{ CSCI 6331 · 4331 | Lecture 6 }

Cryptography

Hoeteck Wee · hoeteck@gwu.edu

http://tinyurl.com/cryptogw/

Announcements

- Evaluation:

  10% In-Class/Piazza, 20% Final Presentation / Project

  30% Homework, 40% Final (Apr 25)

- Homework 3 is out

  due Feb 29 (Wed) in class

Message Authentication Codes

setting.

- both users generate and share a *secret* key k in advance

— runs key generation algorithm $\mathrm{k} \leftarrow \mathrm{Gen}(1^{\mathrm{n}})$

- to send message m, sender computes a MAC tag t and sends (m, t)

— runs tag generation algorithm $\mathrm{t} \leftarrow \mathrm{Mac}_{\mathrm{k}}(\mathrm{m})$

- upon receiving (m, t), receiver verifies whether t is a valid tag on m

— runs verification algorithm $\mathrm{Vrfy}(\mathrm{m}, \mathrm{t}) \in \{0, 1\}$ (1 being valid)

Message Authentication Codes

setting.

- both users generate and share a *secret* key k in advance
— runs key generation algorithm $\mathrm{k} \leftarrow \mathrm{Gen}(1^{\mathrm{n}})$
- to send message m, sender computes a MAC tag t and sends (m, t)
— runs tag generation algorithm $\mathrm{t} \leftarrow \mathrm{Mac}_{\mathrm{k}}(\mathrm{m})$
- upon receiving (m, t), receiver verifies whether t is a valid tag on m
— runs verification algorithm $\mathrm{Vrfy}(\mathrm{m}, \mathrm{t}) \in \{0, 1\}$ (1 being valid)

syntax. *message authentication code* (MAC) is a triple of randomized algorithms $(\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrfy})$

- correctness. for every key k output by $\mathrm{Gen}(1^{\mathrm{n}})$, and every $\mathrm{m} \in \{0, 1\}^*$, we have $\mathrm{Vrfy}_{\mathrm{k}}(\mathrm{m}, \mathrm{Mac}_{\mathrm{k}}(\mathrm{m})) = 1$.

Message Authentication Codes

Security Definition. hard to generate a valid tag on any "new" message that was not previously sent – existentially unforgeable under adaptive chosen-message attack

Security Definition. hard to generate a valid tag on any "new" message that was not previously sent – existentially unforgeable under adaptive chosen-message attack

1. Generate random key k using $\mathrm{Gen}(1^{\mathrm{n}})$
2. Adversary given 1^{n} and oracle access to $\mathrm{Mac}_{\mathrm{k}}(\cdot)$, eventually outputs (m, t). Let Q = set of queries
3. Wins if $\mathrm{Vrfy}_{\mathrm{k}}(\mathrm{m}, \mathrm{t}) = 1$ and $\mathrm{m} \notin \mathrm{Q}$.

definition. (t, ϵ)-secure if for all advesaries running in time t, winning probability bounded by ϵ.

Message Authentication Codes from PRFs

Security Definition. hard to generate a valid tag on any "new" message that was not previously sent – existentially unforgeable under adaptive chosen-message attack

1. Gen : choose random $\mathrm{k} \leftarrow \mathrm{K}$
2. $\mathrm{Mac}_{\mathrm{k}}(\mathrm{m})$: output tag $\mathrm{F}(\mathrm{k}, \mathrm{m})$
3. $\mathrm{Vrfy}_{\mathrm{k}}(\mathrm{m}, \mathrm{t})$: output 1 iff $\mathrm{t} = \mathrm{F}(\mathrm{k}, \mathrm{m})$

- important distinction: *fixed* vs *variable*-length messages
- fixed: given MAC("hello"), MAC("world"), hard to compute MAC("wello"); however, computing MAC("hello world") may be easy.

