\
T & \leftarrow \text{MSB}_t(\text{GHASH}(H, A, C) \oplus F_K(Y_0)) \\
\text{return } (C, T)
\end{align*}
\]
GCM Encryption Algorithm

GCM Decryption Algorithm
GCM Security

- Both secrecy and authenticity guarantees can be reduced to PRF security of the underlying function family $F$.
- Paper containing proof is posted to the web page.