A Few Basic Elements of Communication Security

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

May 6, 2020

Some Advice

Some Advice

- ***Make backups*** of your data
- ***Do NOT trust the network!***
  - just don’t use it to transmit or store your most sensitive information

Some Advice

- ***Make backups*** of your data
- ***Do NOT trust the network!***
  - just don't use it to transmit or store your most sensitive information

---

- Understand the basics of public-key cryptography
- Communicate with ***end-to-end encryption*** (e.g., e-mail)
- Use ***trusted certificates***
- ***Encrypt your confidential data*** (and make backups)
- Use ***strong passwords***
- You might as well encrypt *all* your data
- Tools/technologies: ***ssh***, ***pgp*** (or ***gpg***)

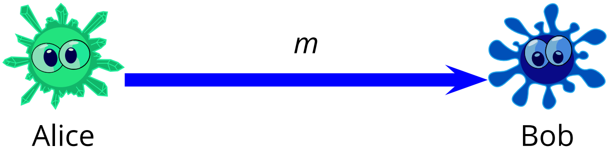
Outline

- Communication security model
- Information-theoretic privacy
- Substitution ciphers
- Intro to modern cryptography
- One-time pad
- Block siphers
- Cryptographic hash functions
- Public-key cryptosystems

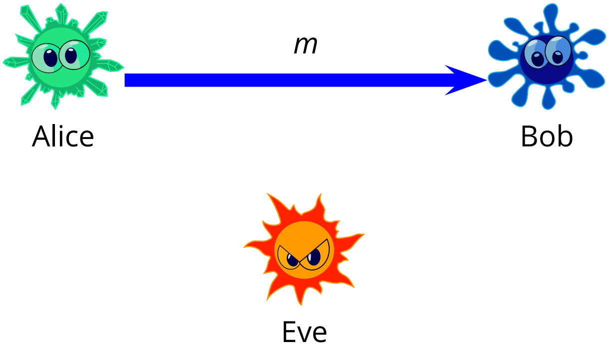
Communication Security

- *Communication model:* Alice sends a message m to Bob

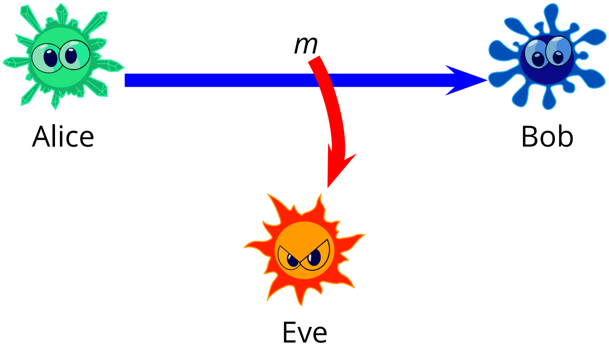
- *Communication model:* Alice sends a message m to Bob



- *Communication model:* Alice sends a message m to Bob

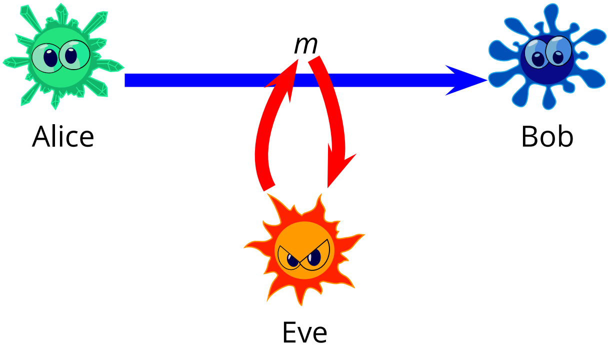


- *Communication model:* Alice sends a message *m* to Bob



- *Passive adversary*
  - can read the message

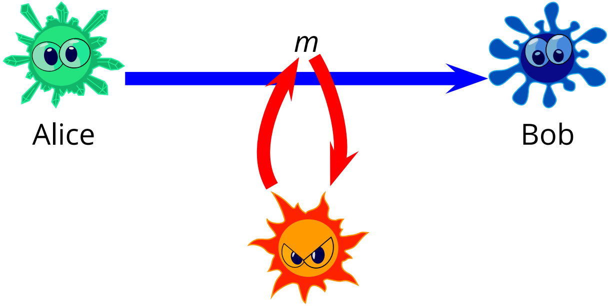
- *Communication model:* Alice sends a message *m* to Bob



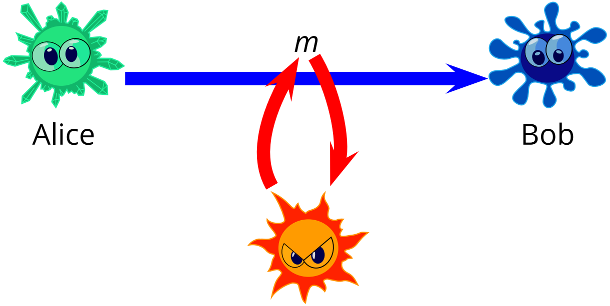
- *Passive adversary*
  - can read the message
- *Active adversary*
  - can modify the message

Goals

Goals

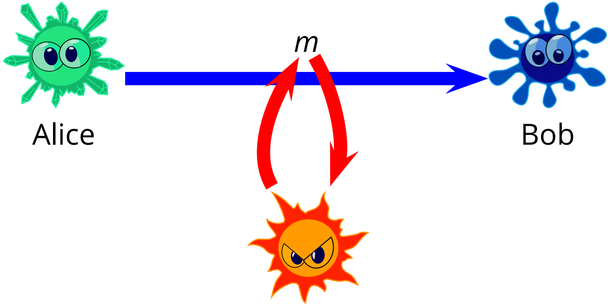


Goals



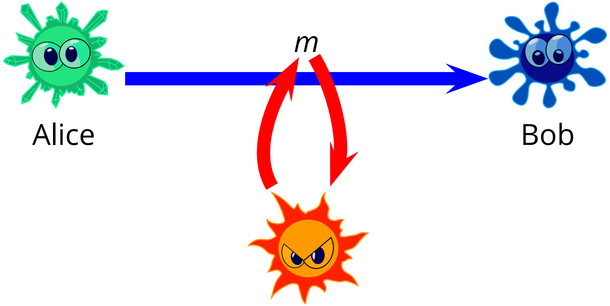
- ***Confidentiality (a.k.a., privacy):*** Alice wants to make sure that only Bob sees the message

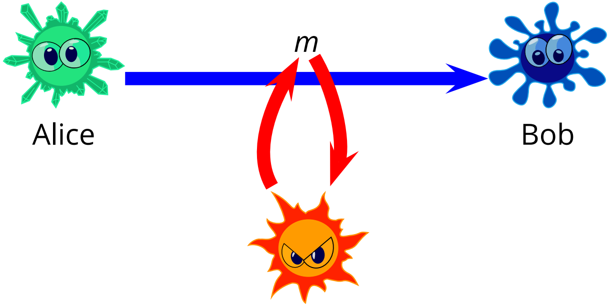
Goals



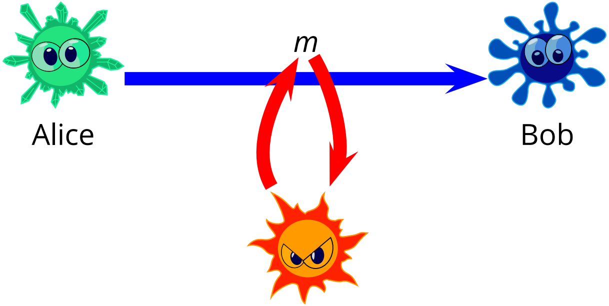
- ***Confidentiality (a.k.a., privacy):*** Alice wants to make sure that only Bob sees the message
- ***Message Integrity:*** Bob wants to make sure that the message he reads was exactly what Alice wrote

Goals (2)


m
Alice
Bob



- ***End-point Authentication:*** Bob wants to make sure he is communicating with Alice



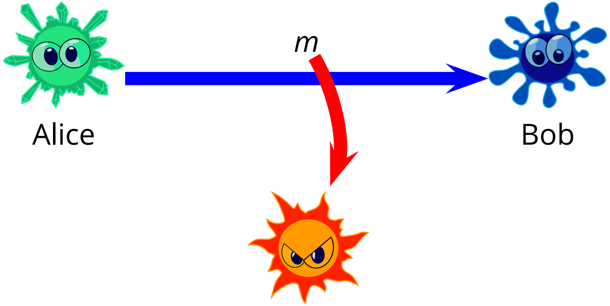
- ***End-point Authentication:*** Bob wants to make sure he is communicating with Alice
- ***Operational/system security:*** Alice and Bob want to maintain full control of their networks

privacy

authentication

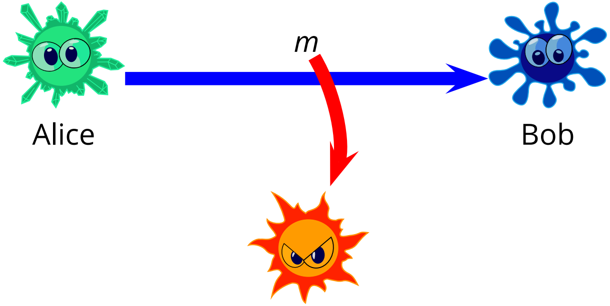
What is Privacy, Exactly?

