Hash Function

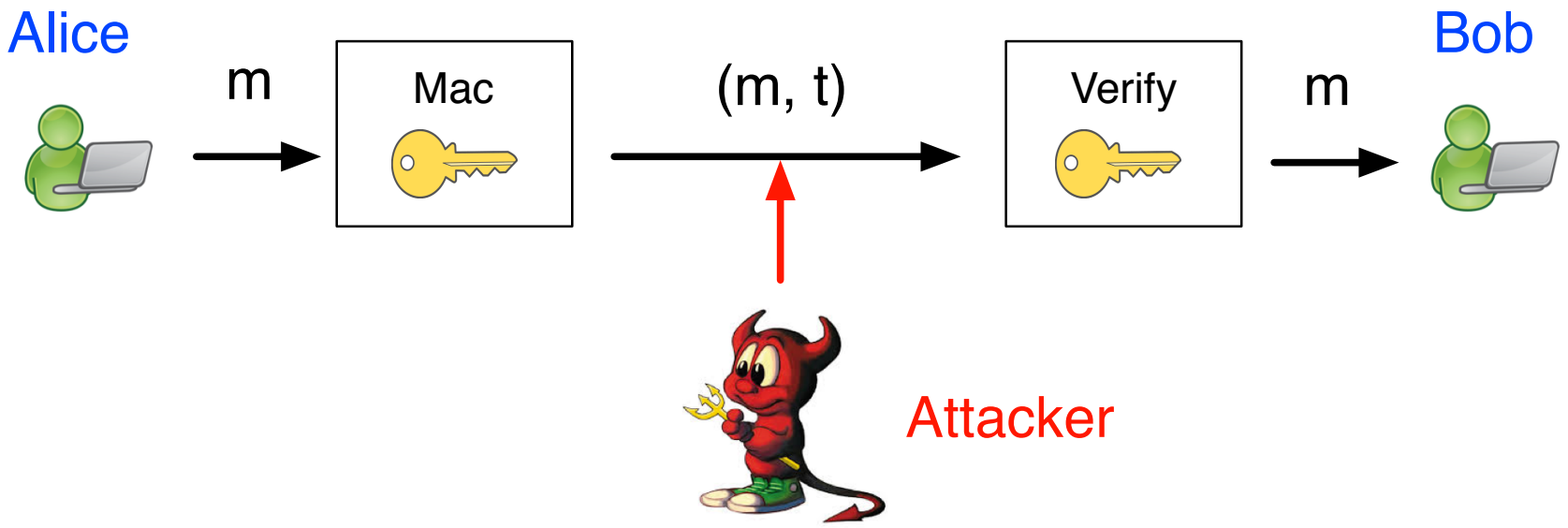
CS 5158/6058 Data Security and Privacy

Spring 2018

Instructor: Boyang Wang

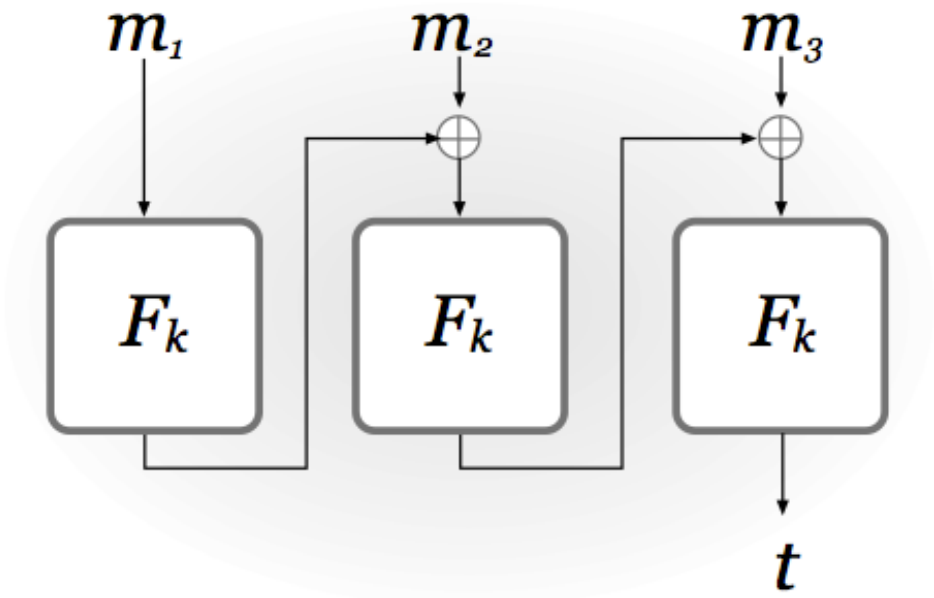
Message Authentication Code

- Alice computes a tag for a message
- Bob can verify a message m using its tag t
  - If valid, accept a message
  - Otherwise, drop or ignore a message