Message Authentication Codes from PRFs

Security Definition. hard to generate a valid tag on any "new" message that was not previously sent – existentially unforgeable under adaptive chosen-message attack

1. Gen : choose random $\mathrm{k} \leftarrow \mathrm{K}$
2. $\mathrm{Mac_k(m)}$: output tag $\mathrm{F(k, m)}$
3. $\mathrm{Vrfy_k(m, t)}$: output 1 iff $\mathrm{t = F(k, m)}$

- important distinction: *fixed* vs *variable*-length messages
- fixed: given MAC("hello"), MAC("world"), hard to compute MAC("wello"); however, computing MAC("hello world") may be easy.

Note. Above construction only works for "short" messages of a fixed length.

Message Authentication Codes from PRFs

Security Definition. hard to generate a valid tag on any "new" message that was not previously sent – existentially unforgeable under adaptive chosen-message attack

1. Gen : choose random $\mathrm{k} \leftarrow \mathrm{K}$
2. $\mathrm{Mac_k}(\mathrm{m})$: output tag $\mathrm{F}(\mathrm{k}, \mathrm{m})$
3. $\mathrm{Vrfy_k}(\mathrm{m}, \mathrm{t})$: output 1 iff $\mathrm{t} = \mathrm{F}(\mathrm{k}, \mathrm{m})$

- important distinction: *fixed* vs *variable*-length messages
- fixed: given MAC("hello"), MAC("world"), hard to compute MAC("wello"); however, computing MAC("hello world") may be easy.

Note. Above construction only works for "short" messages of a fixed length.

— solution 1: CBC-MAC (variant of CBC-mode encryption)

— solution 2: using collision-resistant hash functions

Message Authentication Codes from PRFs

Security Definition. hard to generate a valid tag on any "new" message that was not previously sent – existentially unforgeable under adaptive chosen-message attack

1. Gen : choose random $\mathrm{k} \leftarrow \mathrm{K}$
2. $\mathrm{Mac_k(m)}$: output tag $\mathrm{F(k, m)}$
3. $\mathrm{Vrfy_k(m, t)}$: output 1 iff $\mathrm{t} = \mathrm{F(k, m)}$

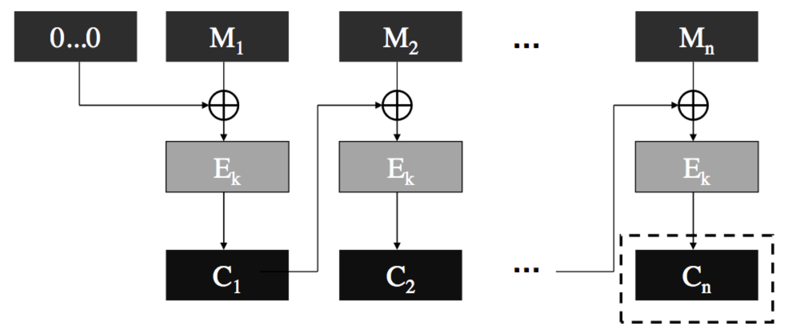
- important distinction: *fixed* vs *variable*-length messages
- fixed: given MAC("hello"), MAC("world"), hard to compute MAC("wello"); however, computing MAC("hello world") may be easy.

Note. Above construction only works for "short" messages of a fixed length.

— solution 1: CBC-MAC (variant of CBC-mode encryption)

— solution 2: using collision-resistant hash functions

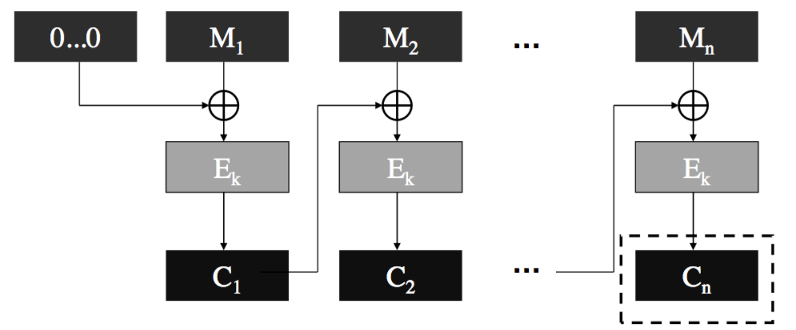
Basic CBC-MAC



comparison with CBC-mode encryption.