What is Privacy, Exactly?



- Alice wants to make sure that ***only Bob "sees" message*** *m*

What is Privacy, Exactly?



- Alice wants to make sure that ***only Bob "sees" message*** m
- What if Eve can *guess* the message?

“Shift” Cipher

"Shift" Cipher

- The ciphertext is

  `BUUBDL BU EBXO`

“Shift” Cipher

- The ciphertext is

```
BUUBDL BU EBXO
```

- Plaintext is

```
ATTACK AT DAWN
```

"Shift" Cipher

- The ciphertext is

```
BUUBDL BU EBXO
```

- Plaintext is

```
ATTACK AT DAWN
```

- How many possible ciphers?
  - How many key bits?

Substitution Cipher

- *Substitution cipher*

- *Substitution cipher*
  - alphabet $\Sigma = \{\text{'A'}, \text{'B'}, \ldots, \text{'Z'}, \text{' '}\}$

- *Substitution cipher*
  - alphabet $\Sigma = \{\text{'A'}, \text{'B'}, \ldots, \text{'Z'}, \text{' '}\}$
  - encryption function: a ***permutation***

$$E : \Sigma \longrightarrow \Sigma$$

Substitution Cipher

- *Substitution cipher*
  - alphabet $\Sigma = \{\text{'A'}, \text{'B'}, \ldots, \text{'Z'}, \text{' '}\}$
  - encryption function: a ***permutation***

$$E : \Sigma \longrightarrow \Sigma$$

Example:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	_
V	Z	L	Q	X	T	_	R	D	U	C	O	J	N	F	M	G	E	H	W	P	I	S	Y	A	B	K

- *Substitution cipher*
  - alphabet $\Sigma = \{\text{'A'}, \text{'B'}, \ldots, \text{'Z'}, \text{' '}\}$
  - encryption function: a ***permutation***

$$E : \Sigma \longrightarrow \Sigma$$

Example:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z _
V Z L Q X T _ R D U C O J N F M G E H W P I S Y A B K
```

How many possible permutations?

- *Substitution cipher*
  - alphabet $\Sigma = \{\text{'A'}, \text{'B'}, \ldots, \text{'Z'}, \text{' '}\}$
  - encryption function: a ***permutation***

$$E : \Sigma \longrightarrow \Sigma$$

Example:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	_
V	Z	L	Q	X	T	_	R	D	U	C	O	J	N	F	M	G	E	H	W	P	I	S	Y	A	B	K

How many possible permutations?

27!

- *Substitution cipher*
  - alphabet $\Sigma = \{\text{'A'}, \text{'B'}, \ldots, \text{'Z'}, \text{' '}\}$
  - encryption function: a ***permutation***

$$E : \Sigma \longrightarrow \Sigma$$

Example:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z _
V Z L Q X T _ R D U C O J N F M G E H W P I S Y A B K
```

How many possible permutations?

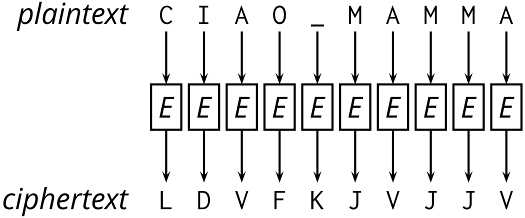
$$27! = 10888869450418352160768000000 \approx 2^{93}$$

- Encrypting some text using a substitution cipher

plaintext `C I A O _ M A M M A`

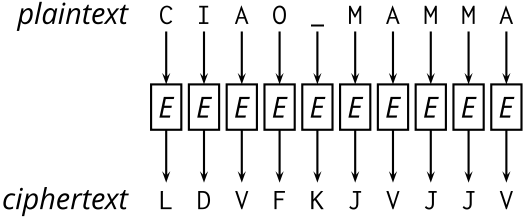
Substitution Cipher

- Encrypting some text using a substitution cipher



- Problems?

- Encrypting some text using a substitution cipher



- Problems?
  - easy to break just by *guessing!*
  - . . .

- Decrypt this ciphertext obtained by encrypting an English text with a substitution-cipher:

  gbafoduayfbhbayvpyfhayoanbahbdl-brcubqyayfkyakddaibqakvbaxvbkybuabzpkd
  yfkyayfbwakvbabquogbuanwayfbcvaxvbkyovagcyfaxbvykcqapqkdcbqkndbavctfyh
  yfkyakioqtayfbhbakvbadclbadcnbvywakquayfbampvhpcyaolafkmmcqbhh
  yfkyayoahbxpvbayfbhbavctfyhatorbvqibqyhakvbacqhycypybuakioqtaibq
  ubvcrcqtayfbcvajphyamogbvhalvoiayfbaxoqhbqyaolayfbatorbvqbu
  yfkyagfbqbrbvakqwaloviaolatorbvqibqyanbxoibhaubhyvpxycrbaolayfbhbabquh
  cyachayfbavctfyaolayfbambomdbayoakdybvaovayoaknodchfacy

From Black Magic to Mathematics

History: secret algorithms, poorly undestood security properties

From Black Magic to Mathematics

History: secret algorithms, poorly undestood security properties

Modern cryptology

- Open and clear models
- Open algorithms (the only secret part is the key material)
- Well-defined *provable* security properties

What is “Provable” Security?

The old way

What is “Provable” Security?

The old way

1. somebody (re-)designs a cryptosystem or protocol

What is “Provable” Security?

The old way

1. somebody (re-)designs a cryptosystem or protocol

2. somebody brakes it

What is “Provable” Security?

The old way

1. somebody (re-)designs a cryptosystem or protocol
2. somebody brakes it
3. go back to step 1

What is “Provable” Security?

The new way (provable security)

What is "Provable" Security?

The new way (provable security)

1. Define formal ***security goals*** and ***adversarial models***

What is "Provable" Security?

The new way (provable security)

1. Define formal ***security goals*** and ***adversarial models***
2. Design a few ***primitives***
   - based on ***public*** and ***time-tested algorithms*** and/or well-studied ***hard mathematical problems***

What is “Provable” Security?

The new way (provable security)

1. Define formal ***security goals*** and ***adversarial models***

2. Design a few ***primitives***

    - based on ***public*** and ***time-tested algorithms*** and/or well-studied ***hard mathematical problems***

3. Design a ***protocol*** (using primitives) with a ***proof of security***

What is "Provable" Security?

The new way (provable security)

1. Define formal ***security goals*** and ***adversarial models***
2. Design a few ***primitives***
   - based on ***public*** and ***time-tested algorithms*** and/or well-studied ***hard mathematical problems***
3. Design a ***protocol*** (using primitives) with a ***proof of security***
   - prove this implication:

$$\textit{primitive is secure} \Rightarrow \textit{protocol is secure}$$

Models and Goals: Define a *Game*

Example: Communicating *one bit* securely: Alice must communicate to Bob whether or not she wants to go out for dinner

Models and Goals: Define a *Game*

Example: Communicating *one bit* securely: Alice must communicate to Bob whether or not she wants to go out for dinner

- Setup:
  - Alice and Bob can meet securely and prepare for the game before the game starts
  - Eve knows that Alice agrees to go out to dinner 60% of the times

Models and Goals: Define a *Game*

Example: Communicating *one bit* securely: Alice must communicate to Bob whether or not she wants to go out for dinner

- Setup:
  - Alice and Bob can meet securely and prepare for the game before the game starts
  - Eve knows that Alice agrees to go out to dinner 60% of the times
- Adversary model:
  - Eve is a *passive adversary*
  - Eve has *unlimited computational power*

Models and Goals: Define a *Game*

Example: Communicating *one bit* securely: Alice must communicate to Bob whether or not she wants to go out for dinner

- Setup:
  - Alice and Bob can meet securely and prepare for the game before the game starts
  - Eve knows that Alice agrees to go out to dinner 60% of the times
- Adversary model:
  - Eve is a *passive adversary*
  - Eve has *unlimited computational power*
- Goal:
  - Eve wins the game if she can guess Alice's answer with probability better than 60%