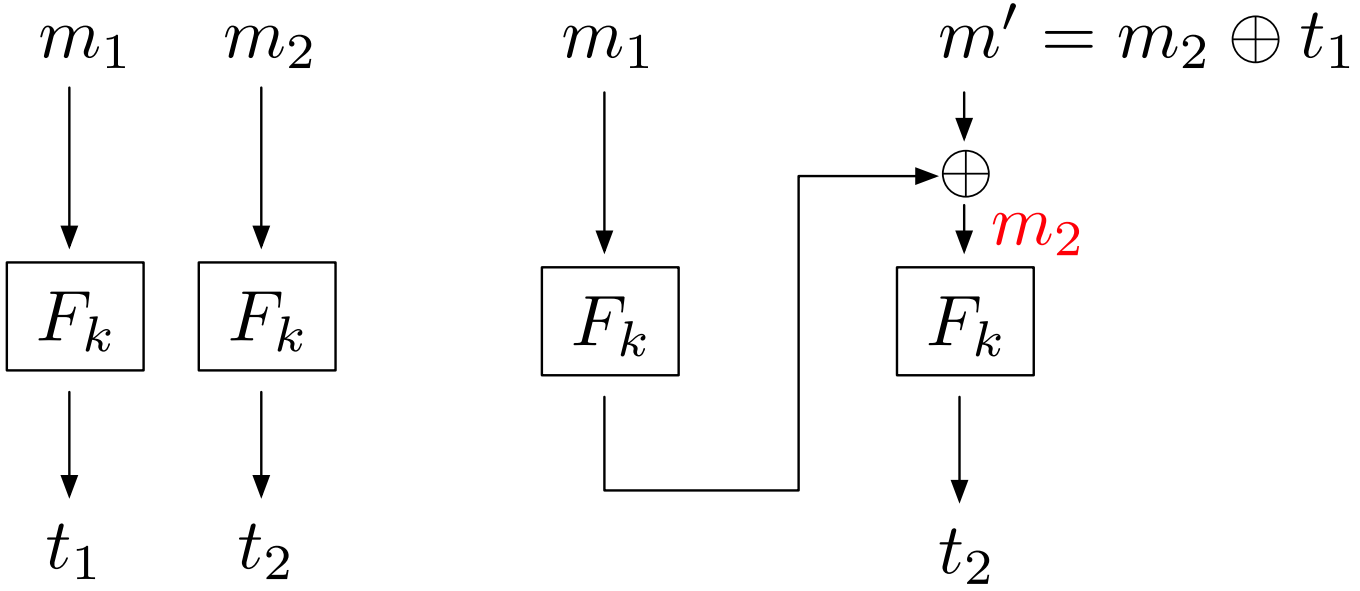


CBC-MAC

Build a fixed-length MAC for l*n-bit messages



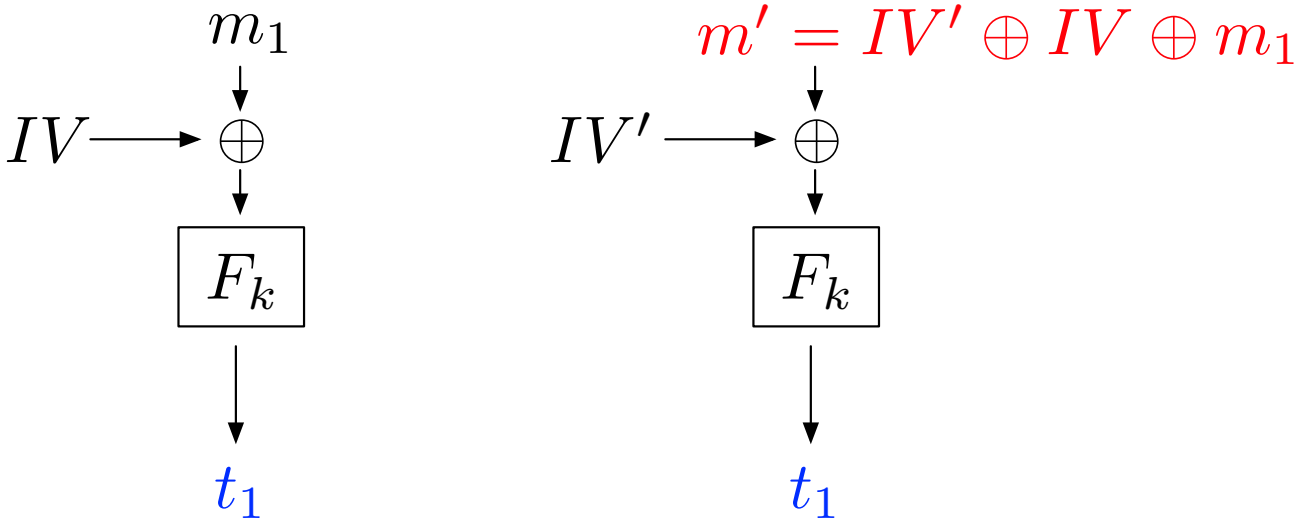
- Secure if F is a PRF, only for l*n-bit messages
  - Not secure for messages with arbitrary length



- Example: Adversary knows $(m_1,t_1) = (10,11)$, $(m_2,t_2) = (01,10)$
  - compute $m' = m_2 \oplus t_1 = 01 \oplus 11 = 10$
  - fake "new" $m = m_1 m' = 1010$, send (m,t_2) to Bob
  - Bob will accept (m,t_2), since Verify(m,t_2) will output Yes

CBC-MAC

- Adversary only has (m_1, IV, t_1)
- Fakes a message m'=IV'⊕IV⊕m_1, where IV' is "new"
  - m' is a "new" message, should not be valid
  - But (m', IV', t_1) will pass the verification



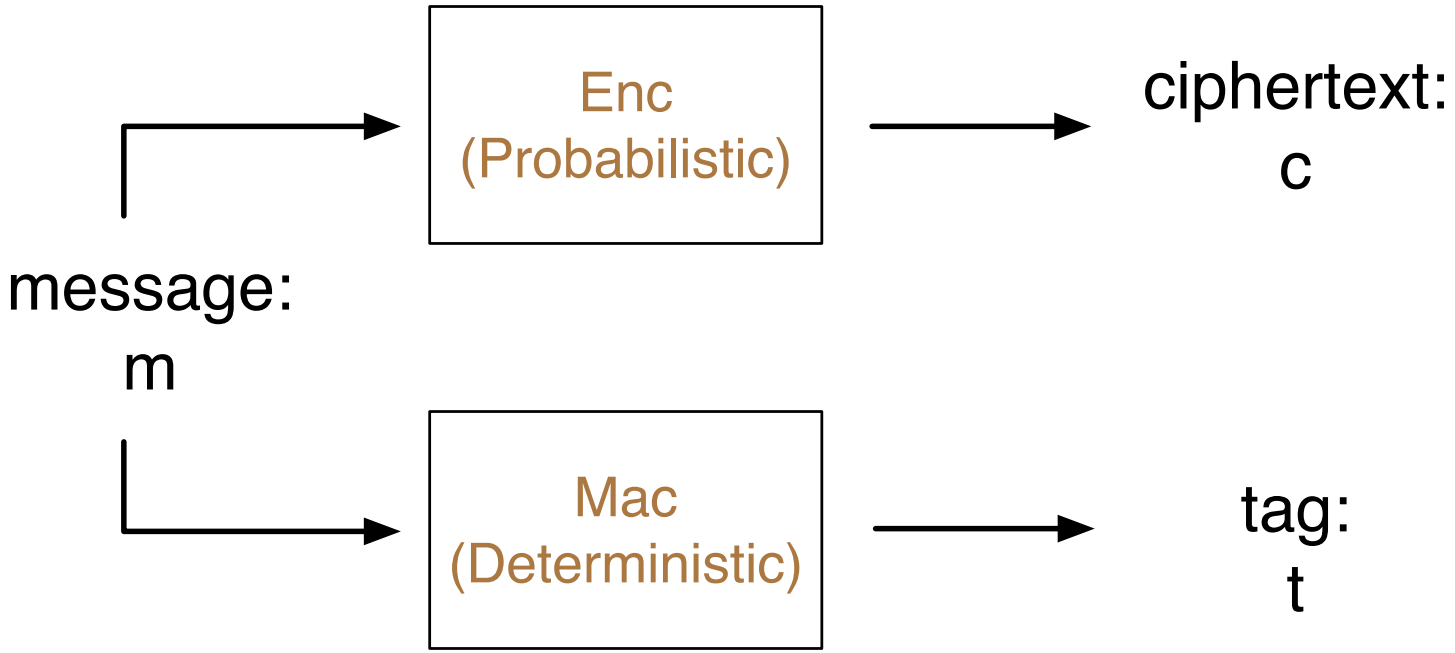
Encrypt & Authenticate

- Authentication on private message
  - Message is from the right entity
  - Message is correct/unchanged
  - Message is encrypted/private

- Three possible solutions:
  - Encrypt-and-authenticate
  - Authenticate-then-encrypt
  - Encrypt-then-authenticate

