

# **A Few Basic Elements of Communication Security**

Antonio Carzaniga

Faculty of Informatics  
Università della Svizzera italiana

December 19, 2018



- ***Make backups*** of your data
- ***Do NOT trust the network!***
- Use ***HTTPS*** instead of HTTP (very common now)

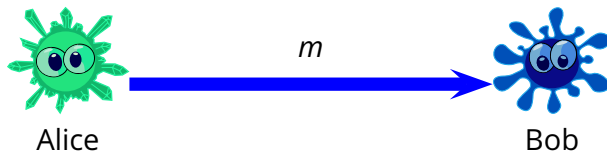
- ***Make backups*** of your data
  - ***Do NOT trust the network!***
  - Use ***HTTPS*** instead of HTTP (very common now)
- 
- Understand the basics of public-key cryptography
  - Communicate with ***end-to-end encryption*** (e.g., e-mail)
  - Use ***trusted certificates***
  - ***Encrypt your confidential data*** (and make backups)
  - Use ***strong passwords***
  - You might as well encrypt *all* your data
  - Tools/technologies: ***ssh***, ***pgp*** (or ***gpg***)

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



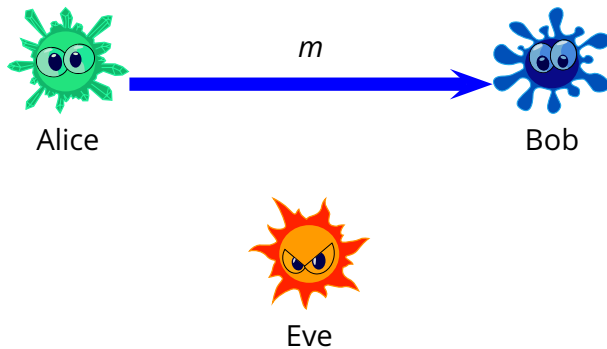
- *Communication model*: Alice sends a message  $m$  to Bob

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

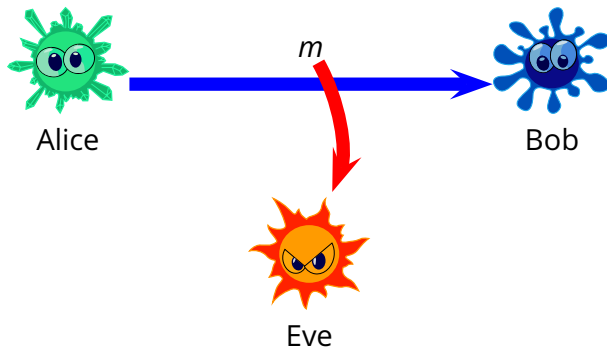




- *Communication model:* Alice sends a message  $m$  to Bob

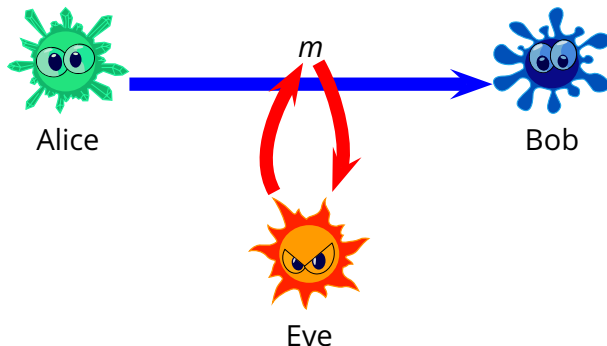


- *Communication model*: Alice sends a message  $m$  to Bob



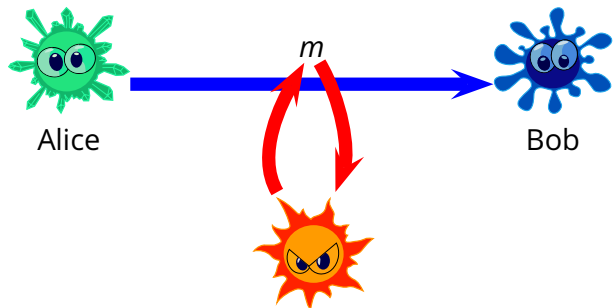
- *Passive adversary*
  - ▶ can read the message

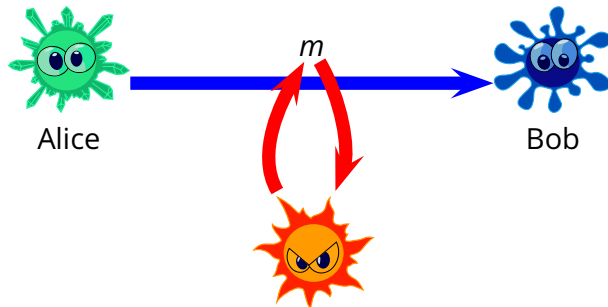
- *Communication model*: Alice sends a message  $m$  to Bob



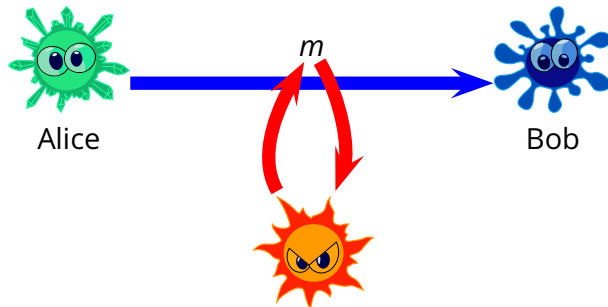
- *Passive adversary*
  - ▶ can read the message
- *Active adversary*
  - ▶ can modify the message