Other Adversarial Models

- Adversary model:
  - Eve is an *active adversary*
  - Eve may also trick you into encrypting some chosen plaintext
  - Eve is computationally limited: she is an ordinary *algorithm*

How to Provably Win the Game

How to Provably Win the Game

game

How to Provably Win the Game

security protocol $\Longrightarrow$ ***game***

How to Provably Win the Game

primitive $\Longrightarrow$ ***security protocol*** $\Longrightarrow$ ***game***

hard math problem $\Longrightarrow$ ***primitive*** $\Longrightarrow$ ***security protocol*** $\Longrightarrow$ ***game***

Example: Factoring $\Rightarrow$ RSA $\Rightarrow$ SSH $\Rightarrow$ Privacy

Let $N = pq$ for two large prime factors p and q

Problem: given N, find p and q

Example: Factoring $\Rightarrow$ RSA $\Rightarrow$ SSH $\Rightarrow$ Privacy

Let $N = pq$ for two large prime factors p and q

Problem: given N, find p and q

Solution: (trivial)

```
FACTOR(N)
1  for i ← 2 to ⌊√N⌋
2      do if i divides N
3          then return i, N/i
```

Example: Factoring ⇒ RSA ⇒ SSH ⇒ Privacy

Let $N = pq$ for two large prime factors p and q

Problem: given N, find p and q

Solution: (trivial)

```
FACTOR(N)
for i ← 2 to ⌊√N⌋
    do if i divides N
        then return i, N/i
```

Complexity: exponential in the *size* of N (number of digits of N)

Example: Factoring ⇒ RSA ⇒ SSH ⇒ Privacy

Let $N = pq$ for two large prime factors p and q

Problem: given N, find p and q

Solution: (trivial)

```
FACTOR(N)
for i ← 2 to ⌊√N⌋
      do if i divides N
            then return i, N/i
```

Complexity: exponential in the *size* of N (number of digits of N)

. . . we don't know how to do better!

Not even Gauss could figure that out!

From Factoring to RSA—in One Page!

From Factoring to RSA—in One Page!

- Pick two large primes p and q and let $N = pq$

From Factoring to RSA—in One Page!

- Pick two large primes p and q and let $N = pq$
- Consider the group $\mathbb{Z}_N^*$

From Factoring to RSA—in One Page!

- Pick two large primes p and q and let $N = pq$
- Consider the group $\mathbb{Z}_N^*$
  - notice that N is not prime
  - the order of $\mathbb{Z}_N^*$ is $|\mathbb{Z}_N^*| = \phi(N) = (p-1)(q-1)$

From Factoring to RSA—in One Page!

- Pick two large primes p and q and let $N = pq$

- Consider the group $\mathbb{Z}_N^*$
  - notice that N is not prime
  - the order of $\mathbb{Z}_N^*$ is $|\mathbb{Z}_N^*| = \phi(N) = (p-1)(q-1)$

- Pick a small odd integer e and compute $d \equiv e^{-1} \mod \phi(N)$

From Factoring to RSA—in One Page!

- Pick two large primes p and q and let $N = pq$
- Consider the group $\mathbb{Z}_N^*$
  - notice that N is not prime
  - the order of $\mathbb{Z}_N^*$ is $|\mathbb{Z}_N^*| = \phi(N) = (p-1)(q-1)$
- Pick a small odd integer e and compute $d \equiv e^{-1} \mod \phi(N)$
  - your ***public key*** is (N, e)

From Factoring to RSA—in One Page!

- Pick two large primes p and q and let $N = pq$
- Consider the group $\mathbb{Z}_N^*$
  - notice that N is not prime
  - the order of $\mathbb{Z}_N^*$ is $|\mathbb{Z}_N^*| = \phi(N) = (p-1)(q-1)$
- Pick a small odd integer e and compute $d \equiv e^{-1} \mod \phi(N)$
  - your ***public key*** is (N, e)
  - your ***secret key*** is (N, d)

From Factoring to RSA—in One Page!

- Pick two large primes p and q and let $N = pq$
- Consider the group $\mathbb{Z}_N^*$
  - notice that N is not prime
  - the order of $\mathbb{Z}_N^*$ is $|\mathbb{Z}_N^*| = \phi(N) = (p-1)(q-1)$
- Pick a small odd integer e and compute $d \equiv e^{-1} \mod \phi(N)$
  - your ***public key*** is (N, e)
  - your ***secret key*** is (N, d)
- ***Encryption:*** $C \leftarrow \mathrm{RSA}_{N,e}(M) := M^e \mod N$

From Factoring to RSA—in One Page!

- Pick two large primes p and q and let $N = pq$
- Consider the group $\mathbb{Z}_N^*$
  - notice that N is not prime
  - the order of $\mathbb{Z}_N^*$ is $|\mathbb{Z}_N^*| = \phi(N) = (p-1)(q-1)$
- Pick a small odd integer e and compute $d \equiv e^{-1} \mod \phi(N)$
  - your ***public key*** is (N, e)
  - your ***secret key*** is (N, d)
- ***Encryption:*** $C \leftarrow \mathrm{RSA}_{N,e}(M) := M^e \mod N$
- ***Decryption:*** $M \leftarrow \mathrm{RSA}_{N,d}(C) := C^d \mod N$

Factoring $\Rightarrow$ RSA $\Rightarrow$ SSH

Factoring $\Rightarrow$ RSA $\Rightarrow$ SSH

- Is RSA secure?

- Is RSA secure? *We don't know!*

Factoring $\Rightarrow$ RSA $\Rightarrow$ SSH

- Is RSA secure? *We don't know!* But. . .

Factoring $\Rightarrow$ RSA $\Rightarrow$ SSH

- Is RSA secure? *We don't know!* But. . .

  if you can break RSA (efficiently) ***then*** you can also factor a product of two large primes (efficiently)

  . . . which means that you are smarter than Gauss!

Factoring $\Rightarrow$ RSA $\Rightarrow$ SSH

- Is RSA secure? *We don't know!* But. . .

  if you can break RSA (efficiently) ***then*** you can also factor a product of two large primes (efficiently)

  . . . which means that you are smarter than Gauss!

- SSH uses the RSA public-key system (possibly, not only)
- Is SSH secure?

Factoring ⇒ RSA ⇒ SSH

- Is RSA secure? *We don't know!* But...

  if you can break RSA (efficiently) ***then*** you can also factor a product of two large primes (efficiently)

  ...which means that you are smarter than Gauss!

- SSH uses the RSA public-key system (possibly, not only)
- Is SSH secure? *Yes!*

Factoring $\Rightarrow$ RSA $\Rightarrow$ SSH

- Is RSA secure? *We don't know!* But...

  if you can break RSA (efficiently) ***then*** you can also factor a product of two large primes (efficiently)

  ...which means that you are smarter than Gauss!

- SSH uses the RSA public-key system (possibly, not only)
- Is SSH secure? *Yes!*

  ...in the sense that, somebody *proved* the implication:

$$\text{RSA is secure} \Rightarrow \text{SSH is secure}$$

- Is RSA secure? *We don't know!* But. . .

  if you can break RSA (efficiently) ***then*** you can also factor a product of two large primes (efficiently)

  . . . which means that you are smarter than Gauss!

- SSH uses the RSA public-key system (possibly, not only)
- Is SSH secure? *Yes!*

  . . . in the sense that, somebody *proved* the implication:

  $$\text{RSA is secure} \Longrightarrow \text{SSH is secure}$$

  so (counter-positive) if you can break SSH then you can also break RSA

  . . . and you are smarter than Gauss!

The Big Picture

- Basic ingredients: cryptographic primitives
  - secret-key (symmetric) cryptography (e.g., AES)
  - public-key (asymmetric) cryptography (e.g., RSA)
  - cryptographic hash functions (e.g., SHA-1)
  - stream ciphers (e.g., RC4)

The Big Picture

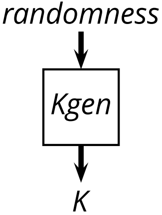
- Basic ingredients: cryptographic primitives
  - secret-key (symmetric) cryptography (e.g., AES)
  - public-key (asymmetric) cryptography (e.g., RSA)
  - cryptographic hash functions (e.g., SHA-1)
  - stream ciphers (e.g., RC4)

- Recipes: cryptographic protocols
  - certificates (e.g., X.509)
  - secure transport (e.g., TLS, IPSec)
  - ...

- Basic ingredients: cryptographic primitives
  - secret-key (symmetric) cryptography (e.g., AES)
  - public-key (asymmetric) cryptography (e.g., RSA)
  - cryptographic hash functions (e.g., SHA-1)
  - stream ciphers (e.g., RC4)
- Recipes: cryptographic protocols
  - certificates (e.g., X.509)
  - secure transport (e.g., TLS, IPSec)
  - ...
- Applications
  - electronic commerce
  - secure shell
  - secure electronic mail
  - virtual private networks
  - ...