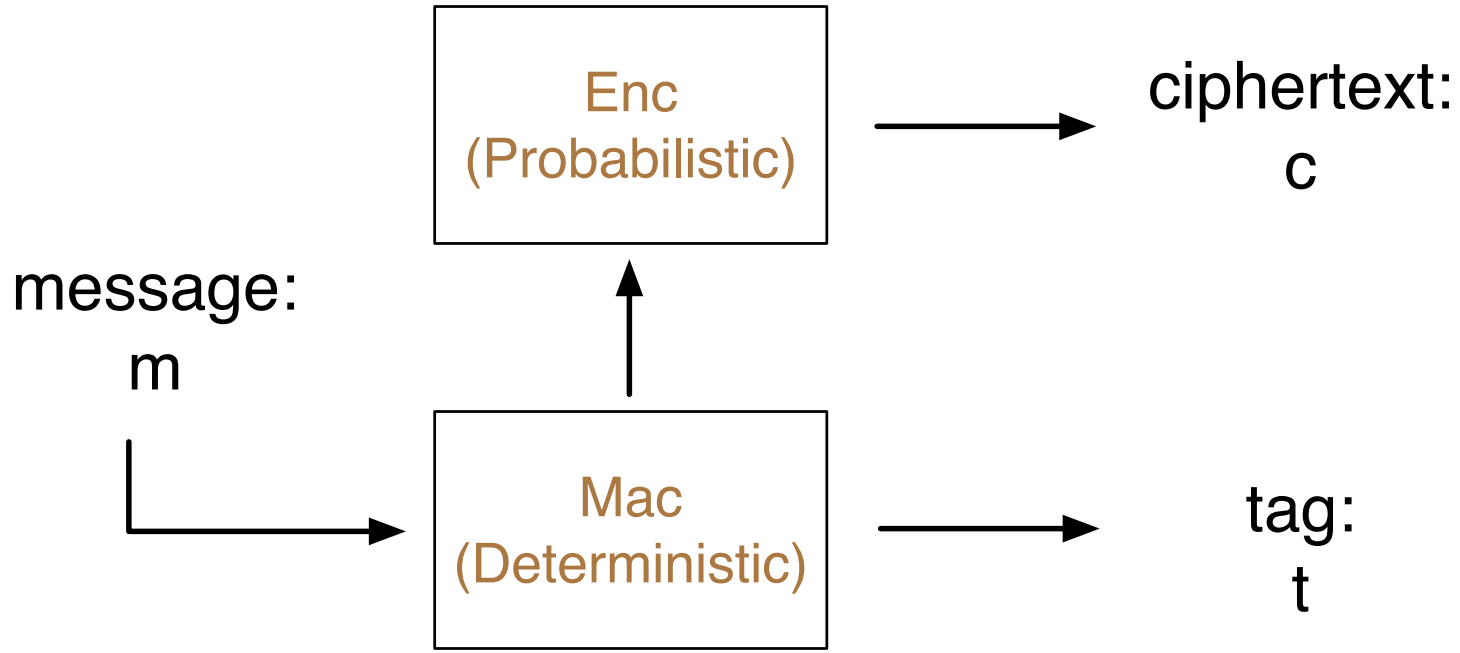
Encrypt & Authenticate

- Given Enc is probabilistic & Mac is deterministic
- Enc and Mac use two different keys
  - Output (c,t) is still deterministic, not CPA-secure



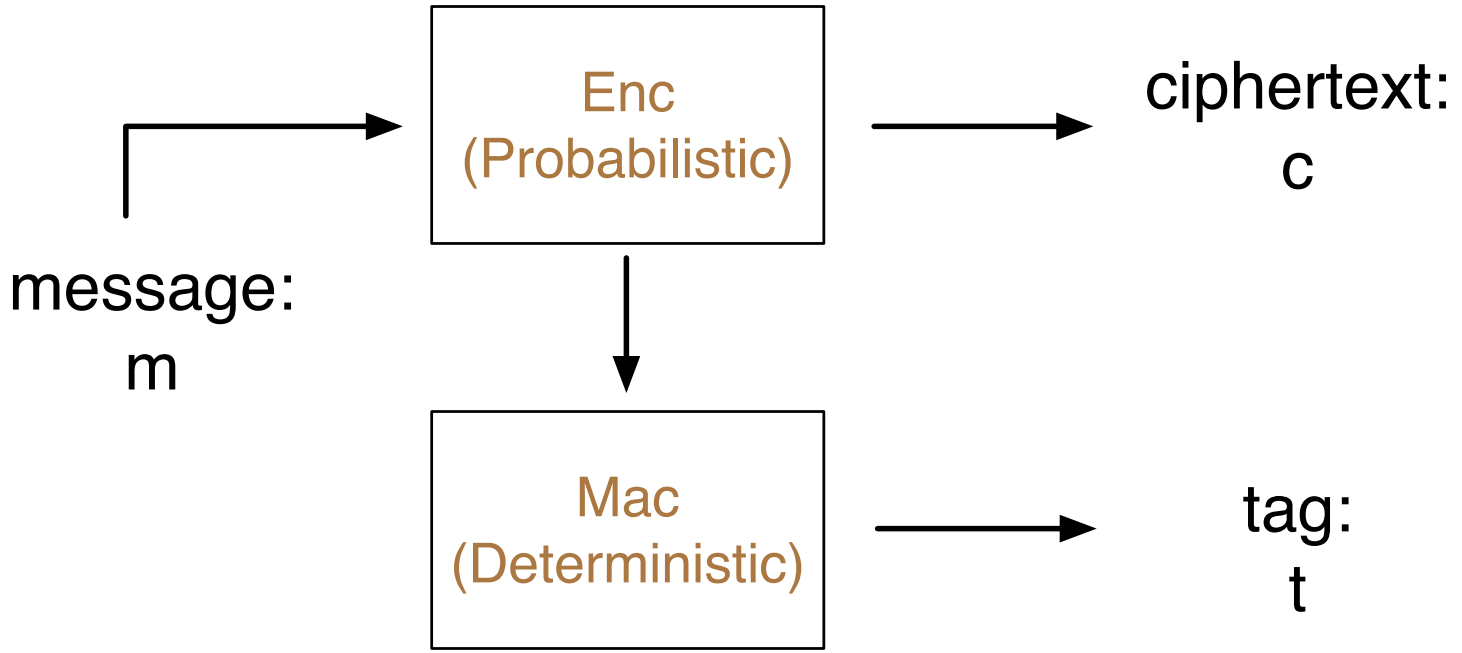
Authenticate-then-Encrypt

- Given Enc is probabilistic & Mac is deterministic
- Enc and Mac use two different keys
  - Output (c,t) is still deterministic, not CPA-secure



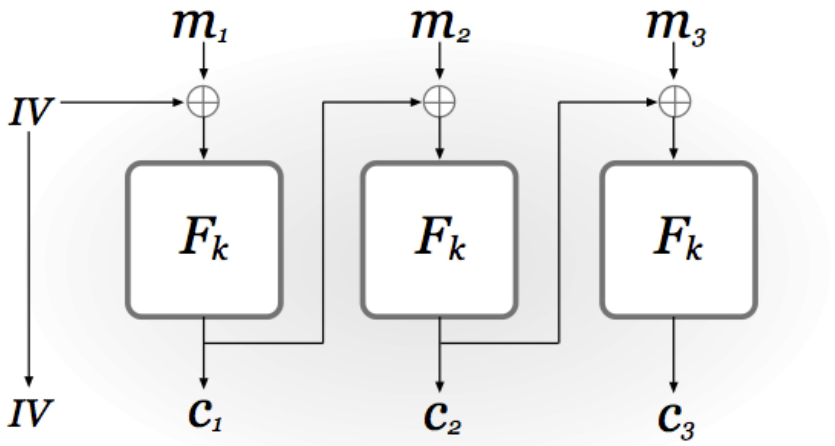
Encrypt-then-Authenticate

- Given Enc is probabilistic & Mac is deterministic
- Enc and Mac use two different keys
  - Output (c,t) is still probabilistic, still CPA-secure