- always use $\mathrm{IV} = 00\ldots0$ (or, no IV); CBC-mode encryption uses *random* IV.
- only output final block $\mathrm{C_n}$; CBC-mode encryption outputs immediate blocks
- proof idea: show all inputs to PRF are distinct with high probability

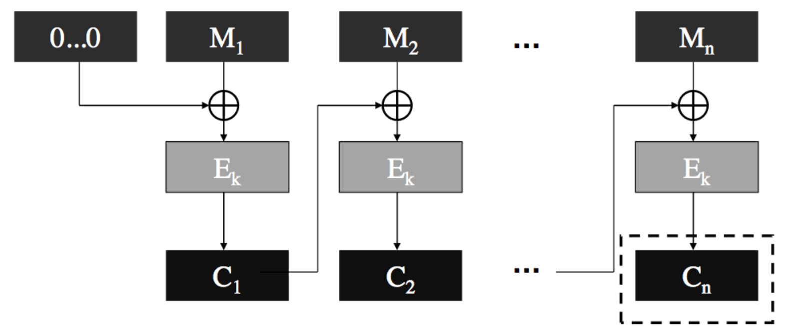
Basic CBC-MAC



comparison with CBC-mode encryption.

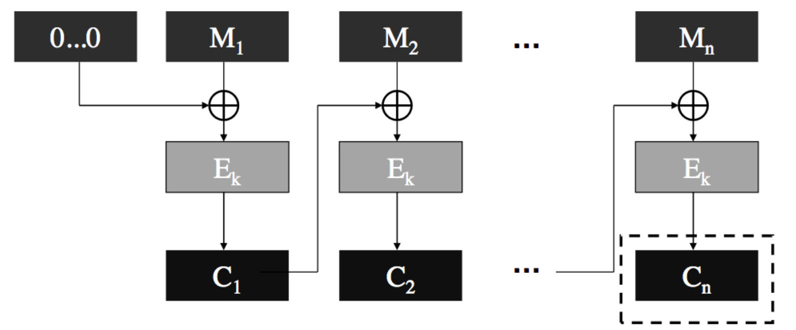
- always use $\mathrm{IV} = 00\ldots0$ (or, no IV); CBC-mode encryption uses *random* IV.
- only output final block $\mathrm{C_n}$; CBC-mode encryption outputs immediate blocks
- proof idea: show all inputs to PRF are distinct with high probability
- important distinction: many cryptography libraries provide a "CBC function"

Basic CBC-MAC



fixed-length. given tags for 5-letter words, hard to forge tag on new 5-letter word.

Basic CBC-MAC

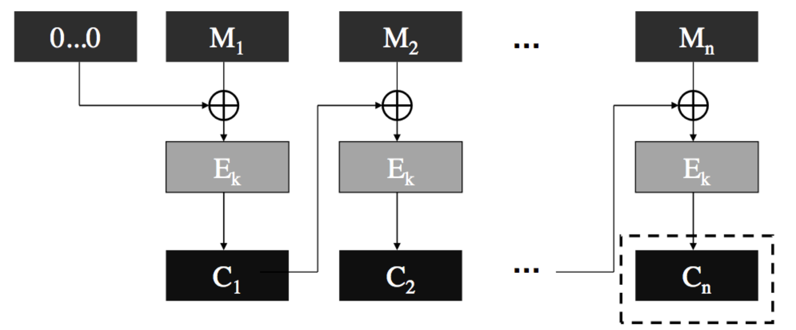


fixed-length. given tags for 5-letter words, hard to forge tag on new 5-letter word.

extension attack. given MAC("hello") = "aydmx", forge MAC("hello world")?

— does getting MAC("world") help?

Basic CBC-MAC



fixed-length. given tags for 5-letter words, hard to forge tag on new 5-letter word.

extension attack. given MAC("hello") = "aydmx", forge MAC("hello world")?