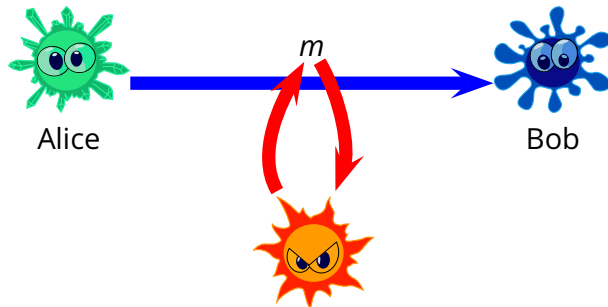
- **Confidentiality (a.k.a., privacy):** Alice wants to make sure that only Bob sees the message

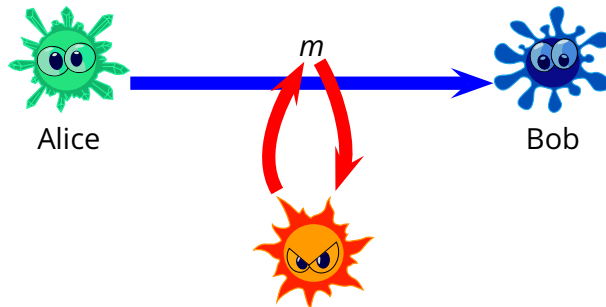


- **Confidentiality (a.k.a., privacy):** Alice wants to make sure that only Bob sees the message
- **Message Integrity:** Bob wants to make sure that the message he reads was exactly what Alice wrote

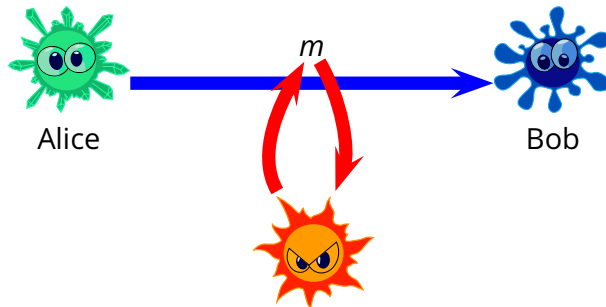








- **End-point Authentication:** Bob wants to make sure he is communicating with Alice



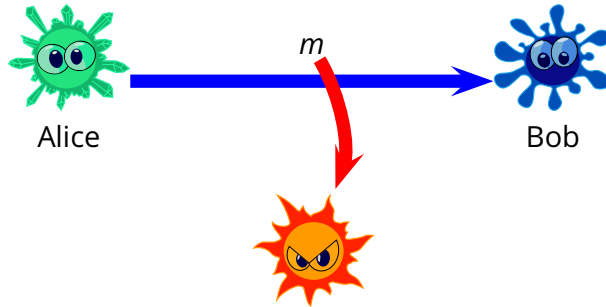
- **End-point Authentication:** Bob wants to make sure he is communicating with Alice
- **Operational/system security:** Alice and Bob want to maintain full control of their networks

***privacy***

***authentication***

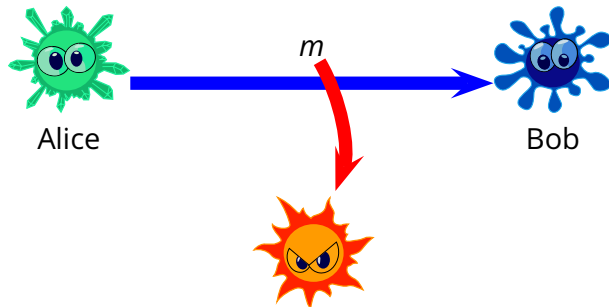
# What is Privacy, Exactly?

# 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



- The ciphertext is

BUUBDL BU EBXO

- The ciphertext is

BUUBDL BU EBXO

- Plaintext is

ATTACK AT DAWN

- 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  $\Sigma = \{'A', 'B', \dots, 'Z', ' '\}$

## ■ *Substitution cipher*

- ▶ alphabet  $\Sigma = \{'A', 'B', \dots, 'Z', ' '\}$
- ▶ encryption function: a ***permutation***

$$E : \Sigma \rightarrow \Sigma$$

## ■ *Substitution cipher*

- ▶ alphabet  $\Sigma = \{'A', 'B', \dots, 'Z', ' '\}$
- ▶ encryption function: a *permutation*

$$E : \Sigma \rightarrow \Sigma$$

### 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  $\Sigma = \{'A', 'B', \dots, 'Z', ' '\}$
- ▶ encryption function: a **permutation**

$$E : \Sigma \rightarrow \Sigma$$

### 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  $\Sigma = \{'A', 'B', \dots, 'Z', ' '\}$
- ▶ encryption function: a **permutation**

$$E : \Sigma \rightarrow \Sigma$$

### 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  $\Sigma = \{'A', 'B', \dots, 'Z', ' '\}$
- ▶ encryption function: a **permutation**

$$E : \Sigma \rightarrow \Sigma$$

### 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

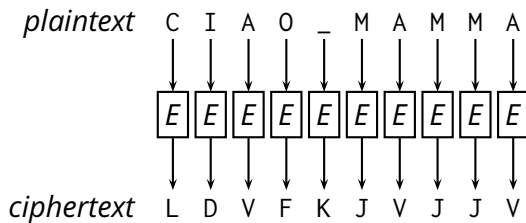
How many possible permutations?