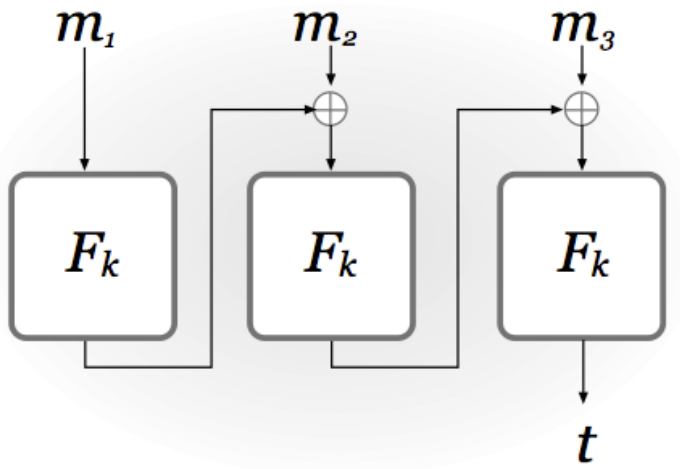


x	00	01	10	11
k=00,F(x)	11	00	01	10
k=01,F(x)	10	11	00	01
k=10,F(x)	01	10	11	00
k=11,F(x)	00	01	10	11

Example: given k_e=10, k_m=01 m=1011, and IV=01



Use Encrypt-then-authenticate (CBC-Enc then CBC-MAC)



What is c=?? and t=??

x	00	01	10	11
k=00,F(x)	11	00	01	10
k=01,F(x)	10	11	00	01
k=10,F(x)	01	10	11	00
k=11,F(x)	00	01	10	11

Example: given k_e=10, k_m=01

m=1011, and IV=01

Compute c first with CBC-Enc

IV $\oplus$ m_1=01$\oplus$10=11

F_k(11)=00, c_1=00

c_1 $\oplus$ m_2=00$\oplus$11=11

F_k(11)=00, c_2=00

c=(IVc_1c_2)=(010000)

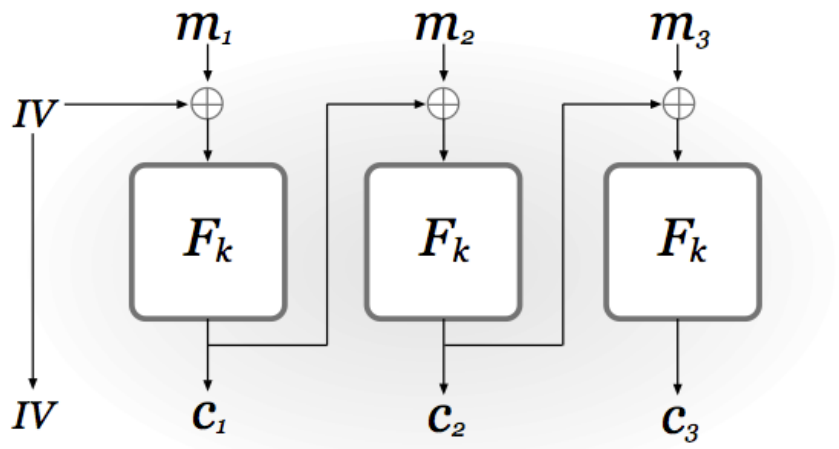


FIGURE 3.7: Cipher Block Chaining (CBC) mode.

x	00	01	10	11
k=00,F(x)	11	00	01	10
k=01,F(x)	10	11	00	01
k=10,F(x)	01	10	11	00
k=11,F(x)	00	01	10	11

Example: given k_e=10, k_m=01

c=010000

Compute t for c with CBC-Mac

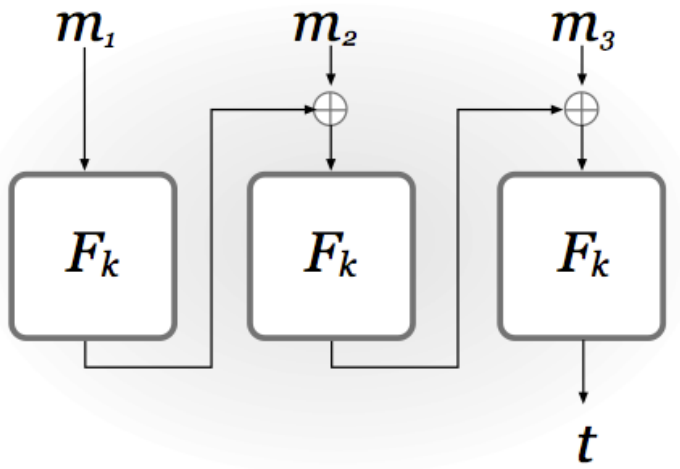
$F_k(01)=11$, $x_1=11$

$x_1 \oplus m_2=11 \oplus 00=11$

$F_k(11)=01$, $x_2=01$

$x_2 \oplus m_3=01 \oplus 00=01$

$F_k(01)=11$, t=11

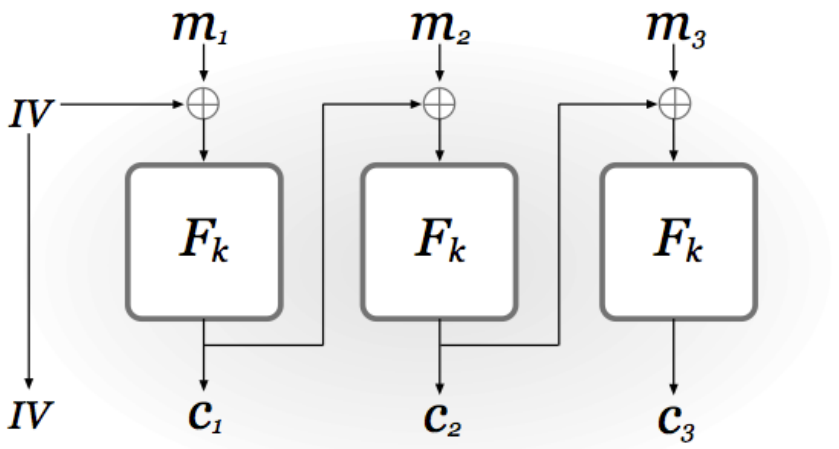


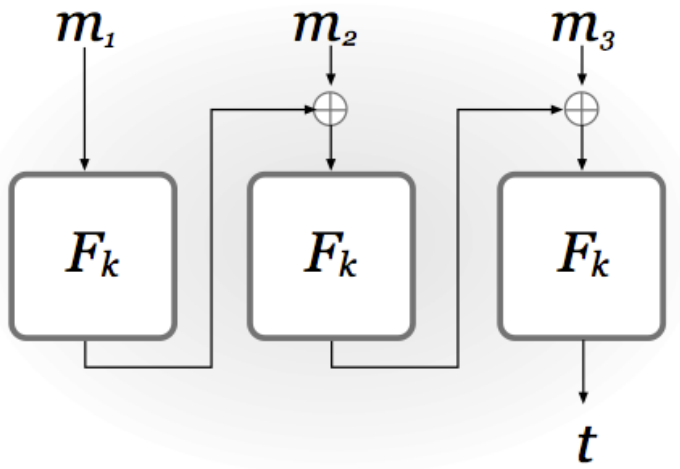
x	00	01	10	11
k=00,F(x)	11	00	01	10
k=01,F(x)	10	11	00	01
k=10,F(x)	01	10	11	00
k=11,F(x)	00	01	10	11

