Cryptography

Lecture 8: Message Authentication Codes

December 3, 2024

Overview

Contents

1. Classical cryptography
   (Shift & Vigenère cipher, one-time pad, perfect secrecy)
2. Security definitions & threat models
   (Computational security, CPA & CCA)
3. Private-key cryptography
   (Message authentication, hash functions, primitives, relevant ciphers)
4. Public-key cryptography
   (Assumptions, key management, digital signatures, relevant ciphers)

Cryptographic Export Restrictions

Timeline (United States)

- 1949 Export Control Act prohibits export of cryptography
  (one-by-one solutions later usually coordinated by IBM)
- 1969 Export Administration Regulations streamline process
  availability outside US considered; limitation to 40 bit key
  (however cryptography still covered by U.S. Munitions List)
- 1996 Executive Order 13026 removal from U.S. Munitions List
- 1998 Limitation to 56 bit key with backdoor or 40 bit key as before
- 1999 Universal limitation to 56 bit key (1,024 bit key for RSA)
- 2000 Removal of length restrictions for retail & open-source software
- 2021 License requirement exceptions for most uses & destinations
  (certain registration requirements remain)

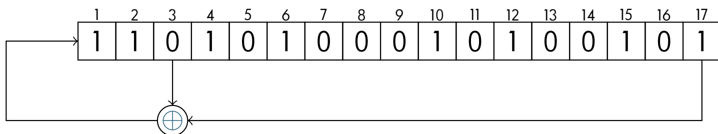
Proposals for Pseudorandom Generators — CSS

CSS

- Stream cipher “Content Scramble System”
- Used to encrypt DVD-Video contents; developed by DVD Forum
- Broken since 1999 by brute-force (takes seconds on modern CPU)
- Utilizes standard construction, so we only describe generator
- Uses 2 LFSR (linear feedback shift registers)

17-bit LFSR with taps 3 and 17

- Shifts right each clock cycle; first bit filled by XORed taps



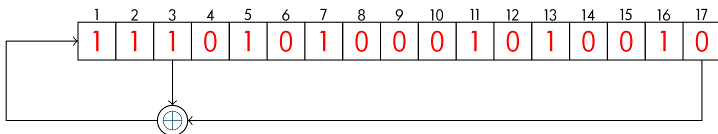
Proposals for Pseudorandom Generators — CSS

CSS

- Stream cipher "Content Scramble System"
- Used to encrypt DVD-Video contents; developed by DVD Forum
- Broken since 1999 by brute-force (takes seconds on modern CPU)
- Utilizes standard construction, so we only describe generator
- Uses 2 LFSR (linear feedback shift registers)

17-bit LFSR with taps 3 and 17

- Shifts right each clock cycle; first bit filled by XORed taps



Proposals for Pseudorandom Generators — CSS

Characteristics of CSS

- No initial vector
- States are 45 bits (17 bit LFSR + 25 bit LFSR + 2 bit inverters + 1 bit carry)
- Seed length 40 bits
- Generates 1 bit per round

Initialization with seed $s = (s_1, \ldots, s_{40})$ and operation mode (i_1, i_2)

1. $r_1 = (r_1[1], \ldots, r_1[17]) \leftarrow (s_1, \ldots, s_8, 1, s_9, \ldots, s_{16})$ and
   $r_2 = (r_2[1], \ldots, r_2[25]) \leftarrow (s_{17}, \ldots, s_{19}, 1, s_{20}, \ldots, s_{40})$
2. Return $(r_1, r_2, i_1, i_2, 0)$

s: 1 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
r_1	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
r_2	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Proposals for Pseudorandom Generators — CSS

Characteristics of CSS

- No initial vector
- States are 45 bits (17 bit LFSR + 25 bit LFSR + 2 bit inverters + 1 bit carry)
- Seed length 40 bits
- Generates 1 bit per round

Initialization with seed $s = (s_1, \ldots, s_{40})$ and operation mode (i_1, i_2)