- does getting MAC("world") help?
- how about getting MAC("adymx")?

"Extended" CBC-MAC

Handling variable-length messages.

- Method 1. Apply PRF to $|m|$ to obtain k'. Compute basic CBC-MAC using k'.
  - ensures different keys are used to authenticate messages of different lengths

“Extended” CBC-MAC

Handling variable-length messages.

- Method 1. Apply PRF to $|m|$ to obtain k'. Compute basic CBC-MAC using k'.
  - ensures different keys are used to authenticate messages of different lengths
- Method 2. Prepend message with $|m|$, encoded as n-bit string, compute basic CBC-MAC on resulting message
  - appending length to *end* of message is *not* secure.

"Extended" CBC-MAC

Handling variable-length messages.

- Method 1. Apply PRF to $|m|$ to obtain k'. Compute basic CBC-MAC using k'.
  - ensures different keys are used to authenticate messages of different lengths
- Method 2. Prepend message with $|m|$, encoded as n-bit string, compute basic CBC-MAC on resulting message
  - appending length to *end* of message is *not* secure.
- Method 3. choose two different keys (k_1, k_2) as MAC key. Let t := basic CBC-MAC on m using k_1; output tag $\hat{t} = F_{k_2}(t)$
  - advantage: can be used for streaming data with unknown length

Collision-Resistant Hash Functions

"hash functions" used in data structures, e.g. $\mathrm{H} : \{0,1\}^* \to \{0,1\}^{128}$

- take arbitrary-length strings and *compress* them into shorter strings
- given $(\mathrm{name}, \mathrm{record})$, store record in cell $\mathrm{H}(\mathrm{name})$
- easy to store and look up record given name

Collision-Resistant Hash Functions

"hash functions" used in data structures, e.g. $\mathrm{H} : \{0,1\}^* \to \{0,1\}^{128}$

- take arbitrary-length strings and *compress* them into shorter strings
- given $(\mathrm{name}, \mathrm{record})$, store record in cell $\mathrm{H}(\mathrm{name})$
- easy to store and look up record given name
- "good" hash function avoids collisions: $\mathrm{x} \neq \mathrm{x}'$ but $\mathrm{H}(\mathrm{x}) = \mathrm{H}(\mathrm{x}')$

Collision-Resistant Hash Functions

"hash functions" used in data structures, e.g. $\mathrm{H} : \{0,1\}^* \rightarrow \{0,1\}^{128}$

- take arbitrary-length strings and *compress* them into shorter strings
- given $(\mathrm{name}, \mathrm{record})$, store record in cell $\mathrm{H}(\mathrm{name})$
- easy to store and look up record given name
- "good" hash function avoids collisions: $\mathrm{x} \neq \mathrm{x}'$ but $\mathrm{H}(\mathrm{x}) = \mathrm{H}(\mathrm{x}')$

collision-resistant hash functions used in cryptography

- mandatory (for security purposes) to avoid collisions
- e.g. hash homework submission / individuals to unique fingerprint?
- examples: $\mathrm{MD5} : \{0,1\}^* \rightarrow \{0,1\}^{128}$, $\mathrm{SHA1} : \{0,1\}^* \rightarrow \{0,1\}^{160}$

Collision-Resistant Hash Functions

"hash functions" used in data structures, e.g. $\mathrm{H} : \{0,1\}^* \to \{0,1\}^{128}$

- take arbitrary-length strings and *compress* them into shorter strings
- given $(\mathrm{name}, \mathrm{record})$, store record in cell $\mathrm{H}(\mathrm{name})$
- easy to store and look up record given name
- "good" hash function avoids collisions: $\mathrm{x} \neq \mathrm{x}'$ but $\mathrm{H}(\mathrm{x}) = \mathrm{H}(\mathrm{x}')$

collision-resistant hash functions used in cryptography

- mandatory (for security purposes) to avoid collisions
- e.g. hash homework submission / individuals to unique fingerprint?
- examples: $\mathrm{MD5} : \{0,1\}^* \to \{0,1\}^{128}$, $\mathrm{SHA1} : \{0,1\}^* \to \{0,1\}^{160}$
- H is collison-resistant if it is infeasible to find collision in H
- only interested in H with input length $>$ output length
- MAC for variable-length message — hash-then-MAC