Example: given k_e=10, k_m=01
m=1011, and IV=01

Use Encrypt-then-authenticate
(CBC-Enc then CBC-MAC)

c=010000 and t=11
(or IV=01, c=0000, t=11)



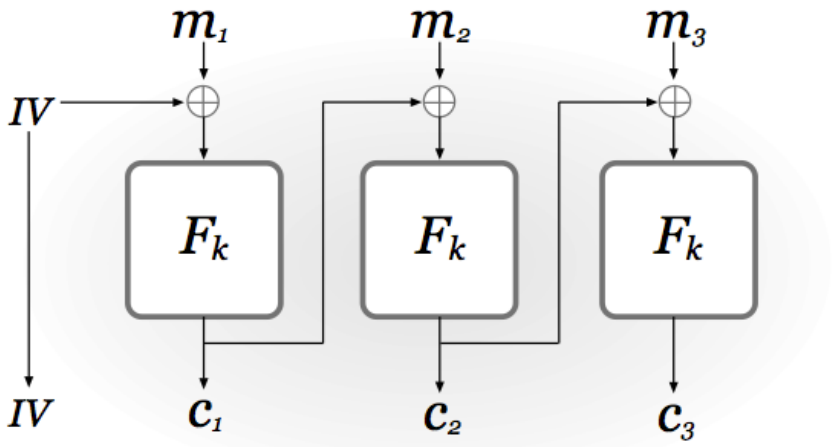


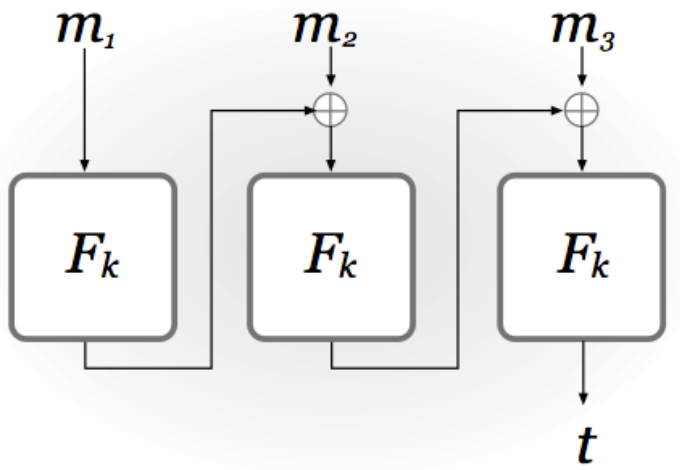
x	00	01	10	11
k=00,F(x)	11	00	01	10
k=01,F(x)	10	11	00	01
k=10,F(x)	01	10	11	00
k=11,F(x)	00	01	10	11

<u>Practice:</u> given k_e=10, k_m=01 m=1100, and IV=11

Use Encrypt-then-authenticate (CBC-Enc then CBC-MAC)

What is c=?? and t=??





x	00	01	10	11
k=00,F(x)	11	00	01	10
k=01,F(x)	10	11	00	01
k=10,F(x)	01	10	11	00
k=11,F(x)	00	01	10	11

Practice: given k_e=10, k_m=01

m=1100, and IV=11

Compute c first with CBC-Enc

IV $\oplus$ m_1=11$\oplus$11=00

F_k(00)=01, c_1=01

c_1 $\oplus$ m_2=01$\oplus$00=01

F_k(01)=10, c_2=10

c=(IVc_1c_2)=(110110)

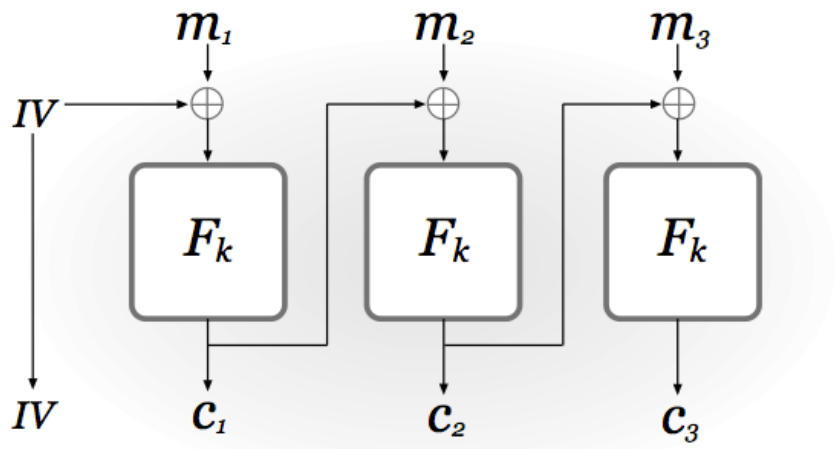


FIGURE 3.7: Cipher Block Chaining (CBC) mode.

x	00	01	10	11
k=00,F(x)	11	00	01	10
k=01,F(x)	10	11	00	01
k=10,F(x)	01	10	11	00
k=11,F(x)	00	01	10	11

Practice: given k_e=10, k_m=01

c=110110

Compute t for c with CBC-Mac

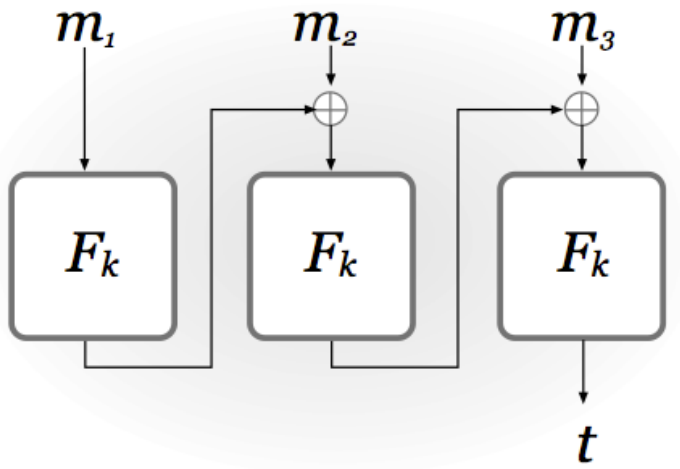
$F_k(11)=01$, $x_1=01$

$x_1 \oplus m_2=01 \oplus 01=00$

$F_k(00)=10$, $x_2=10$

$x_2 \oplus m_3=10 \oplus 10=00$

$F_k(00)=10$, t=10



Hash Function

- A mapping/function from an arbitrary long input to a fixed-length output (a digest or hash value)