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

- Encrypting some text using a substitution cipher

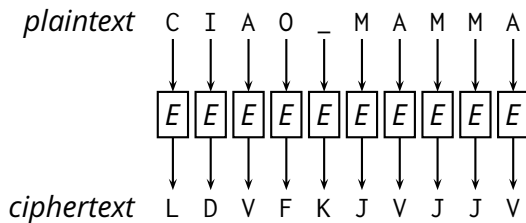
*plaintext*   C   I   A   O   \_   M   A   M   M   A

- 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  
yfyayfbwakvbabquogbuanwayfbcvaxvbkyovagcyfaxbvykcqapqkdcbkndbavctfyh  
yfyakioqtayfbhbakvbadclbadcnbywakquayfbampvhpcyaolafkmmcbhh  
yfyayoahbxpvbayfbhbavctfyhatorbvqibqyhakvbacqhycypybuakioqtaibq  
ubvcrcqtayfbcvajphyamogbvhalvoiyfbaxoqhbqyaolayfbatorbvqbu  
yfyagfbqbrbvakqwaloviaolatorbvqibqyanbxoibhaubhyvpxycrbaolayfbhbabquh  
cyachayfbavctfyalayfbambomdbayoakdybvaovayoaknodchfacy

# From Black Magic to Mathematics

History: secret algorithms, poorly understood security properties



# From Black Magic to Mathematics

History: secret algorithms, poorly understood 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%

### ■ 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  $\Rightarrow$  game***

# How to Provably Win the Game

***primitive  $\Rightarrow$  security protocol  $\Rightarrow$  game***

# How to Provably Win the Game

*hard math problem  $\Rightarrow$  primitive  $\Rightarrow$  security protocol  $\Rightarrow$  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  $i \leftarrow 2$  to  $\lfloor \sqrt{N} \rfloor$   
2      do if  $i$  divides  $N$   
3          then return  $i, N/i$ 
```

## 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  $i \leftarrow 2$  to  $\lfloor \sqrt{N} \rfloor$   
2      do if  $i$  divides  $N$   
3          then return  $i, N/i$ 
```

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

## 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  $i \leftarrow 2$  to  $\lfloor \sqrt{N} \rfloor$   
2      do if  $i$  divides  $N$   
3          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  $d \equiv e^{-1} \pmod{\phi(N)}$

# 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  $d \equiv e^{-1} \pmod{\phi(N)}$ 
  - ▶ 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  $d \equiv e^{-1} \pmod{\phi(N)}$ 
  - ▶ 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  $d \equiv e^{-1} \pmod{\phi(N)}$ 
  - ▶ your **public key** is  $(N, e)$
  - ▶ your **secret key** is  $(N, d)$
- **Encryption:**  $C \leftarrow \text{RSA}_{N,e}(M) := M^e \pmod{N}$



# 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  $d \equiv e^{-1} \pmod{\phi(N)}$ 
  - ▶ your **public key** is  $(N, e)$
  - ▶ your **secret key** is  $(N, d)$
- **Encryption:**  $C \leftarrow \text{RSA}_{N,e}(M) := M^e \pmod{N}$
- **Decryption:**  $M \leftarrow \text{RSA}_{N,d}(C) := C^d \pmod{N}$

**Factoring  $\Rightarrow$  RSA  $\Rightarrow$  SSH**

- Is RSA secure?

**Factoring  $\Rightarrow$  RSA  $\Rightarrow$  SSH**

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

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

- 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!

- 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?

- 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!*

- 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:

RSA is secure  $\Rightarrow$  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:

RSA is secure  $\Rightarrow$  SSH is secure

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

...and you are smarter than Gauss!

- 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)