1. $r_1 = (r_1[1], \ldots, r_1[17]) \leftarrow (s_1, \ldots, s_8, 1, s_9, \ldots, s_{16})$ and
   $r_2 = (r_2[1], \ldots, r_2[25]) \leftarrow (s_{17}, \ldots, s_{19}, 1, s_{20}, \ldots, s_{40})$
2. Return $(r_1, r_2, i_1, i_2, 0)$

s	1	1	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	1	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	0	1	0	0	0	1

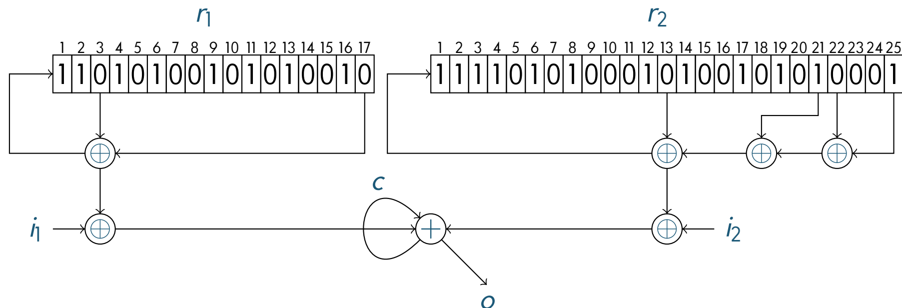
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
r_1	1	1	0	1	0	1	0	0	1	0	1	0	1	0	0	1	0

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
r_2	1	1	1	1	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	0	1	0	0	0	1

Proposals for Pseudorandom Generators — CSS

Next bit generation with state (r_1, r_2, i_1, i_2, c)

1. $b_1 \leftarrow r_1[3] \oplus r_1[17]$ and $b_2 \leftarrow r_2[13] \oplus r_2[21] \oplus r_2[22] \oplus r_2[25]$
2. $o_1 \leftarrow b_1 \oplus i_1$ and $o_2 \leftarrow b_2 \oplus i_2$
3. $(c, o) \leftarrow o_1 + o_2 + c$ (regular 1-bit addition with carry)
4. Shift registers r_1 and r_2 filling with b_1 and b_2
5. Return output bit o and state (r_1, r_2, i_1, i_2, c)



Proposals for Pseudorandom Generators — CSS

Standard attack

- Brute-force attack against smaller register r_1
  using known initial segment of message (MPEG header $\approx$ 20 bytes)
  ($2^{16} = 65,536$ possibilities)
- Seed length fixed, so no increase of security possible

ChaCha

- Stream cipher published in 2008 as successor to Salsa20
- Public domain
- Used by Google in TLS, standard random in BSD-type systems
  (Generator for `/dev/urandom` in Linux since kernel 4.8)
- No known sensible attacks
  (best known attack requires $\approx 2^{250}$ operations against 256 bit key)
- Uses only shifts, XORs, and addition

Daniel J. Bernstein (* 1971)

- US-German cryptographer & mathematician
- Professor at Eindhoven University of Technology
- Developed Salsa20, ChaCha, Curve25519, Poly1305

© Alexander Klink

Proposals for Pseudorandom Generators — ChaCha

Characteristics of ChaCha (word = 32 bits)

- 4 word = 128 bit initial vector (2 word nonce & 2 word counter)
- States are 16 words = 512 bits
- Seed length 8 words = 256 bits
- Generates 16 words = 512 bits per round

Proposals for Pseudorandom Generators — ChaCha

Initialization with seed s and initial vector v

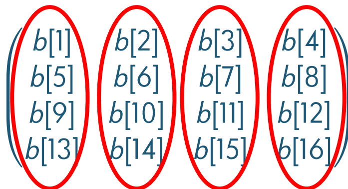
1. $b[1] \cdots b[4] \leftarrow$ "expand 32-byte k" (4 words)
2. $b[5] \cdots b[12] \leftarrow s$ (8 words)
3. $b[13] \cdots b[16] \leftarrow v$ (4 words)
4. return b

$$\begin{pmatrix} b[1] & b[2] & b[3] & b[4] \\ b[5] & b[6] & b[7] & b[8] \\ b[9] & b[10] & b[11] & b[12] \\ b[13] & b[14] & b[15] & b[16] \end{pmatrix} = \begin{pmatrix} \text{"expa"} & \text{"nd 3"} & \text{"2-by"} & \text{"te k"} \\ s[1] & s[2] & s[3] & s[4] \\ s[5] & s[6] & s[7] & s[8] \\ v[1] & v[2] & v[3] & v[4] \end{pmatrix}$$