Collision-Resistant Hash Functions

security definition. H is (t, ϵ)-collision-resistant if for all A running in time t, probability A outputs a collision, i.e. $\mathrm{x} \neq \mathrm{x}'$ but $\mathrm{H}(\mathrm{x}) = \mathrm{H}(\mathrm{x}')$ is $< \epsilon$

Collision-Resistant Hash Functions

security definition. H is (t, ϵ)-collision-resistant if for all A running in time t, probability A outputs a collision, i.e. $\mathrm{x} \neq \mathrm{x}'$ but $\mathrm{H}(\mathrm{x}) = \mathrm{H}(\mathrm{x}')$ is $< \epsilon$

- formally, *family* of hash functions $\mathrm{H}^{\mathrm{s}}(\cdot)$ indexed by key s
- hard to find collisions in H^{s} for a randomly-generated s, adversary sees s

Collision-Resistant Hash Functions

security definition. H is (t, ϵ)-collision-resistant if for all A running in time t, probability A outputs a collision, i.e. $x \neq x'$ but $H(x) = H(x')$ is $< \epsilon$

- formally, *family* of hash functions $H^s(\cdot)$ indexed by key s
- hard to find collisions in H^s for a randomly-generated s, adversary sees s
- practice: key is initialization vector, e.g. `h0 := 0x67452301` in $\mathrm{MD5}$, etc.

Collision-Resistant Hash Functions

security definition. H is (t, ϵ)-collision-resistant if for all A running in time t, probability A outputs a collision, i.e. $\mathrm{x} \neq \mathrm{x}'$ but $\mathrm{H}(\mathrm{x}) = \mathrm{H}(\mathrm{x}')$ is $< \epsilon$

- formally, *family* of hash functions $\mathrm{H}^{\mathrm{s}}(\cdot)$ indexed by key s
- hard to find collisions in H^{s} for a randomly-generated s, adversary sees s
- practice: key is initialization vector, e.g. `h0 := 0x67452301` in $\mathrm{MD5}$, etc.

generic "birthday" attack. on $\mathrm{H} : \{0,1\}^* \to \{0,1\}^\ell$

- if there are > 366 people in the room, certainly two have same birthday

Collision-Resistant Hash Functions

security definition. H is (t, ϵ)-collision-resistant if for all A running in time t, probability A outputs a collision, i.e. $\mathrm{x} \neq \mathrm{x}'$ but $\mathrm{H}(\mathrm{x}) = \mathrm{H}(\mathrm{x}')$ is $< \epsilon$

- formally, *family* of hash functions $\mathrm{H}^{\mathrm{s}}(\cdot)$ indexed by key s
- hard to find collisions in H^{s} for a randomly-generated s, adversary sees s
- practice: key is initialization vector, e.g. `h0 := 0x67452301` in $\mathrm{MD5}$, etc.

generic "birthday" attack. on $\mathrm{H} : \{0,1\}^* \to \{0,1\}^\ell$

- if there are > 366 people in the room, certainly two have same birthday
- claim. if there are > 23 people in the room, good chance two have same birthday

Collision-Resistant Hash Functions

security definition. H is (t, ϵ)-collision-resistant if for all A running in time t, probability A outputs a collision, i.e. $\mathrm{x} \neq \mathrm{x}'$ but $\mathrm{H}(\mathrm{x}) = \mathrm{H}(\mathrm{x}')$ is $< \epsilon$

- formally, *family* of hash functions $\mathrm{H}^{\mathrm{s}}(\cdot)$ indexed by key s
- hard to find collisions in H^{s} for a randomly-generated s, adversary sees s
- practice: key is initialization vector, e.g. `h0 := 0x67452301` in $\mathrm{MD5}$, etc.