Symmetric Encryption

Symmetric Encryption

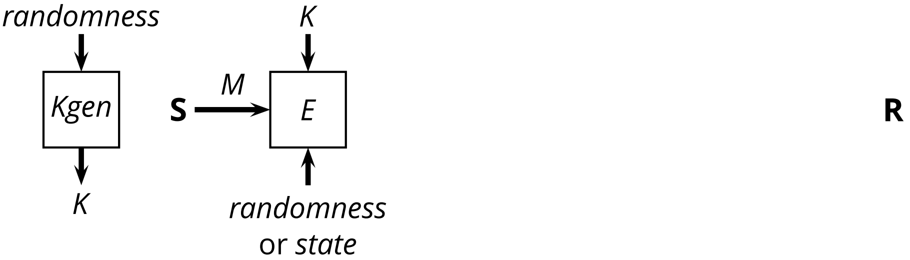


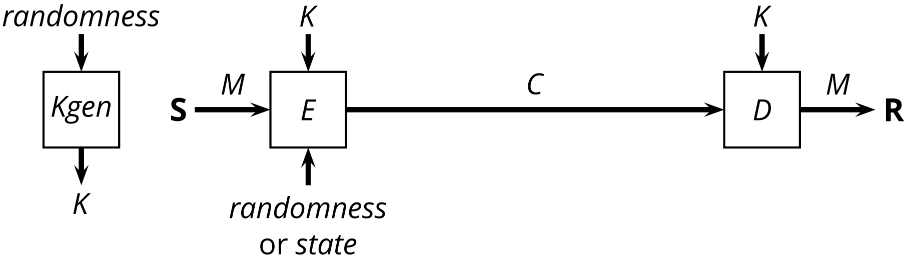

randomness
Kgen
K
S
R

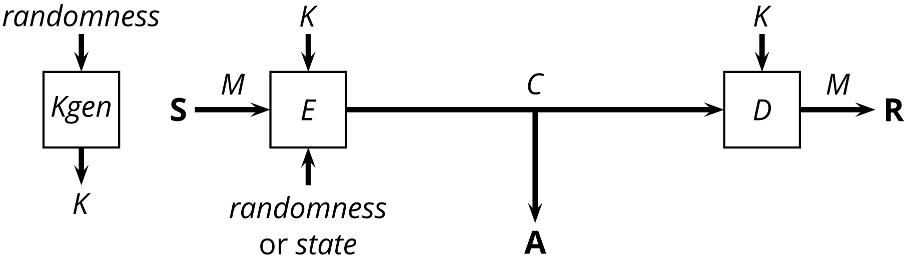

randomness
Kgen
K
S
M
E
R

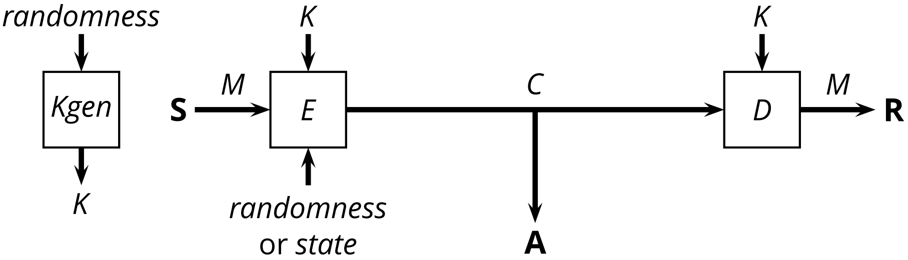

randomness
Kgen
K
S
M
K
E
R

Symmetric Encryption




randomness
Kgen
K
S
M
K
E
randomness
or state
C
K
D
M
R


randomness
Kgen
K
S
M
K
E
randomness
or state
C
A
K
D
M
R



S	sender
R	receiver
A	adversary
E	encryption algorithm
D	dencryption algorithm
M	plaintext message
C	ciphertext message
K	key

One-Time Pad

- *Assumptions:* the message M and the key K are two n-bit strings

$$M \in \{0,1\}^n; \qquad K \stackrel{\$}{\longleftarrow} \{0,1\}^n$$

- *Assumptions:* the message M and the key K are two n-bit strings

$$M \in \{0,1\}^n; \qquad K \xleftarrow{\$} \{0,1\}^n$$

the key K is chosen uniformly at random from $\{0,1\}^n$

- *Assumptions:* the message M and the key K are two n-bit strings

$$M \in \{0,1\}^n; \qquad K \xleftarrow{\$} \{0,1\}^n$$

  the key K is chosen uniformly at random from $\{0,1\}^n$

- *Scheme*
  - encryption:

$$E(K, M) := M \oplus K$$

    the key K is then thrown away an never reused

  - decryption:

$$D(K, C) := C \oplus K$$

- *Assumptions:* the message M and the key K are two n-bit strings

$$M \in \{0,1\}^n; \qquad K \xleftarrow{\$} \{0,1\}^n$$

  the key K is chosen uniformly at random from $\{0,1\}^n$

- *Scheme*
  - encryption:

$$E(K, M) := M \oplus K$$

    the key K is then thrown away an never reused
  - decryption:

$$D(K, C) := C \oplus K$$

- **Example:** M 0110010110111011

One-Time Pad

- *Assumptions:* the message M and the key K are two n-bit strings

$$M \in \{0,1\}^n; \qquad K \stackrel{\$}{\longleftarrow} \{0,1\}^n$$

  the key K is chosen uniformly at random from $\{0,1\}^n$

- *Scheme*
  - encryption:
  $$E(K, M) := M \oplus K$$
  the key K is then thrown away an never reused
  - decryption:
  $$D(K, C) := C \oplus K$$

- **Example:**

M	0110010110111011
K	1011000101000101

- *Assumptions:* the message M and the key K are two n-bit strings

$$M \in \{0,1\}^n; \qquad K \xleftarrow{\$} \{0,1\}^n$$

  the key K is chosen uniformly at random from $\{0,1\}^n$

- *Scheme*
  - encryption:

$$E(K, M) := M \oplus K$$

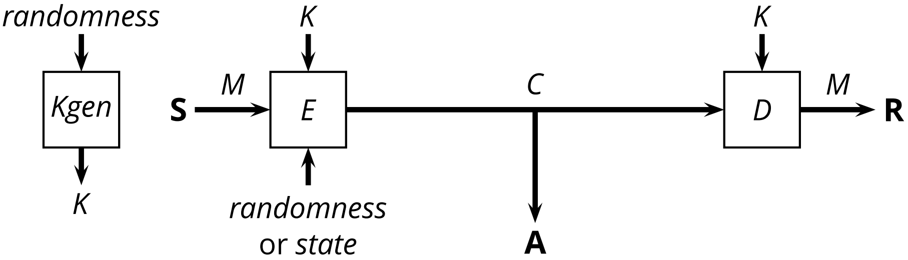
    the key K is then thrown away an never reused
  - decryption:

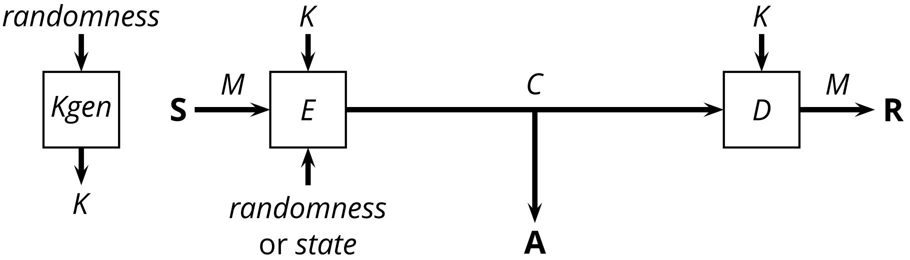
$$D(K, C) := C \oplus K$$

- **Example:**

M	0110010110111011
K	1011000101000101
C	1101010011111110

Symmetric Encryption (2)





S	sender
R	receiver
A	adversary
E	encryption algorithm
D	dencryption algorithm
M	plaintext message
C	ciphertext message
K	key

Rules of the game:

- *Kgen*, *E* and *D* are *public* algorithms
- **A** can not "steal" the key *K*
- **A** can not "break into" **S** or **R**
- **A** might know something about *M*

A must guess *M* correctly to win the game

So, What is Privacy Exactly?

So, What is Privacy Exactly?

- A scheme is secure if ***we learn nothing from the ciphertext*** C

So, What is Privacy Exactly?

- A scheme is secure if ***we learn nothing from the ciphertext*** C

- A more formal definition:

  let $K \xleftarrow{\$} \mathcal{K}$; for every $m_1 \neq m_2 \in \mathcal{M}$, and for any C

So, What is Privacy Exactly?