Proposals for Pseudorandom Generators — ChaCha

Next 16 word generation with state b

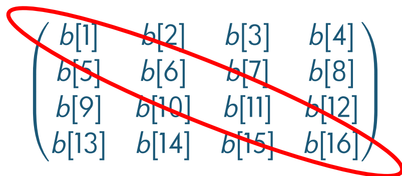
1. $o \leftarrow b$
2. for $i \in \{1, \ldots, 10\}$
   1. quart($b[1], b[5], b[9], b[13]$) ; quart($b[2], b[6], b[10], b[14]$) (columns)
   2. quart($b[3], b[7], b[11], b[15]$) ; quart($b[4], b[8], b[12], b[16]$) (columns)
   3. quart($b[1], b[6], b[11], b[16]$) ; quart($b[2], b[7], b[12], b[13]$) (diagonals)
   4. quart($b[3], b[8], b[9], b[14]$) ; quart($b[4], b[5], b[10], b[15]$) (diagonals)
3. $o[i] \leftarrow \text{Add}(o[i], b[i])$ for all $1 \leq i \leq 16$
4. $b[13]b[14] \leftarrow \text{Add}(b[13]b[14], 1)$ (increase counter)
5. Return output o and state b



Proposals for Pseudorandom Generators — ChaCha

Next 16 word generation with state b

1. $o \leftarrow b$
2. for $i \in \{1, \ldots, 10\}$
   1. quart($b[1], b[5], b[9], b[13]$) ; quart($b[2], b[6], b[10], b[14]$) (columns)
   2. quart($b[3], b[7], b[11], b[15]$) ; quart($b[4], b[8], b[12], b[16]$) (columns)
   3. quart($b[1], b[6], b[11], b[16]$) ; quart($b[2], b[7], b[12], b[13]$) (diagonals)
   4. quart($b[3], b[8], b[9], b[14]$) ; quart($b[4], b[5], b[10], b[15]$) (diagonals)
3. $o[i] \leftarrow \text{Add}(o[i], b[i])$ for all $1 \leq i \leq 16$
4. $b[13]b[14] \leftarrow \text{Add}(b[13]b[14], 1)$ (increase counter)
5. Return output o and state b



Proposals for Pseudorandom Generators — ChaCha

Quarter round $\mathsf{quart}(x, y, z, w)$

1. $x \leftarrow \mathsf{Add}(x, y)$; $w \leftarrow \mathsf{XOR}(w, x)$; $w \leftarrow \mathsf{RotL}(w, 16)$
2. $z \leftarrow \mathsf{Add}(z, w)$; $y \leftarrow \mathsf{XOR}(y, z)$; $y \leftarrow \mathsf{RotL}(y, 12)$
3. $x \leftarrow \mathsf{Add}(x, y)$; $w \leftarrow \mathsf{XOR}(w, x)$; $w \leftarrow \mathsf{RotL}(w, 8)$
4. $z \leftarrow \mathsf{Add}(z, w)$; $y \leftarrow \mathsf{XOR}(y, z)$; $y \leftarrow \mathsf{RotL}(y, 7)$

Message Authentication Codes (MACs)