- 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

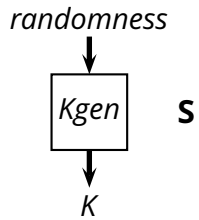
# Symmetric Encryption

*randomness*



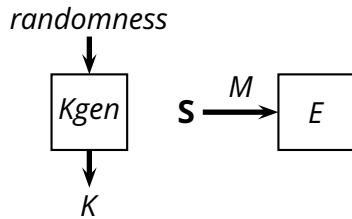
*K*

# Symmetric Encryption



**R**

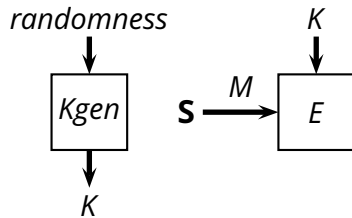
# Symmetric Encryption



**R**

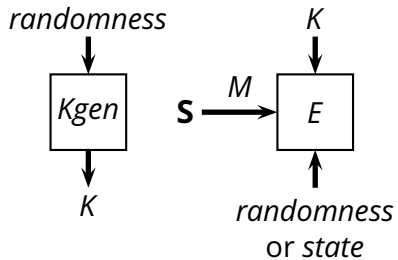


# Symmetric Encryption



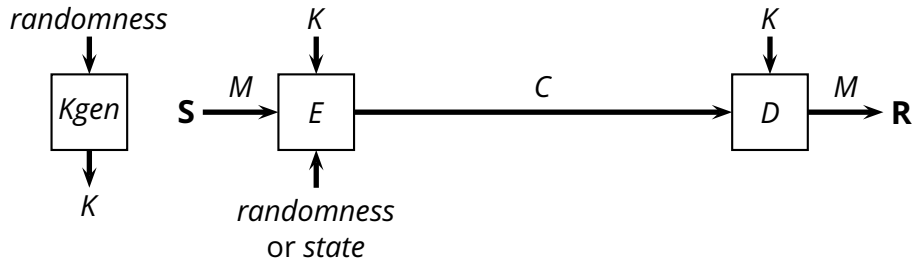
**R**

# Symmetric Encryption

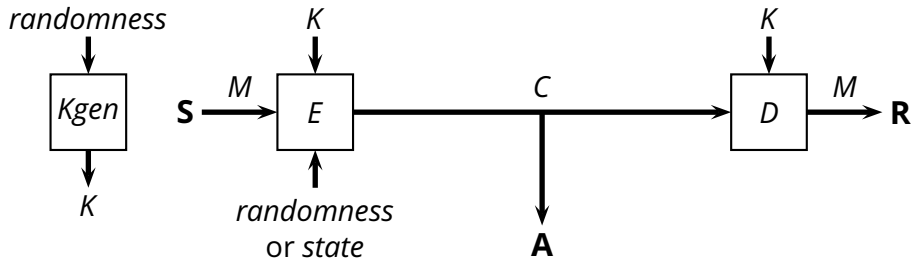


**R**

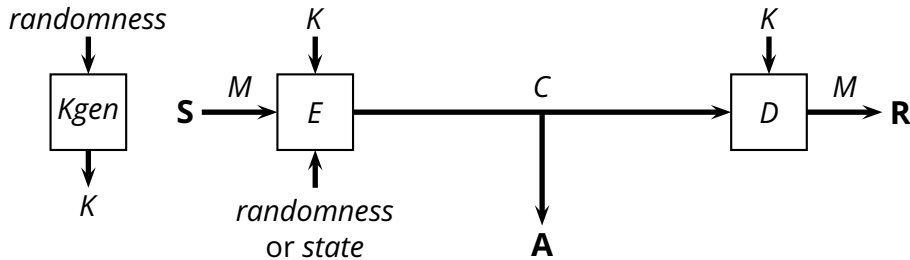
# Symmetric Encryption



# Symmetric Encryption



# Symmetric Encryption



---

<b>S</b>	sender
<b>R</b>	receiver
<b>A</b>	adversary

---

<i>E</i>	encryption algorithm
<i>D</i>	decryption algorithm

---

<i>M</i>	plaintext message
<i>C</i>	ciphertext message
<i>K</i>	key

---



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

$$M \in \{0, 1\}^n; \quad K \overset{\$}{\leftarrow} \{0, 1\}^n$$

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

$$M \in \{0, 1\}^n; \quad K \overset{\$}{\leftarrow} \{0, 1\}^n$$

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

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

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 and never reused

- ▶ decryption:

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

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

$$M \in \{0, 1\}^n; \quad K \overset{\$}{\leftarrow} \{0, 1\}^n$$

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 and never reused

- ▶ decryption:

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

- **Example:**      $M$    0110010110111011

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

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

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 and 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

$$M \in \{0, 1\}^n; \quad K \overset{\$}{\leftarrow} \{0, 1\}^n$$

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 and never reused

- ▶ decryption:

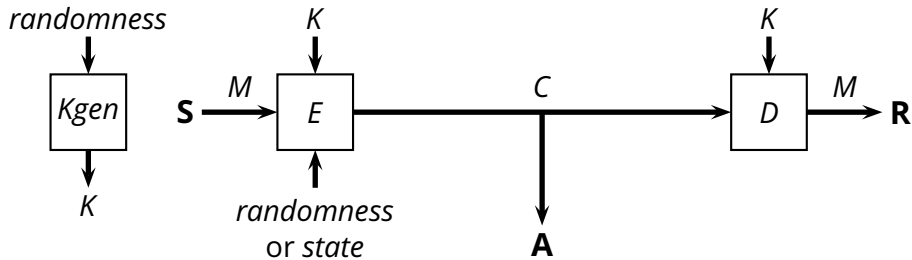
$$D(K, C) := C \oplus K$$

- **Example:**

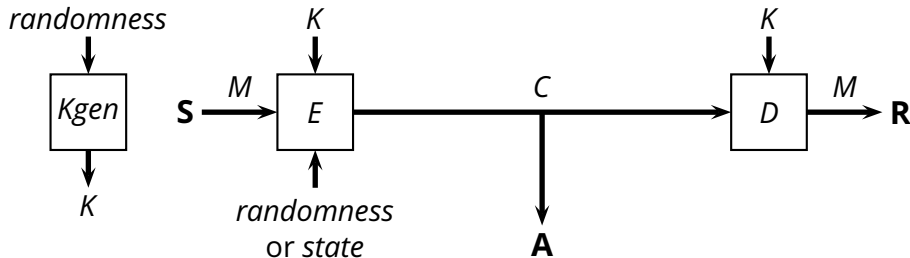
$M$	0110010110111011
$K$	1011000101000101
$C$	1101010011111110