- A scheme is secure if ***we learn nothing from the ciphertext*** C

- A more formal definition:

  let $K \xleftarrow{\$} \mathcal{K}$; for every $m_1 \neq m_2 \in \mathcal{M}$, and for any C

$$\Pr_{K \in \mathcal{K}}[E_K(m_1) = C] = \Pr_{K \in \mathcal{K}}[E_K(m_2) = C]$$

So, What is Privacy Exactly?

- A scheme is secure if ***we learn nothing from the ciphertext*** C
- A more formal definition:

  let $K \xleftarrow{\$} \mathcal{K}$; for every $m_1 \neq m_2 \in \mathcal{M}$, and for any C

$$\Pr_{K \in \mathcal{K}}[E_K(m_1) = C] = \Pr_{K \in \mathcal{K}}[E_K(m_2) = C]$$

- ***Given a ciphertext*** C***, every plaintext*** m ***is equiprobable***
  - so, seeing any particular $C = E_K(m)$ tells us *nothing* about m

So, What is Privacy Exactly?

- A scheme is secure if ***we learn nothing from the ciphertext*** C
- A more formal definition:

  let $K \xleftarrow{\$} \mathcal{K}$; for every $m_1 \neq m_2 \in \mathcal{M}$, and for any C

$$\Pr_{K \in \mathcal{K}}[E_K(m_1) = C] = \Pr_{K \in \mathcal{K}}[E_K(m_2) = C]$$

- ***Given a ciphertext*** C***, every plaintext*** m ***is equiprobable***
  - so, seeing any particular $C = E_K(m)$ tells us *nothing* about m
- Is a shift cipher perfectly secure?

So, What is Privacy Exactly?

- A scheme is secure if ***we learn nothing from the ciphertext*** C
- A more formal definition:

  let $K \xleftarrow{\$} \mathcal{K}$; for every $m_1 \neq m_2 \in \mathcal{M}$, and for any C

$$\Pr_{K \in \mathcal{K}}[E_K(m_1) = C] = \Pr_{K \in \mathcal{K}}[E_K(m_2) = C]$$

- ***Given a ciphertext*** C***, every plaintext*** m ***is equiprobable***
  - so, seeing any particular $C = E_K(m)$ tells us *nothing* about m
- Is a shift cipher perfectly secure?
- Is a substitution cipher perfectly secure?

So, What is Privacy Exactly?

- A scheme is secure if ***we learn nothing from the ciphertext*** C
- A more formal definition:

  let $K \xleftarrow{\$} \mathcal{K}$; for every $m_1 \neq m_2 \in \mathcal{M}$, and for any C

$$\Pr_{K \in \mathcal{K}}[E_K(m_1) = C] = \Pr_{K \in \mathcal{K}}[E_K(m_2) = C]$$

- ***Given a ciphertext*** C***, every plaintext*** m ***is equiprobable***
  - so, seeing any particular $C = E_K(m)$ tells us *nothing* about m
- Is a shift cipher perfectly secure?
- Is a substitution cipher perfectly secure?
- Is one-time-pad perfectly secure?

The Cost of Perfect Privacy

The Cost of Perfect Privacy

- *Perfect privacy* implies that

$$|\mathcal{K}| \geq |\mathcal{M}|$$

- *Perfect privacy* implies that

$$|\mathcal{K}| \geq |\mathcal{M}|$$

- *Proof:* assume not.
  Fix a possible ciphertext C, i.e., there is a message m and a key k such that $E_K(m) = C$, and $\Pr_{K \in \mathcal{K}}[E_K(m) = C] > 0$

  Let $P_C = \{m \in \mathcal{M}$ such that $E_k(m) = C$ for some $k\}$

  Since every k maps exactly one message m to C, and since we have fewer keys than messages, then there is an $m' \notin P_C$ such that no key k maps m' to C; therefore $\Pr_{K \in \mathcal{K}}[E_K(m') = C] = 0$, which violates the perfect-secrecy condition that for all m and m', $\Pr_{K \in \mathcal{K}}[E_K(m) = C] = \Pr_{K \in \mathcal{K}}[E_K(m') = C]$

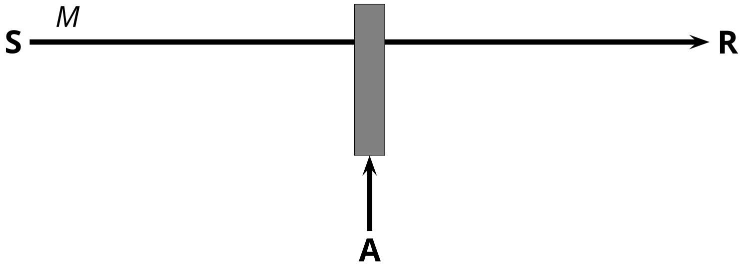
Message Authenticity

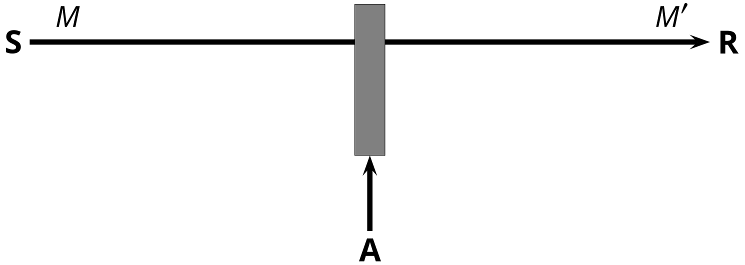
Message Authenticity

S

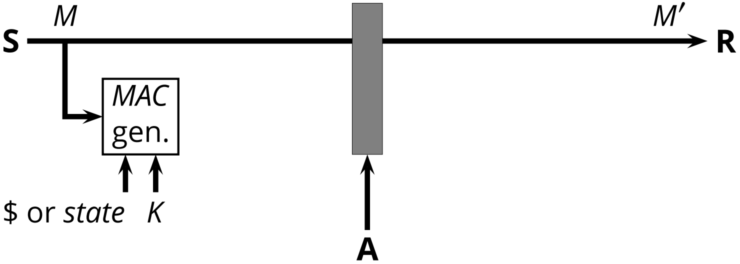
R

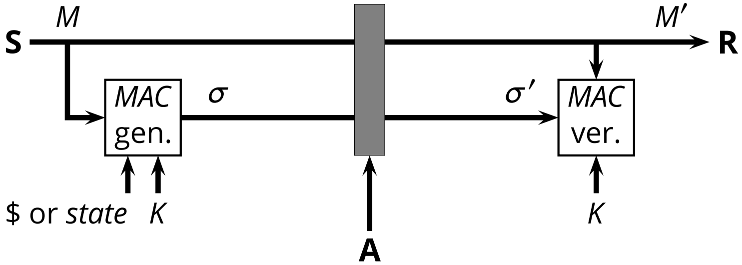

M
S
R


S
M
R
A

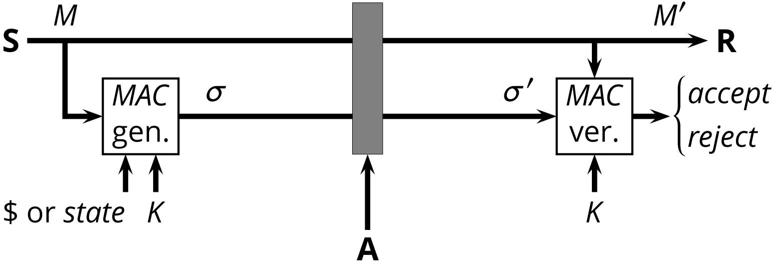

S
M
M′
R
A

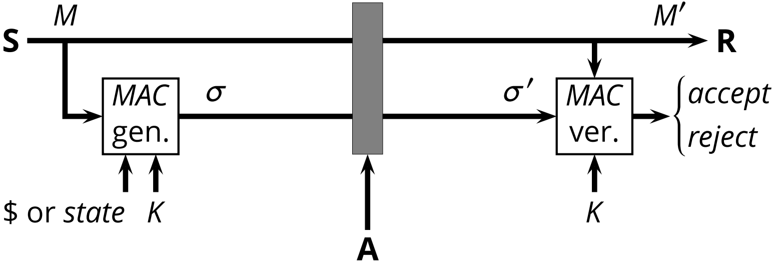
Message Authenticity




S
M
M′
R
MAC gen.
σ
σ′
MAC ver.
$ or state
K
K
A

Message Authenticity





σ	message authentication code (MAC)
K	key
\$	randomness
MAC gen.	MAC *generation* algorithm
MAC ver.	MAC *verification* algorithm