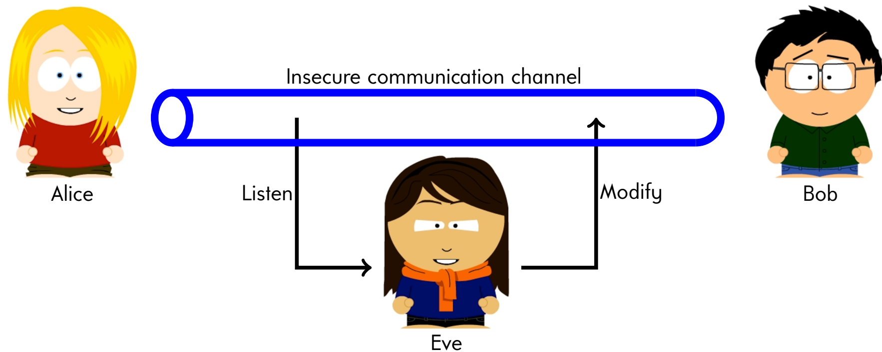
Motivation

- Ensure integrity (instead of secrecy) of transmitted messages (detect changes in message & authenticate origin)
- Integrity $\neq$ secrecy
  (change ciphertext to modify received message in one-time pad)
- Integrity provided by standard error-correcting codes insufficient
  (different error-profile: random bit flips vs. malicious adversary)

Applications

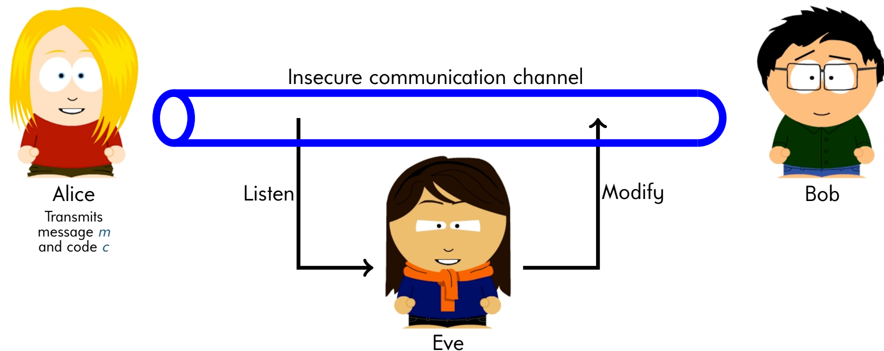
- Signed code
- Banner ad & cookie verification

Message Authentication Codes



Notes

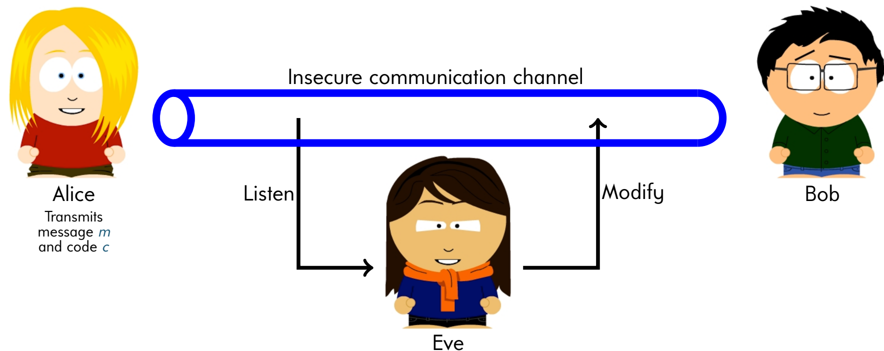
Message Authentication Codes



Notes

- Alice sends message together with code (tag)

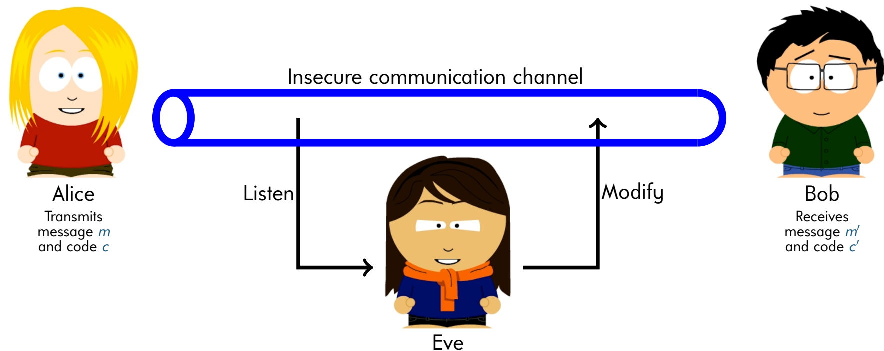
Message Authentication Codes



Notes

- Alice sends message together with code (tag)
- Eve might intercept & modify message & code