# Symmetric Encryption (2)

## Symmetric Encryption (2)



## Symmetric Encryption (2)



**S** sender

**R** receiver

**A** adversary

*E* encryption algorithm

*D* decryption 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



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

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

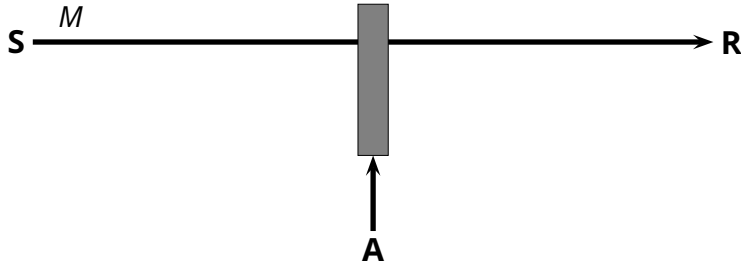
**S**

**R**

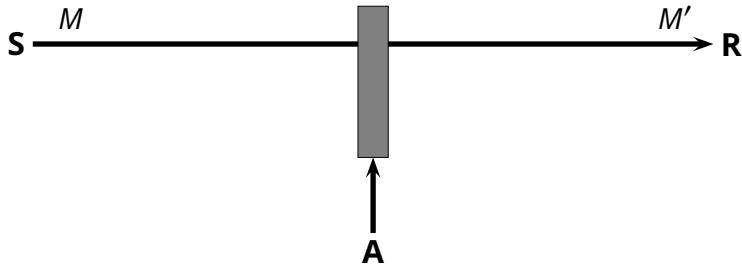
# Message Authenticity



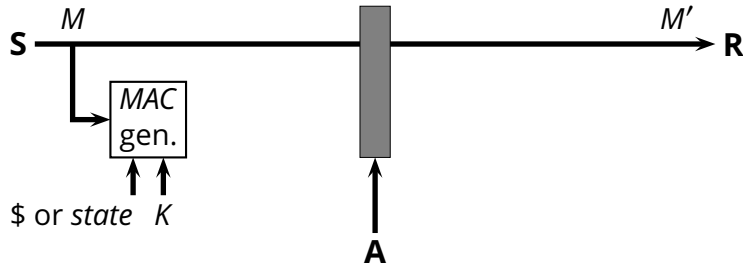
# Message Authenticity



# Message Authenticity

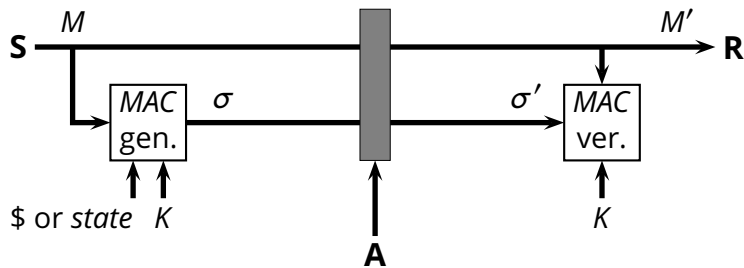


# Message Authenticity

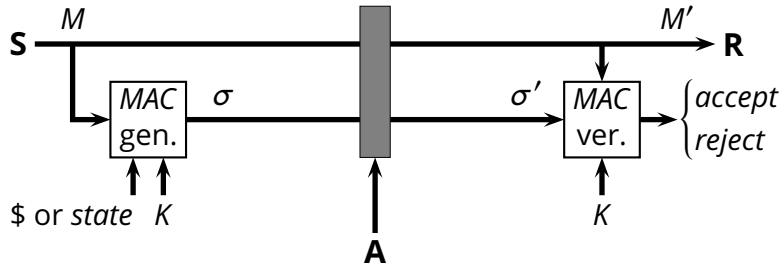




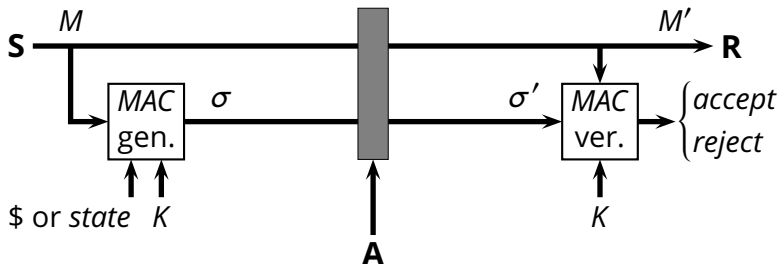
# Message Authenticity



# Message Authenticity



# Message Authenticity



$\sigma$	message authentication code (MAC)
$K$	key
$\$$	randomness
MAC gen.	MAC <i>generation</i> algorithm
MAC ver.	MAC <i>verification</i> algorithm