  - m="*Thomas Edward Patrick Brady Jr. (born August 3, 1977) is an American football quarterback for the New England Patriots of the National Football League (NFL). He is one of only two players to win five Super Bowls and the only player to win them all playing for one team.*" (from wikipedia)
    - H(m) = `0xde4901db32` (40 bits)

Hash Function

- Star Wars: The Last Jedi (10 GB)
  - H(m) = `0xd490d1bca2`
- All the UC emails (500 GB)
  - H(m) = `0x93da128bed`

- Two similar messages have totally different digests
  - m = "Welcome to Cincinnati"
    - H(m) = `0x5836f729ac`
  - m = "Welcome to Cincinnati!"
    - H(m) = `0x91be257108`

Collision

- Hash Function: H(m) = h, deterministic
  - Easy to compute h, hard to invert
  - Collision: two inputs map to a same output
    - For x and y, x!=y, but H(x) == H(y)
    - Collision must exist
      - Number of inputs >> number of outputs
    - Collision-resistance: but hard to find
      - Try all the possible inputs will certainly find a collision, but it takes (almost) forever

Collision

- Collision: two inputs map to a same output
  - For x and y, x!=y, but H(x) == H(y)
  - Why collision must exist?
    - Number of inputs >> number of outputs

  - Example: 41 inputs v.s. 40 outputs: we have 41 students, 40 computers, each student uses one computer, then at least two students will share a same computer (a collision).

Applications

- Due to collision-resistance, hash function can be used to check data is correct

- Example 1: login passwords for online websites
  - Alice types password "`1234`"
  - Alice sends "**`alice`**" + H(`1234`) to Amazon
  - Amazon checks its database, if there is a <"**`alice`**", H(`1234`)>, then Alice can login.

Applications

- Due to collision-resistance, hash function can be used to check data is correct

- Example 2: updating a software
  - Alice needs to update Windows 7 to 10
  - Alice downloads Windows 10
  - Alice computes H(`Windows10`)
  - Alice checks website for true hash value h, if it is equal to H(`Windows10`), then Alice install app.

Applications

- Due to collision-resistance, hash function can be used to check data is correct

- Example 3: malware detection
  - Alice finds an odd app abc.exe
  - Alice computes H(**`abc.exe`**)
  - Alice uploads to Avast
  - Avast checks its database, if there is a H(**`abc.exe`**) that is malware, alerts Alice and Alice removes it.

Non-crypto Hash Functions

- Regular non-crypto hash function
  - Collision-resistance is desired (but not required)
  - Look-up table: use H(x) as an index, find x with O(1) running time

    <**packers**, H(**packers**) >

    <**steelers**, H(**steelers**)>

    ……

    <**patriots**, H(**patriots**)>
  - Without hash, linear search time takes O(n)

Hash Function

- Cryptographic hash function
  - Collision-resistance is required
  - Harder to design, slower than regular hash func.
  - Crypto hash: MD5, SHA1, SHA2, etc.

- MD5
  - Output 128-bit digests, published in 1992
  - Attack in 2004 (1 hour), in 2013 (1 second)

Hash Functions in Practice

- SHA-1, Secure Hash Algorithm 1
  - Output 160-bit digests, published in 1995,
  - Replaced in 2010, Apple and Mozilla in 2017

- SHA-2, Secure Hash Algorithm 2
  - E.g., SHA-256, output 256-bit digests
  - Published in 2001, still secure so far

Hash Functions in Practice

- Assume we have a file "**`hello.txt`**", in this file we have a string "**`Welcome to Cincinnati`**"
- You have openssl installed on your PC/laptop
  - Check: **`openssl version`**

- **`openssl dgst -md5 hello.txt`**
  - 10a1b211061f18917c0563b3deb80214

- **`openssl dgst -sha1 hello.txt`**
  - 48da86fe64c817f0c656c177cb92ec6acf3173ed

Hash Function

- Formal Definition: a keyed function
- KeyGen: takes a security parameter 1^n as input, and outputs a key s
- Hash: takes a key s and a string $x \in \{0,1\}^*$ as input, and outputs a string $h \in \{0,1\}^l$
- Key is public, i.e., unkeyed in practice
  - Alice sends a file with its hash, Bob verifies the integrity of file with this hash

Collision Resistance

- Any PPT adversary finds a collision with a negligible probability

A security game $\mathsf{HashColl}_{\mathcal{A},\Pi}(n)$

- A key s is generated by running $\mathsf{KeyGen}(1^n)$
- Adversary $\mathcal{A}$ is given key s and outputs x, x'.
- Output 1 if $\mathsf{Hash}_s(x) = \mathsf{Hash}_s(x')$, and 0 otherwise

$$\Pr[\mathsf{HashColl}_{\mathcal{A},\Pi}(n) = 1] \leq \mathtt{negl}(n)$$

Merkle-Damgard Transform

- MD transform: from fixed-length to arbitrary-length
  - If we have a hash function h: $\{0,1\}^{2n} \longrightarrow \{0,1\}^{n}$
  - then we can build a hash function H with arbitrary-length: $\{0,1\}^{*} \longrightarrow \{0,1\}^{n}$

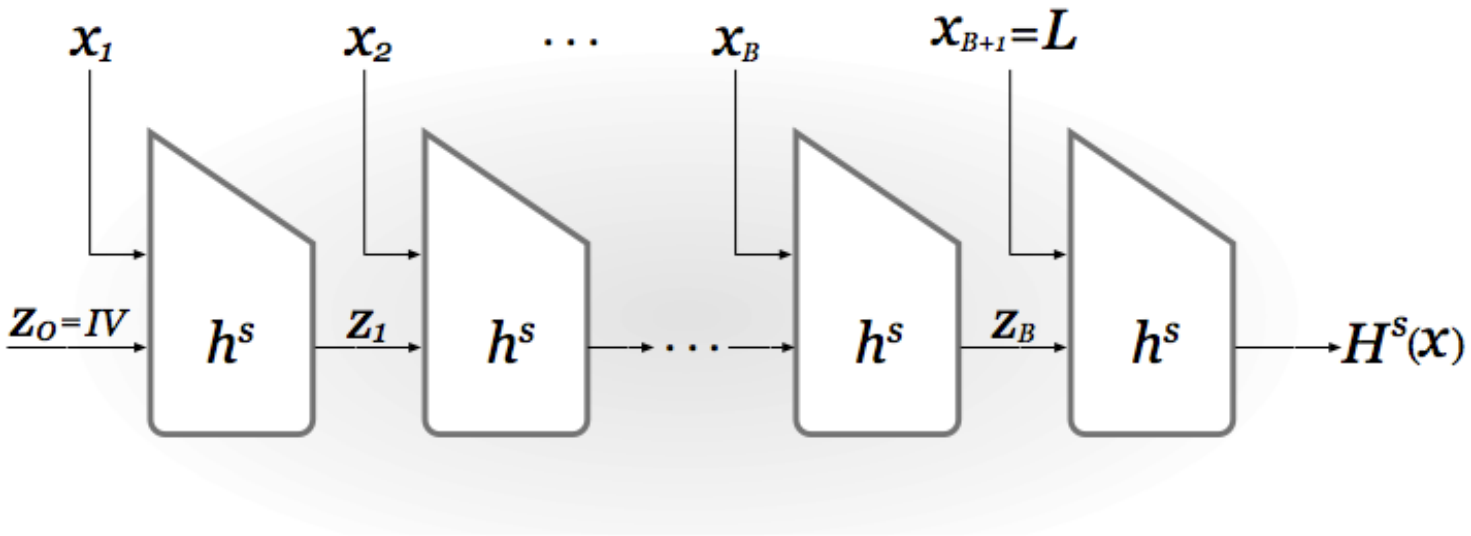


FIGURE 5.1: The Merkle–Damgård transform.