Asymmetric Encryption

Asymmetric Encryption

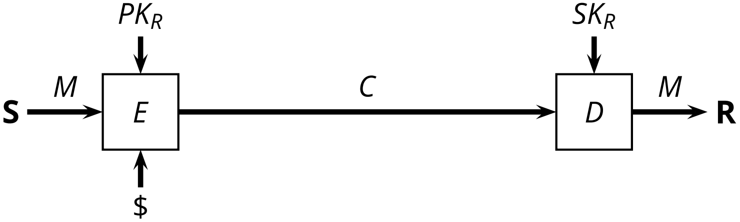
S

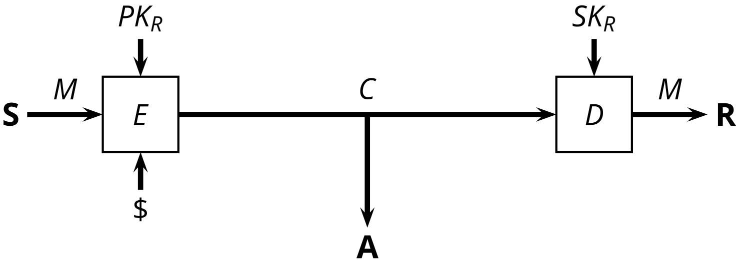
R


S
M
E
R

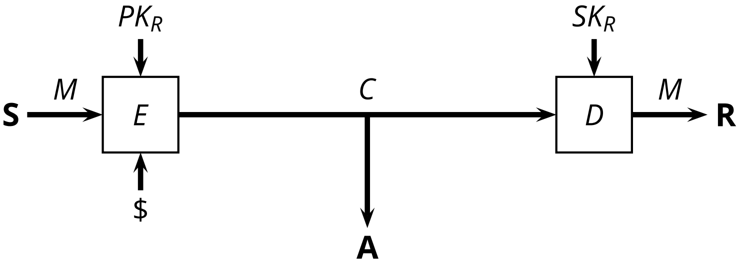
Asymmetric Encryption


PK_R
M
S
E
$
R

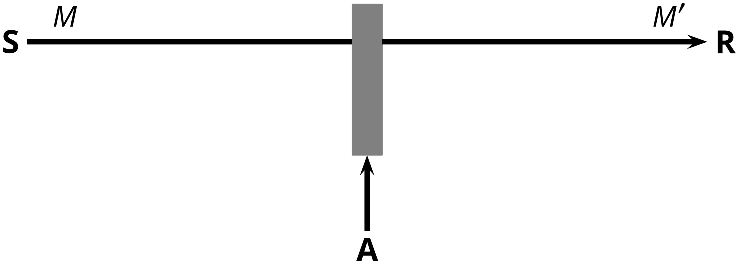

PK_R
SK_R
S
M
E
C
D
M
R
$

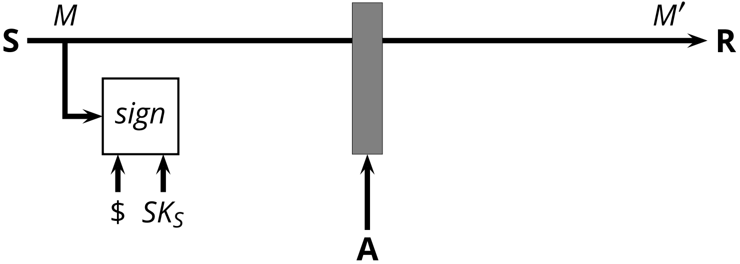


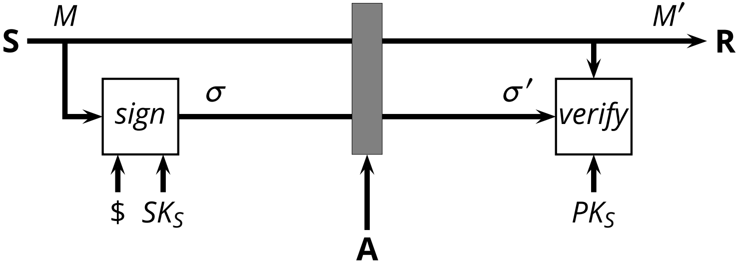
PK_R
SK_R
S
M
E
C
D
M
R
$
A

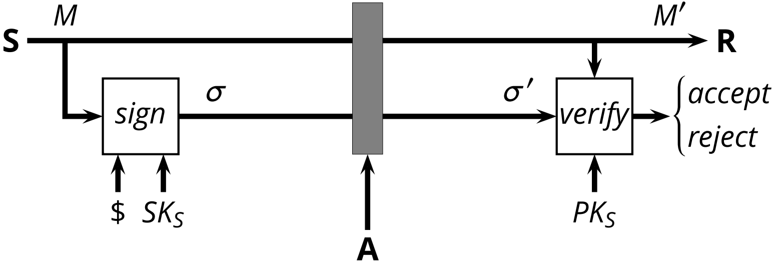


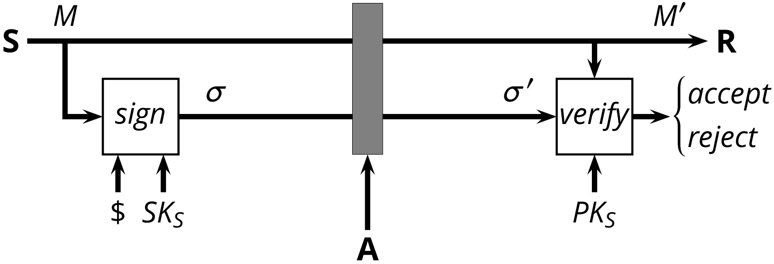
PK_R	receiver's *public key*
SK_R	receiver's *secret key*
M	plaintext message
C	ciphertext message


M
S
M′
R
A


S
M
M′
R
sign
$
SK_S
A


S
M
R
M′
sign
σ
σ′
verify
$
SK_S
PK_S
A


S
M
R
M′
sign
σ
σ′
verify
accept
reject
$
SK_S
PK_S
A



σ	digital signature
SK_S	sender's *secret key*
PK_S	sender's *public key*
$	randomness
sign	*signing* algorithm
verify	*verification* algorithm

Primitives vs. Protocols

Primitives vs. Protocols

- ***Protocol***
  - an *algorithm*
  - solves a specific security problem (e.g., signing a message)

Primitives vs. Protocols

- ***Protocol***
  - an *algorithm*
  - solves a specific security problem (e.g., signing a message)
- ***Primitive***

Primitives vs. Protocols

- ***Protocol***
  - an *algorithm*
  - solves a specific security problem (e.g., signing a message)

- ***Primitive***
  - also an *algorithm*
  - the elementary subroutines of protocols
  - implement (try to approximate) well-defined mathematical object
  - embody “hard problems”

Stream Ciphers

- A *stream cipher* is a generator of a *pseudo-random streams*

Stream Ciphers

- A *stream cipher* is a generator of a *pseudo-random streams*
  - given an initialization key K
  - generates an infinite pseudo-random sequence of bits

- A *stream cipher* is a generator of a *pseudo-random streams*
  - given an initialization key *K*
  - generates an infinite pseudo-random sequence of bits
- E.g., RC4

Padding with a Stream Cipher

Padding with a Stream Cipher

- *Assumptions:* S and R share a secret key K and agree to use a stream cipher S_K
  - S and R maintain some *state:* position s initialized to $s = 0$

Padding with a Stream Cipher

- *Assumptions:* S and R share a secret key K and agree to use a stream cipher S_K
  - S and R maintain some *state:* position s initialized to $s = 0$

- *Encryption protocol*

1. S computes $C \leftarrow M \oplus S_K[s \ldots s + |M| - 1]$
2. S updates its position $s \leftarrow s + |M|$

Padding with a Stream Cipher

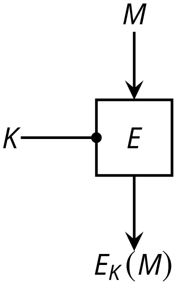
- *Assumptions:* S and R share a secret key K and agree to use a stream cipher S_K
  - S and R maintain some *state:* position s initialized to $s = 0$

- *Encryption protocol*
  1. S computes $C \leftarrow M \oplus S_K[s \ldots s + |M| - 1]$
  2. S updates its position $s \leftarrow s + |M|$

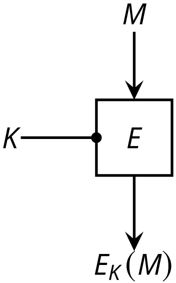
- *Dencryption protocol*
  1. R computes $M \leftarrow C \oplus S_K[s \ldots s + |C| - 1]$
  2. R updates its position $s \leftarrow s + |C|$