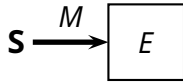
# Asymmetric Encryption

# Asymmetric Encryption

**S**

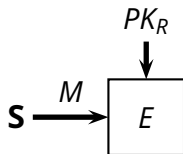
**R**

# Asymmetric Encryption

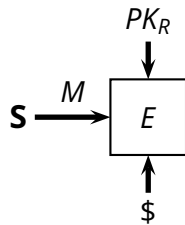


**R**

# Asymmetric Encryption



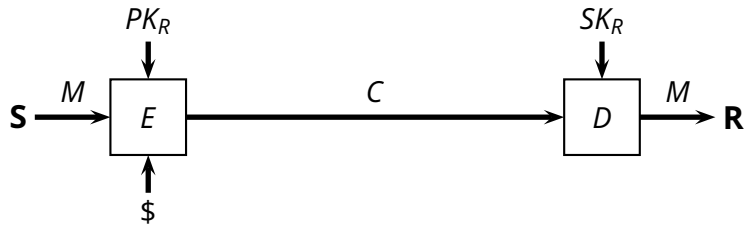
# Asymmetric Encryption



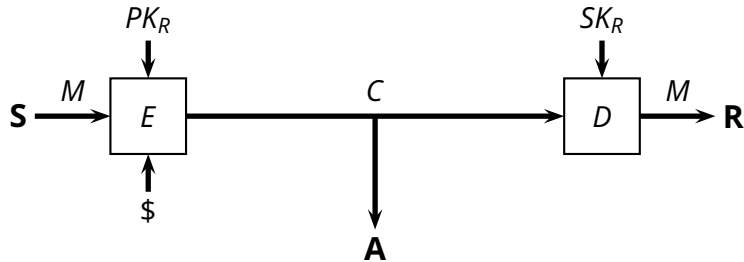
**R**



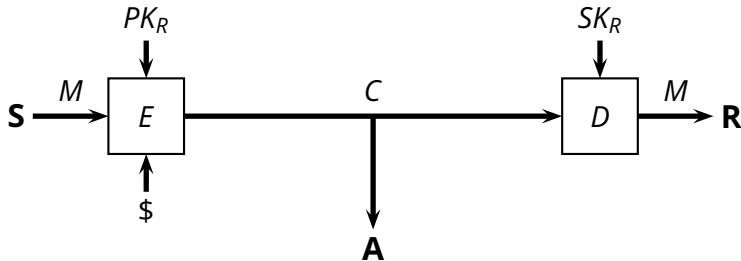
# Asymmetric Encryption



# Asymmetric Encryption

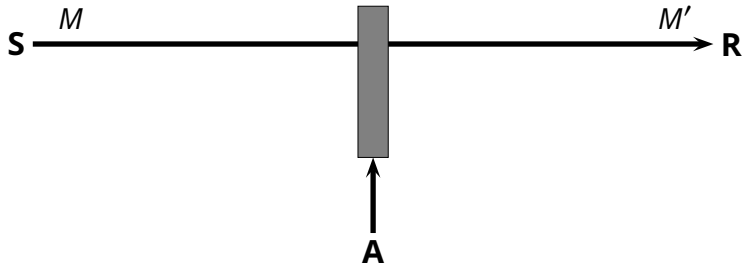


# Asymmetric Encryption

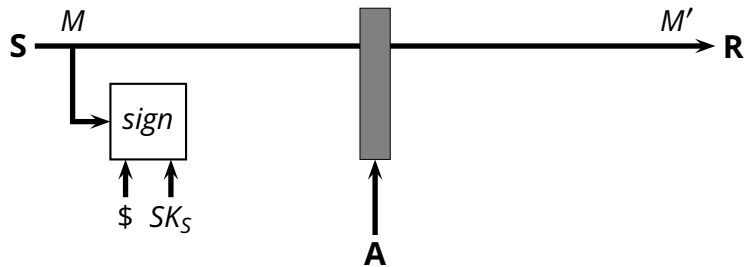


$PK_R$	receiver's <i>public</i> key
$SK_R$	receiver's <i>secret</i> key
$M$	plaintext message
$C$	ciphertext message

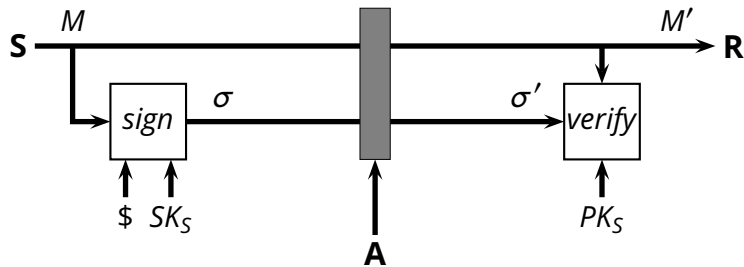




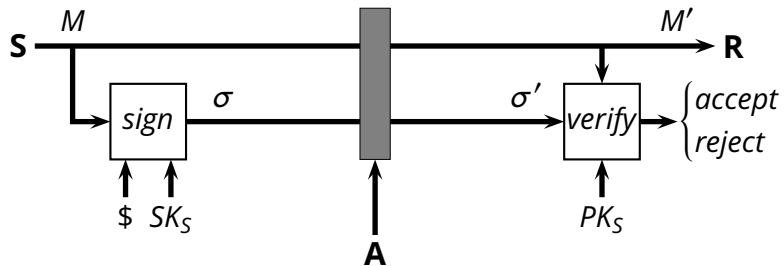
# Digital Signatures



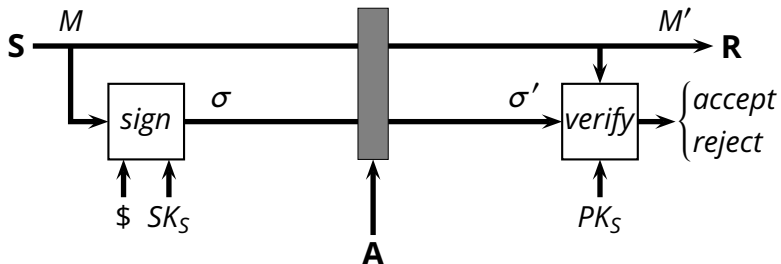
# Digital Signatures



# Digital Signatures







$\sigma$	digital signature
$SK_S$	sender's <i>secret</i> key
$PK_S$	sender's <i>public</i> key
$\$$	randomness
<i>sign</i>	<i>signing</i> algorithm
<i>verify</i>	<i>verification</i> algorithm