Merkle-Damgard Transform

- Divide message into blocks
- Compute hash of a block, and use its output as half of the input for the next block's hash function
- Length L (in binary) is used as the last block(s)

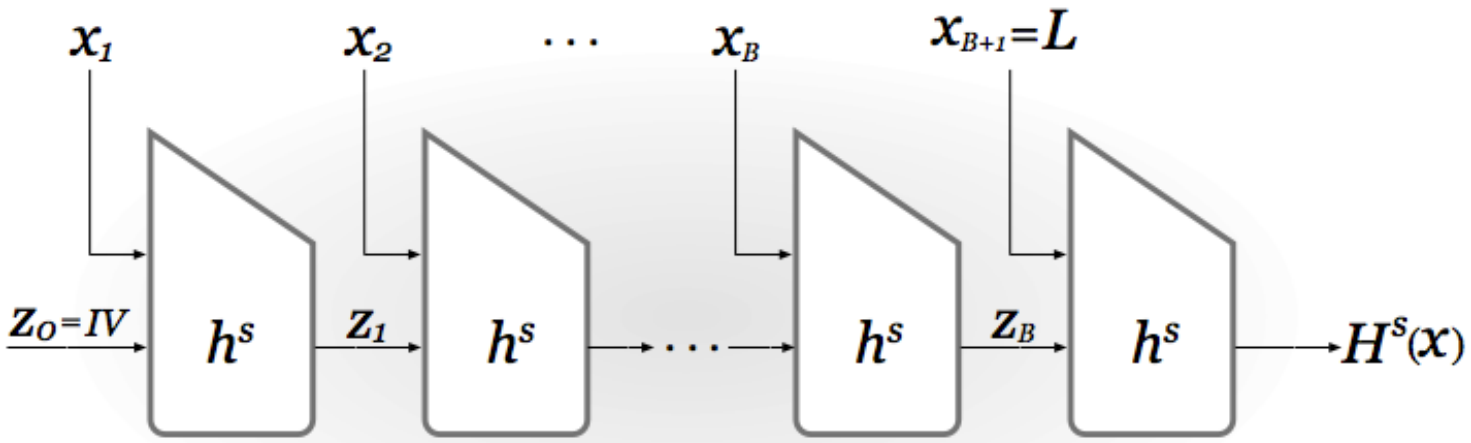
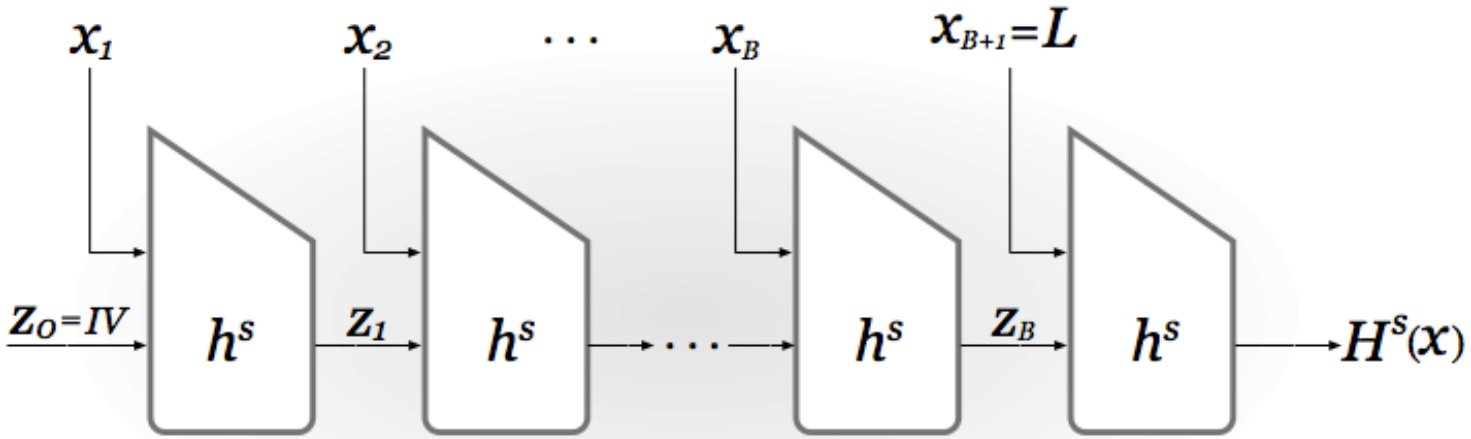
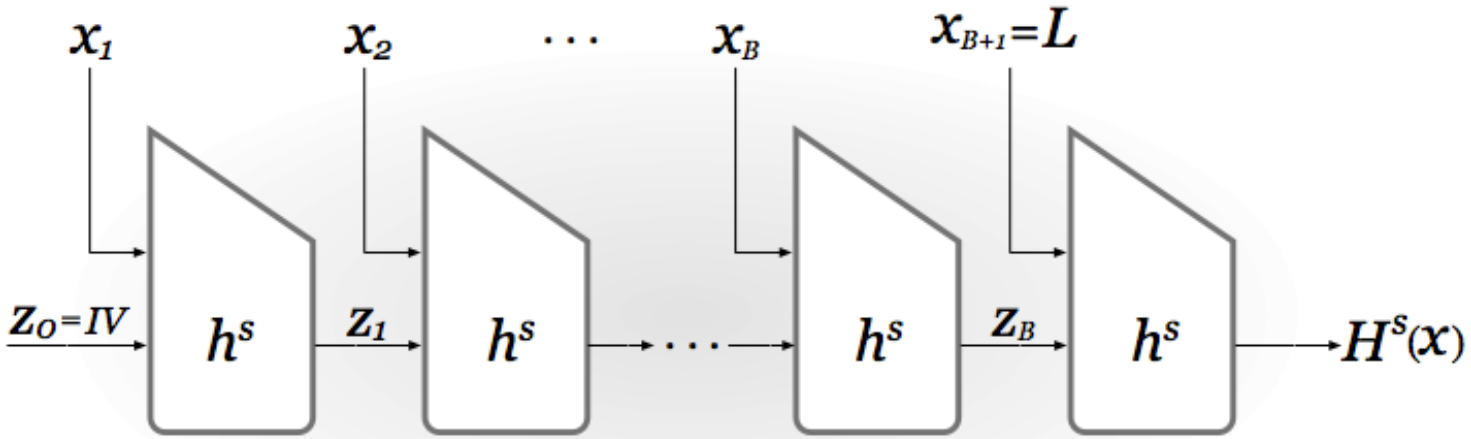


FIGURE 5.1: The Merkle–Damgård transform.