Message Authentication Codes



Notes

- Alice sends message together with code (tag)
- Eve might intercept & modify message & code
- Bob receives potentially different message & code
  (verifies whether they are consistent)

Message Authentication Codes

§7.2 Definition (cf. §3.6; Efficient message authentication scheme)

(Efficient) message authentication scheme is triple $(\mathrm{gen}, \mathrm{mac}, \mathrm{val})$ of probabilistic polynomial-time computable functions

- $\mathrm{gen}\colon R \times \{1\}^* \to K$ i.e. family $(\mathrm{P}_K^n)_{n \in \mathbb{N}}$ of distributions $\mathrm{P}_K^n\colon K \to [0,1]$
- $\mathrm{mac}\colon R \times K \times M \to C$ and $\mathrm{val}\colon K \times M \times C \to \{0,1\}$
- $|\mathrm{gen}(r, 1^n)| \geq n$ for all $r \in R$ and $n \in \mathbb{N}$
- $\mathrm{val}_k(m, c) = 1$ for all $k \in \mathrm{ran}(\mathrm{gen})$, $m \in M$, and $c \in C$ with $\mathrm{P}\big[c \leftarrow \mathrm{mac}_k(m)\big] \neq 0$ (perfect correctness)

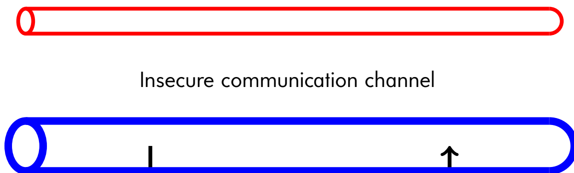
Notes

- Validation (instead of decryption) yielding success or failure
- Canonical validation is $\mathrm{val}_k(m, c) = \big(c \stackrel{?}{=} \mathrm{mac}_k(m)\big)$ for deterministic mac_k (most real-world message authentication schemes)