# Primitives vs. Protocols

## ■ *Protocol*

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

## ■ *Protocol*

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

## ■ *Primitive*

## ■ *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”



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

- 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  $C \leftarrow M \oplus S_K[s \dots s + |M| - 1]$
  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*

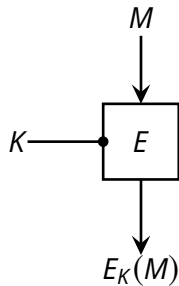
1.  $S$  computes  $C \leftarrow M \oplus S_K[s \dots s + |M| - 1]$
2.  $S$  updates its position  $s \leftarrow s + |M|$

## ■ *Decryption protocol*

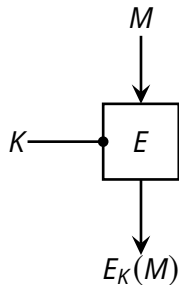
1.  $R$  computes  $M \leftarrow C \oplus S_K[s \dots s + |C| - 1]$
2.  $R$  updates its position  $s \leftarrow s + |C|$

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

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



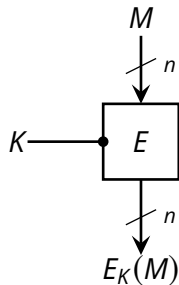
■ *Block Cipher*:  $E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$



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

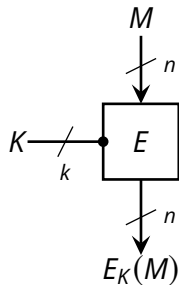


■ *Block Cipher*:  $E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$



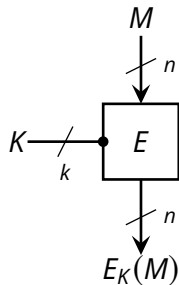
- ▶  $E_K(\cdot)$  is a *permutation*, so  $E_K^{-1}(\cdot)$  is always defined
- ▶ fixed-length input and output ( $n$ )

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



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

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



- ▶  $E_K(\cdot)$  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  $E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$
- ▶ split  $M$  into  $n$ -bit blocks  $M = M_0 || M_1 || \dots || M_\ell$  ( $\ell = \lfloor N/n \rfloor$ )

## ■ Symmetric encryption

- ▶ Input:  $k$ -bit key  $K$ ,  $N$ -bit message  $M$
- ▶ Output:  $N$ -bit ciphertext  $C$

## ■ Cipher Block Chaining (CBC)

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

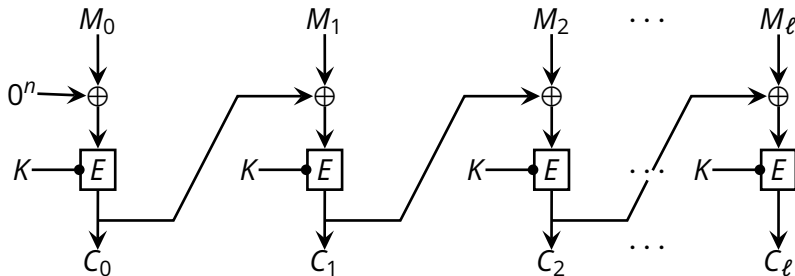
```
CBC( $K, M$ )  
1   $x \leftarrow 0^n$   
2  for  $i \leftarrow 0$  to  $\lfloor |M|/n \rfloor$   
3      do  $C[ni \dots ni + n - 1] \leftarrow E_K(x \oplus M[ni \dots ni + n - 1])$   
4           $x \leftarrow C[ni \dots ni + n - 1]$   
5  return  $C$ 
```

## ■ Symmetric encryption

- ▶ Input:  $k$ -bit key  $K$ ,  $N$ -bit message  $M$
- ▶ Output:  $N$ -bit ciphertext  $C$

## ■ Cipher Block Chaining (CBC)

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





- Write the decryption algorithm for CBC

- Write the decryption algorithm for CBC

CBC-DECRYPT( $K, C$ )

```
1   $x \leftarrow 0^n$ 
2  for  $i \leftarrow 0$  to  $\lfloor |C|/n \rfloor$ 
3      do  $M[ni \dots ni + n - 1] \leftarrow x \oplus E_K^{-1}(C[ni \dots ni + n - 1])$ 
4           $x \leftarrow C[ni \dots ni + n - 1]$ 
5  return  $M$ 
```

## An Encryption Protocol (2)

## An Encryption Protocol (2)

- Is this CBC protocol secure?

■ Is this CBC protocol secure?