generic "birthday" attack. on $\mathrm{H} : \{0,1\}^* \to \{0,1\}^\ell$

- if there are > 366 people in the room, certainly two have same birthday
- claim. if there are > 23 people in the room, good chance two have same birthday
- brute force. try $2^\ell + 1$ different inputs
- lemma. pick $\approx 2^{\ell/2}$ random inputs, good chance of finding collision

Collision-Resistant Hash Functions

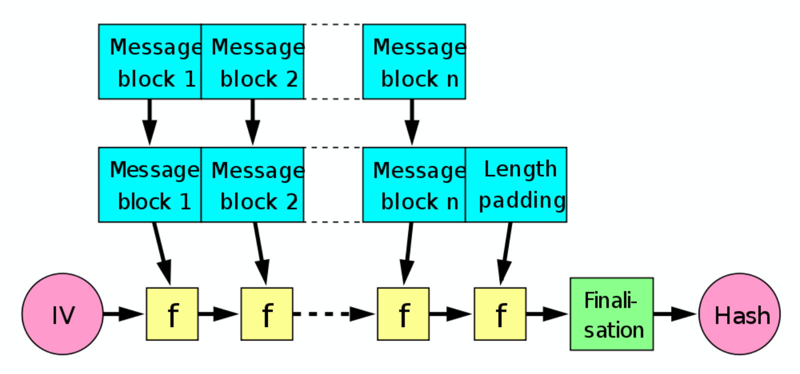
security definition. H is (t, ϵ)-collision-resistant if for all A running in time t, probability A outputs a collision, i.e. $\mathrm{x} \neq \mathrm{x}'$ but $\mathrm{H}(\mathrm{x}) = \mathrm{H}(\mathrm{x}')$ is $< \epsilon$

- formally, *family* of hash functions $\mathrm{H}^{\mathrm{s}}(\cdot)$ indexed by key s
- hard to find collisions in H^{s} for a randomly-generated s, adversary sees s
- practice: key is initialization vector, e.g. `h0 := 0x67452301` in $\mathrm{MD5}$, etc.

generic "birthday" attack. on $\mathrm{H} : \{0,1\}^* \to \{0,1\}^\ell$

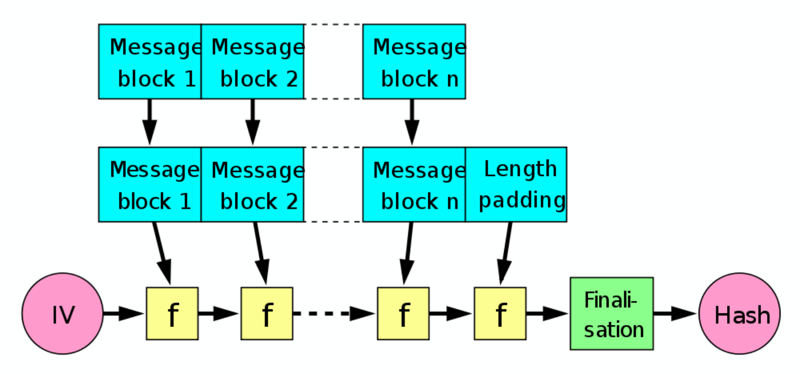
- if there are > 366 people in the room, certainly two have same birthday
- claim. if there are > 23 people in the room, good chance two have same birthday
- brute force. try $2^\ell + 1$ different inputs
- lemma. pick $\approx 2^{\ell/2}$ random inputs, good chance of finding collision
- examples: $\mathrm{MD5} : \{0,1\}^* \to \{0,1\}^{128}$ broken with 2^{64} computations; $\mathrm{SHA1} : \{0,1\}^* \to \{0,1\}^{160}$ broken with 2^{80} computations

Merkle-Damgård transform



Q. How to hash long messages starting from $\mathrm{H} : \{0,1\}^{256} \to \{0,1\}^{128}$?

Merkle-Damgård transform



Q. How to hash long messages starting from $H : \{0,1\}^{256} \to \{0,1\}^{128}$?

— intuition: if two strings x, x' collide, then there must be distinct intermediate values that collide