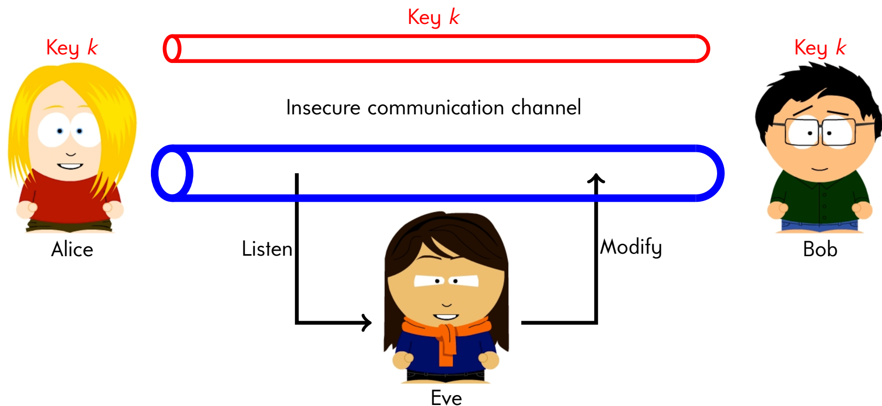
Message Authentication Codes

Secure communication channel

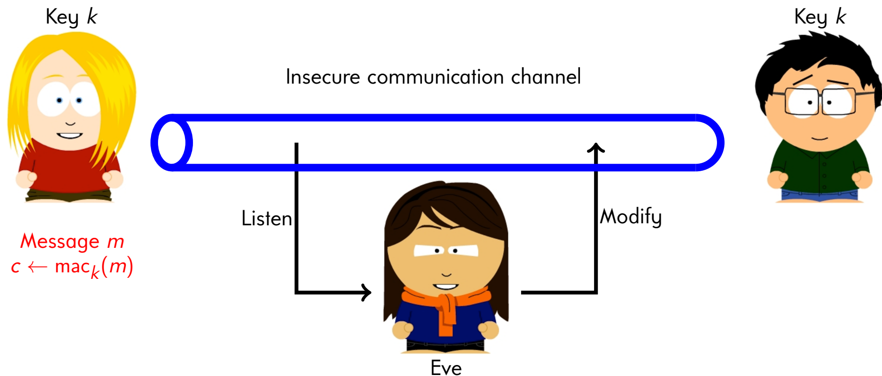
$k \leftarrow \text{gen}(1^n)$



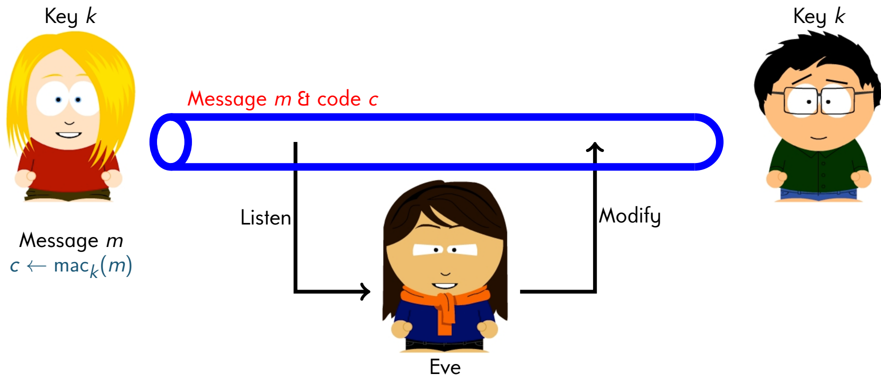
Message Authentication Codes



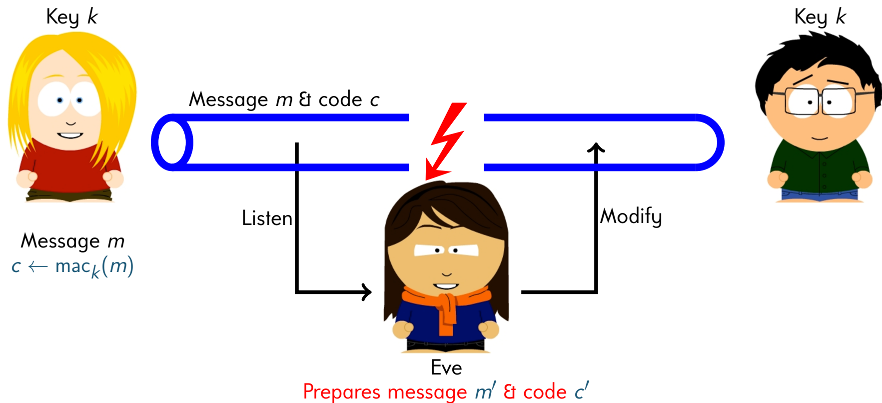
Message Authentication Codes



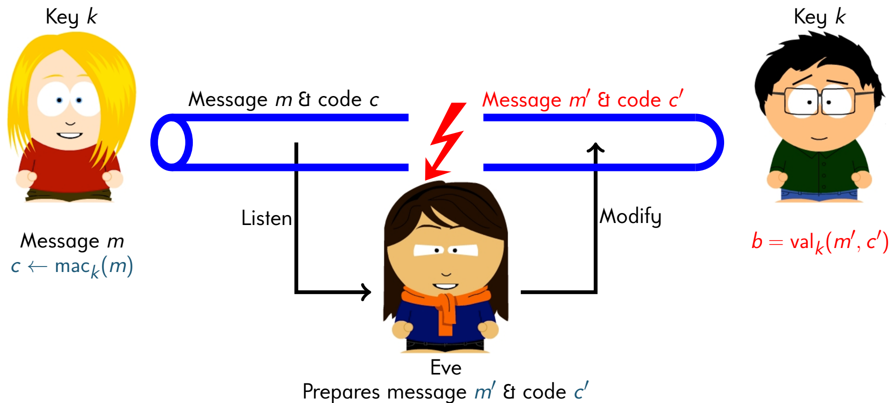
Message Authentication Codes



Message Authentication Codes



Message Authentication Codes



Message Authentication Codes

Security

- Impossible for Eve to generate message & code that validates
  (Eve should not be able to fool Bob)
  → existential unforgeability
- Impossible to achieve since Eve can replay
  (sent observed valid message & code again)
  → Eve needs to forge new message & code pair
- Assume CPA-model by default
  (adaptive chosen-message attack)