- *Block Cipher:* $E : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$

- *Block Cipher:* $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$

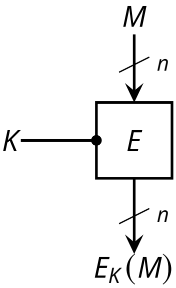


- *Block Cipher:* $E : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$



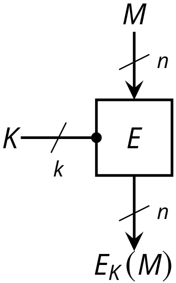
  - $E_K(\cdot)$ is a *permutation*, so $E_K^{-1}(\cdot)$ is always defined

- *Block Cipher:* $E : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$



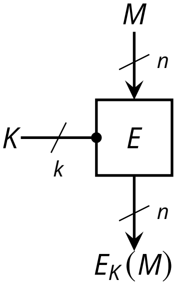
  - $E_K(\cdot)$ is a *permutation*, so $E_K^{-1}(\cdot)$ is always defined
  - fixed-length input and output (n)

- *Block Cipher:* $E : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$



- $E_K(\cdot)$ is a *permutation*, so $E_K^{-1}(\cdot)$ is always defined
- fixed-length input and output (n)
- fixed-length key (k)

- *Block Cipher:* $E : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$



  - $E_K(\cdot)$ is a *permutation*, so $E_K^{-1}(\cdot)$ is always defined
  - fixed-length input and output (n)
  - fixed-length key (k)
  - e.g., DES, AES

An Encryption Protocol

- *Symmetric encryption*
  - *Input:* k-bit key K, N-bit message M
  - *Output:* N-bit ciphertext C

- *Symmetric encryption*
  - *Input:* k-bit key K, N-bit message M
  - *Output:* N-bit ciphertext C
- *Cipher Block Chaining (CBC)*
  - use a block cipher $E : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$
  - split M into n-bit blocks $M = M_0 || M_1 || \ldots || M_\ell \quad (\ell = \lfloor N/n \rfloor)$

An Encryption Protocol

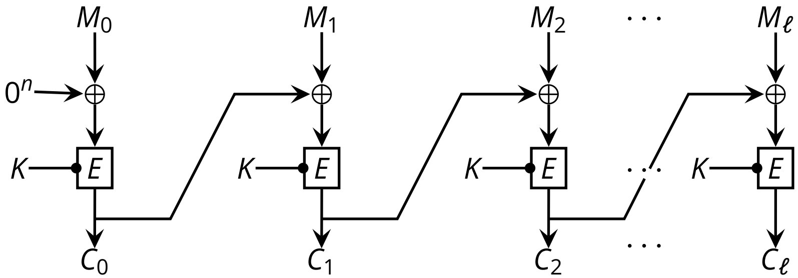
- *Symmetric encryption*
  - *Input:* k-bit key K, N-bit message M
  - *Output:* N-bit ciphertext C

- *Cipher Block Chaining (CBC)*
  - use a block cipher $E : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$
  - split M into n-bit blocks $M = M_0 || M_1 || \ldots || M_\ell \quad (\ell = \lfloor N/n \rfloor)$

```
CBC(K, M)
x ← 0^n
for i ← 0 to ⌊|M|/n⌋
        do C[ni . . . ni + n − 1] ← E_K(x ⊕ M[ni . . . ni + n − 1])
           x ← C[ni . . . ni + n − 1]
return C
```

- *Symmetric encryption*
  - *Input:* k-bit key K, N-bit message M
  - *Output:* N-bit ciphertext C

- *Cipher Block Chaining (CBC)*
  - use a block cipher $E : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$
  - split M into n-bit blocks $M = M_0 || M_1 || \ldots || M_\ell \quad (\ell = \lfloor N/n \rfloor)$



Exercise

- Write the decryption algorithm for CBC

Exercise

- Write the decryption algorithm for CBC

CBC-DECRYPT(K, C)

$x \leftarrow 0^n$
for $i \leftarrow 0$ **to** $\lfloor |C|/n \rfloor$
    do $M[ni \ldots ni+n-1] \leftarrow x \oplus E_K^{-1}(C[ni \ldots ni+n-1])$
        $x \leftarrow C[ni \ldots ni+n-1]$
return M

An Encryption Protocol (2)

An Encryption Protocol (2)

- Is this CBC protocol secure?

An Encryption Protocol (2)

- Is this CBC protocol secure?
  - ***any deterministic stateless protocol is insecure***
  - we need *state* and/or *randomness*

An Encryption Protocol (2)

- Is this CBC protocol secure?
  - ***any deterministic stateless protocol is insecure***
  - we need *state* and/or *randomness*
- What if $|M| \neq 0 \mod n$?

An Encryption Protocol (2)

- Is this CBC protocol secure?
  - ***any deterministic stateless protocol is insecure***
  - we need *state* and/or *randomness*
- What if $|M| \neq 0 \mod n$?
- Is CBC parallelizable?

CBC With Random IV

- ***CBC$:*** cipher block chaining with random IV

CBC With Random IV

- ***CBC$:*** cipher block chaining with random IV

CBC$-ENCRYPT$(K, M)$

if $|M| = 0 \vee |M| \neq 0 \mod n$
  then return $\perp$
$M[1] \cdot M[2] \cdots M[\ell] \leftarrow M$
$IV \xleftarrow{\$} \{0,1\}^n$
$C[0] \leftarrow IV$
for $i \leftarrow 1$ **to** ℓ
  do $C[i] \leftarrow E_K(C[i-1] \oplus M[i])$
$C \leftarrow C[1] \cdot C[2] \cdots C[\ell]$
return $\langle IV, C \rangle$

CBC With Random IV (2)

- ***CBC$:*** cipher block chaining with random IV (decryption)

CBC With Random IV (2)

- ***CBC$:*** cipher block chaining with random IV (decryption)

CBC$-DECRYPT$(K, IV, C)$

if $|C| = 0 \vee |C| \neq 0 \mod n$
    then return $\perp$
$C[1] \cdot C[2] \cdots C[\ell] \leftarrow C$
$C[0] \leftarrow IV$
for $i \leftarrow 1$ **to** ℓ
        do $M[i] \leftarrow C[i-1] \oplus E_K(C[i])$
$M \leftarrow M[1] \cdot M[2] \cdots M[\ell]$
return M

CBC With Stateful Counter

- ***CBCC:*** cipher block chaining with stateful counter

CBC With Stateful Counter

- ***CBCC:*** cipher block chaining with stateful counter

CBCC-Encrypt(K, M)

static $ctr \leftarrow 0$
if $ctr \geq 2^n \vee |M| = 0 \vee |M| \neq 0 \mod n$
    then return $\perp$
$M[1] \cdot M[2] \cdots M[\ell] \leftarrow M$
$IV \leftarrow [ctr]_n$
$C[0] \leftarrow [ctr]_n$
for $i \leftarrow 1$ **to** ℓ
        do $C[i] \leftarrow E_K(C[i-1] \oplus M[i])$
$C \leftarrow C[1] \cdot C[2] \cdots C[\ell]$
$ctr \leftarrow ctr + 1$
return $\langle IV, C \rangle$

CBC With Stateful Counter (2)

- ***CBCC:*** cipher block chaining with stateful counter

CBC With Stateful Counter (2)

- ***CBCC:*** cipher block chaining with stateful counter

CBCC-Decrypt(K, IV, C)

if $IV + |C| \geq 2^n \vee |C| = 0 \vee |C| \neq 0 \mod n$
    then return $\perp$
$C[1] \cdot C[2] \cdots C[\ell] \leftarrow C$
$IV \leftarrow [ctr]_n$
$C[0] \leftarrow IV$
for $i \leftarrow 1$ **to** ℓ
    do $M[i] \leftarrow C[i-1] \oplus E_K^{-1}(C[i])$
$M \leftarrow M[1] \cdot M[2] \cdots M[\ell]$
return M

- ***CTR$:*** counter mode with random initial counter

- ***CTR$:*** counter mode with random initial counter
  - *family of functions:* $F : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$

- ***CTR\$:*** counter mode with random initial counter
  - *family of functions:* $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$

CTR\$-ENCRYPT(K, M)

$R \stackrel{\$}{\longleftarrow} \{0,1\}^n$
$Pad \leftarrow F_K([R]_n)$
for $i \leftarrow 1$ **to** $\lceil |M|/n \rceil - 1$
    do $Pad \leftarrow Pad \cdot F_K([R+i]_n)$
$Pad \leftarrow$ first $|M|$ bits of Pad
$C \leftarrow M \oplus Pad$
return $\langle R, C \rangle$

- ***CTR$:*** counter mode with random initial counter (decryption)
  - *family of functions:* $F : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$

- ***CTR\$:*** counter mode with random initial counter (decryption)
  - *family of functions:* $F : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$