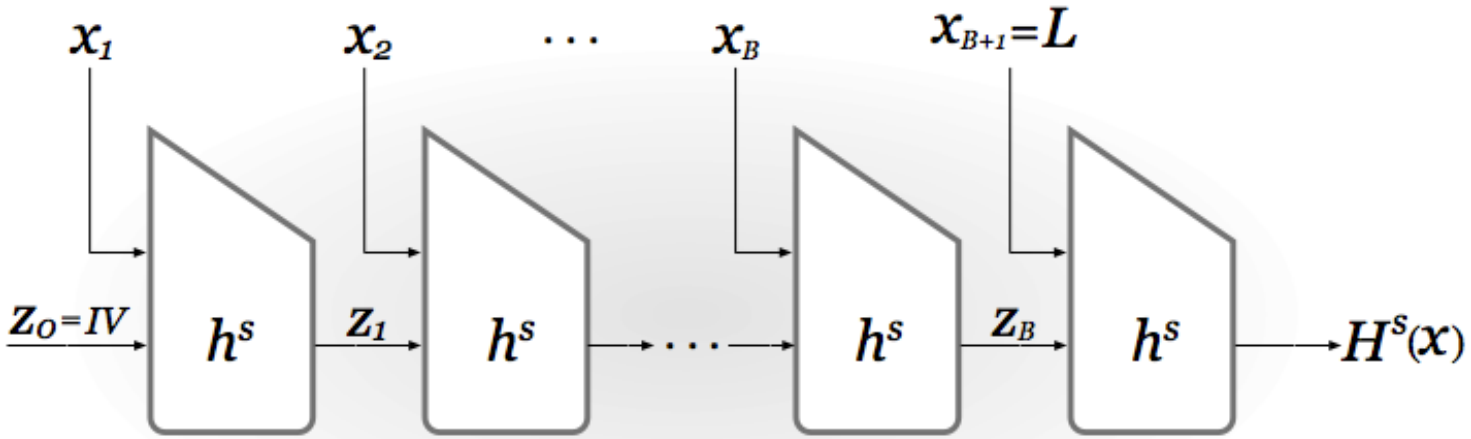
x	00	01	10	11
h(x)	1	0	1	0

- Example: m=101, L=3, IV=0 (fixed), H(x) =??
- m=101=($x_1x_2x_3$), L=3=(L_1L_2)=10
  - $h(IV||x_1) = h(01) = 0,\ Z_1 = 0;$
  - $h(Z_1||x_2) = h(00) = 1,\ Z_2 = 1;$
  - $h(Z_2||x_3) = h(11) = 0,\ Z_3 = 0;$



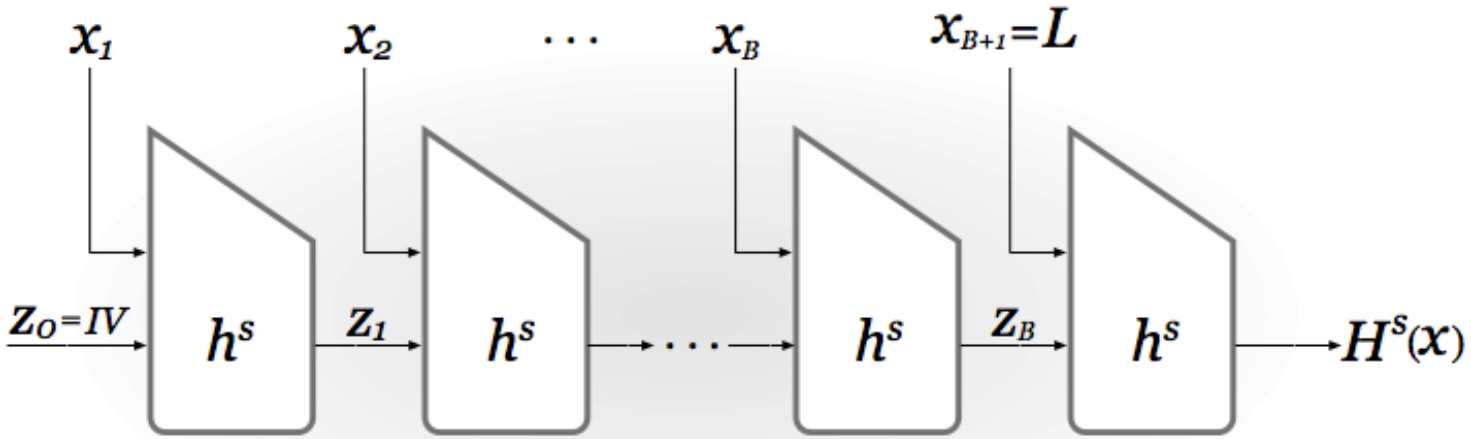
x	00	01	10	11
h(x)	1	0	1	0

- m=101=($x_1x_2x_3$), L=3=(L_1L_2)=10
  - h(Z_2||x_3) = h(11) = 0, Z_3 = 0;
  - h(Z_3||L_1) = h(01) = 0, Z_4 = 0;
  - h(Z_4||L_2) = h(00) = 1, H(m) = 1;
- From h: $\{0,1\}^2$ —> $\{0,1\}^1$ to H: $\{0,1\}^3$ —> $\{0,1\}^1$



x	00	01	10	11
h(x)	1	0	1	0

- Practice: m=1011, L=4, IV=0 (fixed), H(x) =??
- m=1011=($x_1x_2x_3x_4$), L=4=(L_1L_2)=11
  - h(IV||x_1) = h(01) = 0, Z_1 = 0;
  - h(Z_1||x_2) = h(00) = 1, Z_2 = 1;
  - h(Z_2||x_3) = h(11) = 0, Z_3 = 0;



x	00	01	10	11
h(x)	1	0	1	0

- m=1011=($x_1x_2x_3x_4$), L=4=(L_1L_2)=11
  - $h(Z_2||x_3) = h(11) = 0,\ Z_3 = 0;$
  - $h(Z_3||x_4) = h(01) = 0,\ Z_4 = 0;$
  - $h(Z_4||L_1) = h(01) = 0,\ Z_5 = 0;$
  - $h(Z_5||L_2) = h(01) = 0,$ H(m) = 0;

Merkle-Damgard Transform

- Arbitrary-length (KeyGen, Hash) is collision resistant if fixed-length (g, h) is collision resistant.

- Collision in Hash(m) could happen in two cases:
  - Messages are same, but length is different
    - m==m, L!=L', indicate h(z||L)==h(z||L')
  - Messages are different, but length is same
    - m!=m', L==L, indicate h(z||m)==h(z||m')
- h is a collision resistant, two cases both happen with negligible probability

Additional Reading

Chapter 5, *Introduction to Modern Cryptography, Drs. J. Katz and Y. Lindell, 2nd edition*