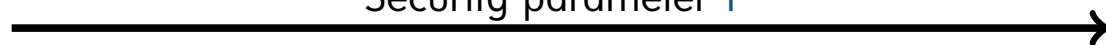
Message Authentication Codes

Alice

Eve

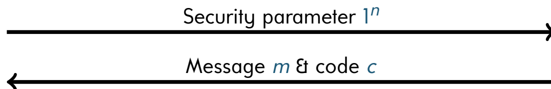
Message Authentication Codes

Alice
$k \leftarrow \text{gen}(1^n)$



Eve

Message Authentication Codes



Alice
$k \leftarrow \text{gen}(1^n)$

Eve

Message Authentication Codes

Alice

$k \leftarrow \text{gen}(1^n)$

$b = \text{val}_k(m, c)$

$b' = \left((m, c) \overset{?}{\notin} Q\right)$

Eve

Message Authentication Codes

Alice

$k \leftarrow \text{gen}(1^n)$

$b = \text{val}_k(m, c)$

$b' = \left((m, c) \overset{?}{\notin} Q\right)$

Return $b \wedge b'$

Eve

Message Authentication Codes

§7.3 Definition (Existential MAC forging game)

Let $n \in \mathbb{N}$, $\mathcal{E} = (\mathrm{gen}, \mathrm{mac}, \mathrm{val})$ efficient message authentication scheme and $\mathcal{A}$ stateful algorithm. Adversarial forging game $\mathrm{Forge}^{\mathrm{MAC}}_{\mathcal{E},\mathcal{A}}(n)$ is

1. Security parameter 1^n sent to adversary $\mathcal{A}$ and $k \leftarrow \mathrm{gen}(1^n)$
2. $\mathcal{A}$ selects message $m \in M$ and $c \in C$
3. $b = \mathrm{val}_k(m, c)$ and $b' = \left(\langle m, c\rangle \overset{?}{\notin} Q\right)$
   where Q set of message & code pairs obtained via oracle
4. Return $b \wedge b'$ ($\mathcal{A}$ wins)

Notes

- Q can be obtained by having Alice answer oracle queries
- New forged pair $\langle m, c\rangle$ sufficient; need not be new message
  (needs to be new message for deterministic scheme)

Message Authentication Codes

§7.4 Definition (Secure MAC)

Message authentication scheme $\mathcal{E} = (\mathsf{gen}, \mathsf{mac}, \mathsf{val})$ is secure
if for every PPT algorithm $\mathcal{A}$ with access to mac-oracle (encryption oracle)

$$\mathrm{P}\left[\mathsf{Forge}^{\mathsf{MAC}}_{\mathcal{E},\mathcal{A}}(n)\right] \simeq 0$$

Notes

- Adversary $\mathcal{A}$ wins only with negligible probability
- No requirements on forged messages (can be nonsensical)
- Still allows replay attacks

§7.5 Principle

Secure message authentication schemes permit replay attacks

Message Authentication Codes

§7.6 Construction

Let f be pseudorandom function. Message authentication scheme $\mathcal{E}'_f = (\mathrm{gen}, \mathrm{mac}, \mathrm{val})$ with

- $\mathrm{P}^n_K(k) = 2^{-n}$ for all $k \in \{0,1\}^n$
- $\mathrm{mac}_k(m) = f_k(m)$ for all $k, m \in \{0,1\}^n$
- Canonical validation

Notes

- Directly utilize pseudorandom function
- Practically useless since code & message have equal length
- Canonical validation susceptible to timing attacks

§8.1 Theorem (Secure MAC)

Scheme $\mathcal{E}'_f$ is secure for every pseudorandom function f

Proof (1/3)

Consider (inefficient) message authentication scheme $\mathcal{E}' = (\text{gen}', \text{mac}', \text{val}')$ given by $\mathsf{P}^n_K(g) = 2^{-2^n n}$ for every $g \colon \{0,1\}^n \to \{0,1\}^n$, $\text{mac}'_g(m) = g(m)$ for every $m \in \{0,1\}^n$, and canonical validation (essentially $\mathcal{E}'_f$ but using truely random function g instead of f_k for random k).