CTR\$-DECRYPT(K, R, C)

$Pad \leftarrow F_K([R]_n)$
for $i \leftarrow 1$ **to** $\lceil |C|/n \rceil - 1$
  do $Pad \leftarrow Pad \cdot F_K([R+i]_n)$
$Pad \leftarrow$ first $|C|$ bits of Pad
$M \leftarrow C \oplus Pad$
return M

Counter Mode (3)

- ***CTRC:*** counter mode with stateful counter
  - *family of functions:* $F : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$

Counter Mode (3)

- ***CTRC:*** counter mode with stateful counter
  - *family of functions:* $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$

CTRC(K, M)

static $R \leftarrow 0$
$\ell \leftarrow \lceil |M|/n \rceil$
if $R + \ell - 1 \geq 2^n$
    then return $\bot$
$Pad \leftarrow F_K([R]_n)$
for $i \leftarrow 1$ **to** $\ell - 1$
        do $Pad \leftarrow Pad \cdot F_K([R+i]_n)$
$Pad \leftarrow$ first $|M|$ bits of Pad
$C \leftarrow M \oplus Pad$
$R \leftarrow R + \ell$
return $\langle R - \ell, C \rangle$

Counter Mode (4)

- ***CTRC:*** counter mode with stateful counter (decryption)
  - *family of functions:* $F : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$

- ***CTRC:*** counter mode with stateful counter (decryption)
  - *family of functions:* $F : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$

$\text{CTRC-D}\textsc{ecrypt}(K, R, C)$

- $Pad \leftarrow F_K([R]_n)$
- **for** $i \leftarrow 1$ **to** $\lceil |C|/n \rceil - 1$
  - **do** $Pad \leftarrow Pad \cdot F_K([R+i]_n)$
- $Pad \leftarrow$ first $|C|$ bits of Pad
- $M \leftarrow C \oplus Pad$
- **return** M

- *MAC generation*
  - *Input:* k-bit key K, N-bit message M
  - *Output:* n-bit message authentication code σ

Authentication Protocol

- *MAC generation*
  - *Input:* k-bit key K, N-bit message M
  - *Output:* n-bit message authentication code σ

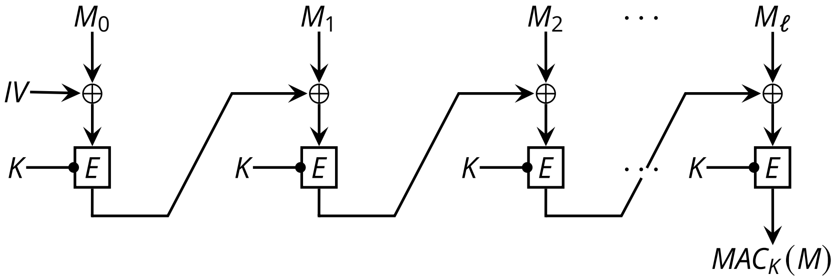
- *CBC with random IV*
  - use a block cipher $E : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$
  - split M into n-bit blocks $M = M_0||M_1||\ldots||M_\ell \quad (\ell = \lfloor N/n \rfloor)$

MAC(K, M)

$IV \xleftarrow{\$} \{0,1\}^n$
$C \leftarrow IV$
for $i \leftarrow 0$ **to** $\lfloor |M|/n \rfloor$
    do $C \leftarrow E_K(C \oplus M[ni \ldots ni + n - 1])$
return $\langle IV, C \rangle$

- *MAC generation*
  - *Input:* k-bit key K, N-bit message M
  - *Output:* n-bit message authentication code σ

- *CBC with random IV*
  - use a block cipher $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$
  - split M into n-bit blocks $M = M_0||M_1||\dots||M_\ell \quad (\ell = \lfloor N/n \rfloor)$



CBC MAC: Generation

- ***CBC MAC:*** cipher block chaining MAC with random IV

CBC MAC: Generation

- ***CBC MAC:*** cipher block chaining MAC with random IV

CBC-MAC\$$(K, M)$

if $|M| = 0 \lor |M| \neq 0 \mod n$
  then return $\perp$
$M[1] \cdot M[2] \cdots M[\ell] \leftarrow M$
$IV \xleftarrow{\$} \{0,1\}^n$
$C \leftarrow IV$
for $i \leftarrow 1$ **to** ℓ
    do $C \leftarrow E_K(C \oplus M[i])$
return $\langle IV, C \rangle$

CBC MAC: Verification

- ***CBC MAC:*** cipher block chaining MAC with random IV

CBC MAC: Verification

- ***CBC MAC:*** cipher block chaining MAC with random IV

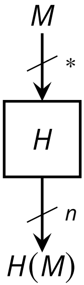
CBC-MAC\$-VERIFY(K, IV, σ, M)

if $|M| = 0 \lor |M| \neq 0 \mod n$
    then return $\perp$
$M[1] \cdot M[2] \cdots M[\ell] \leftarrow M$
$C \leftarrow IV$
for $i \leftarrow 1$ **to** ℓ
        do $C \leftarrow E_K(C \oplus M[i])$
if $C = \sigma$
    then return ACCEPT
    else return REJECT

Cryptographic Hash Functions

- *Cryptographic Hash:* $H : \{0,1\}^* \longrightarrow \{0,1\}^n$

- *Cryptographic Hash:* $H : \{0,1\}^* \longrightarrow \{0,1\}^n$



- *Cryptographic Hash:* $H : \{0,1\}^* \longrightarrow \{0,1\}^n$
  - $H(\cdot)$ is a good *hash* function when (***informally***)

$$\forall m \in \{0,1\}^*, h \in \{0,1\}^n, \Pr[H(m) = h] = \frac{1}{2^n}$$

- *Cryptographic Hash:* $H : \{0,1\}^* \longrightarrow \{0,1\}^n$
  - $H(\cdot)$ is a good *hash* function when (***informally***)

$$\forall m \in \{0,1\}^*, h \in \{0,1\}^n, \Pr[H(m) = h] = \frac{1}{2^n}$$

  - it is “difficult” to find *collisions*

$$\text{find } m_1, m_2 \in \{0,1\}^* : m_1 \neq m_2, H(m_1) = H(m_2)$$

- *Cryptographic Hash:* $H : \{0,1\}^* \longrightarrow \{0,1\}^n$
  - $H(\cdot)$ is a good *hash* function when (***informally***)

$$\forall m \in \{0,1\}^*, h \in \{0,1\}^n, \Pr[H(m) = h] = \frac{1}{2^n}$$

  - it is “difficult” to find *collisions*

$$\text{find } m_1, m_2 \in \{0,1\}^* : m_1 \neq m_2, H(m_1) = H(m_2)$$

  - it is “difficult” to find a *preimage*

$$\text{given } m \in \{0,1\}^*,\ \text{find } m' : H(m') = m$$

Cryptographic Hash Functions

- *Cryptographic Hash:* $H : \{0,1\}^* \longrightarrow \{0,1\}^n$
  - $H(\cdot)$ is a good *hash* function when (***informally***)

$$\forall m \in \{0,1\}^*, h \in \{0,1\}^n, \Pr[H(m) = h] = \frac{1}{2^n}$$

  - it is “difficult” to find *collisions*

$$\text{find } m_1, m_2 \in \{0,1\}^* : m_1 \neq m_2, H(m_1) = H(m_2)$$

  - it is “difficult” to find a *preimage*

$$\text{given } m \in \{0,1\}^*,\ \text{find } m' : H(m') = m$$

  - e.g., SHA-1

Summary

- Basic ingredients: cryptographic primitives
  - secret-key (symmetric) cryptography (e.g., AES)
  - public-key (asymmetric) cryptography (e.g., RSA)
  - cryptographic hash functions (e.g., SHA-1)
  - stream ciphers (e.g., RC4)

Summary

- Basic ingredients: cryptographic primitives
  - secret-key (symmetric) cryptography (e.g., AES)
  - public-key (asymmetric) cryptography (e.g., RSA)
  - cryptographic hash functions (e.g., SHA-1)
  - stream ciphers (e.g., RC4)

- Recipes: cryptographic protocols
  - certificates (e.g., X.509)
  - secure transport (e.g., TLS, IPSec)
  - ...

- Basic ingredients: cryptographic primitives
  - secret-key (symmetric) cryptography (e.g., AES)
  - public-key (asymmetric) cryptography (e.g., RSA)
  - cryptographic hash functions (e.g., SHA-1)
  - stream ciphers (e.g., RC4)
- Recipes: cryptographic protocols
  - certificates (e.g., X.509)
  - secure transport (e.g., TLS, IPSec)
  - ...
- Applications
  - electronic commerce
  - secure shell
  - secure electronic mail
  - virtual private networks
  - ...