Let $\mathcal{A}$ be PPT adversary. Since $\mathcal{A}$ needs to forge valid code for new message m (because $\mathcal{E}'$ is deterministic) and $g(m)$ is chosen uniformly at random, we immediately have $\mathsf{P}\big[\text{Forge}^{\text{MAC}}_{\mathcal{E}',\mathcal{A}}(n)\big] \leq 2^{-n}$.

Next we prove the first almost equality in

$$\mathsf{P}\big[\text{Forge}^{\text{MAC}}_{\mathcal{E}'_f,\mathcal{A}}(n)\big] \simeq \mathsf{P}\big[\text{Forge}^{\text{MAC}}_{\mathcal{E}',\mathcal{A}}(n)\big] \simeq 0$$

(winning chances of $\mathcal{A}$ are essentially the same against schemes $\mathcal{E}'_f$ and $\mathcal{E}'$)

Proof (2/3)

Construct PPT distinguisher D (against pseudorandomness of f) with oracle access to $\mathcal{O}$ and input 1^n:

1. Set $Q \leftarrow \emptyset$
2. Run $\mathcal{A}(1^n)$ and answer its mac oracle queries as stated below
3. Receive message $m \in \{0,1\}^n$ and code $c \in \{0,1\}^n$
4. Return $\left(\mathcal{O}(m) \stackrel{?}{=} c\right) \wedge \left(m \stackrel{?}{\notin} Q\right)$

Answer all mac oracle queries $\mathsf{mac}(m)$ by $\mathcal{A}$ as follows:

$$Q \leftarrow Q \cup \{m\} \text{ ; return } \mathcal{O}(m)$$

Clearly D is PPT algorithm since $\mathcal{A}$ is efficient.

Proof (3/3)

We observe that this distinguisher D succeeds exactly when

- $\mathrm{Forge}^{\mathrm{MAC}}_{\mathcal{E}'_f,\mathcal{A}}(n)$ succeeds if oracle f_k with $k \leftarrow U_n$ is supplied
- $\mathrm{Forge}^{\mathrm{MAC}}_{\mathcal{E}',\mathcal{A}}(n)$ succeeds if oracle g with $g \leftarrow F_n$ is supplied

Hence

$$\begin{aligned} \mathrm{P}\left[\mathrm{Forge}^{\mathrm{MAC}}_{\mathcal{E}'_f,\mathcal{A}}(n)\right] &= \mathrm{E}\Big[\mathrm{P}\big(D^{f_k}(1^n)\big)\Big]_{k\leftarrow U_n} \\ &\simeq \mathrm{E}\Big[\mathrm{P}\big(D^{g}(1^n)\big)\Big]_{g\leftarrow F_n} = \mathrm{P}\left[\mathrm{Forge}^{\mathrm{MAC}}_{\mathcal{E}',\mathcal{A}}(n)\right] \end{aligned}$$

which proves

$$\mathrm{P}\left[\mathrm{Forge}^{\mathrm{MAC}}_{\mathcal{E}'_f,\mathcal{A}}(n)\right] \simeq \mathrm{P}\left[\mathrm{Forge}^{\mathrm{MAC}}_{\mathcal{E}',\mathcal{A}}(n)\right] \simeq 0 \qquad \square$$

Message Authentication Codes

Notes

- Secure MAC for short, fixed-length messages (equally long code)

$\rightarrow$ Need to generalize to more flexible MACs

- MACs build from block cipher primitive (pseudorandom function)
- Utilize block cipher operation modes

Summary

- Message authentication scheme
- Security of MACs