- ▶ ***any deterministic stateless protocol is insecure***
- ▶ we need *state* and/or *randomness*

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

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

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



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

```
CBC$-ENCRYPT( $K, M$ )
1  if  $|M| = 0 \vee |M| \neq 0 \pmod n$ 
2    then return  $\perp$ 
3   $M[1] \cdot M[2] \cdots M[\ell] \leftarrow M$ 
4   $IV \xleftarrow{\$} \{0, 1\}^n$ 
5   $C[0] \leftarrow IV$ 
6  for  $i \leftarrow 1$  to  $\ell$ 
7    do  $C[i] \leftarrow E_K(C[i-1] \oplus M[i])$ 
8   $C \leftarrow C[1] \cdot C[2] \cdots C[\ell]$ 
9  return  $\langle IV, C \rangle$ 
```

## CBC With Random IV (2)

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

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

```
CBC$-DECRYPT( $K, IV, C$ )  
1  if  $|C| = 0 \vee |C| \neq 0 \pmod n$   
2    then return  $\perp$   
3   $C[1] \cdot C[2] \cdots C[\ell] \leftarrow C$   
4   $C[0] \leftarrow IV$   
5  for  $i \leftarrow 1$  to  $\ell$   
6    do  $M[i] \leftarrow C[i-1] \oplus E_K(C[i])$   
7   $M \leftarrow M[1] \cdot M[2] \cdots M[\ell]$   
8  return  $M$ 
```

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

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

CBCC-ENCRYPT( $K, M$ )

```
1  static  $ctr \leftarrow 0$ 
2  if  $ctr \geq 2^n \vee |M| = 0 \vee |M| \neq 0 \pmod n$ 
3    then return  $\perp$ 
4   $M[1] \cdot M[2] \cdots M[\ell] \leftarrow M$ 
5   $IV \leftarrow [ctr]_n$ 
6   $C[0] \leftarrow [ctr]_n$ 
7  for  $i \leftarrow 1$  to  $\ell$ 
8    do  $C[i] \leftarrow E_K(C[i-1] \oplus M[i])$ 
9   $C \leftarrow C[1] \cdot C[2] \cdots C[\ell]$ 
10  $ctr \leftarrow ctr + 1$ 
11 return  $\langle IV, C \rangle$ 
```

## CBC With Stateful Counter (2)

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

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

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

```
1  if  $IV + |C| \geq 2^n \vee |C| = 0 \vee |C| \neq 0 \pmod n$   
2    then return  $\perp$   
3   $C[1] \cdot C[2] \cdots C[\ell] \leftarrow C$   
4   $IV \leftarrow [ctr]_n$   
5   $C[0] \leftarrow IV$   
6  for  $i \leftarrow 1$  to  $\ell$   
7    do  $M[i] \leftarrow C[i-1] \oplus E_K^{-1}(C[i])$   
8   $M \leftarrow M[1] \cdot M[2] \cdots M[\ell]$   
9  return  $M$ 
```

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



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

■ **CTR\$**: counter mode with random initial counter

- ▶ family of functions:  $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$

CTR\$-ENCRYPT( $K, M$ )

```
1   $R \xleftarrow{\$} \{0, 1\}^n$ 
2   $Pad \leftarrow F_K([R]_n)$ 
3  for  $i \leftarrow 1$  to  $\lceil |M|/n \rceil - 1$ 
4      do  $Pad \leftarrow Pad \cdot F_K([R + i]_n)$ 
5   $Pad \leftarrow$  first  $|M|$  bits of  $Pad$ 
6   $C \leftarrow M \oplus Pad$ 
7  return  $\langle R, C \rangle$ 
```

■ **CTR\$:** counter mode with random initial counter (decryption)

- ▶ *family of functions:*  $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$

■ **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( $K, R, C$ )

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

- **CTRC:** counter mode with stateful counter

- ▶ *family of functions:*  $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$

■ **CTRC:** counter mode with stateful counter

- ▶ family of functions:  $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$

CTRC( $K, M$ )

```

1  static  $R \leftarrow 0$ 
2   $\ell \leftarrow \lceil |M|/n \rceil$ 
3  if  $R + \ell - 1 \geq 2^n$ 
4    then return  $\perp$ 
5   $Pad \leftarrow F_K([R]_n)$ 
6  for  $i \leftarrow 1$  to  $\ell - 1$ 
7    do  $Pad \leftarrow Pad \cdot F_K([R + i]_n)$ 
8   $Pad \leftarrow$  first  $|M|$  bits of  $Pad$ 
9   $C \leftarrow M \oplus Pad$ 
10  $R \leftarrow R + \ell$ 
11 return  $\langle R - \ell, C \rangle$ 
```

- **CTRC:** counter mode with stateful counter (decryption)

- ▶ *family of functions:*  $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$

■ **CTRC**: counter mode with stateful counter (decryption)

- ▶ *family of functions*:  $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$

CTRC-DECRYPT( $K, R, C$ )

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



## ■ *MAC generation*

- ▶ *Input:  $k$ -bit key  $K$ ,  $N$ -bit message  $M$*
- ▶ *Output:  $n$ -bit message authentication code  $\sigma$*

## ■ MAC generation

- ▶ *Input:*  $k$ -bit key  $K$ ,  $N$ -bit message  $M$
- ▶ *Output:*  $n$ -bit message authentication code  $\sigma$

## ■ 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  $M = M_0 || M_1 || \dots || M_\ell$  ( $\ell = \lfloor N/n \rfloor$ )

MAC( $K, M$ )

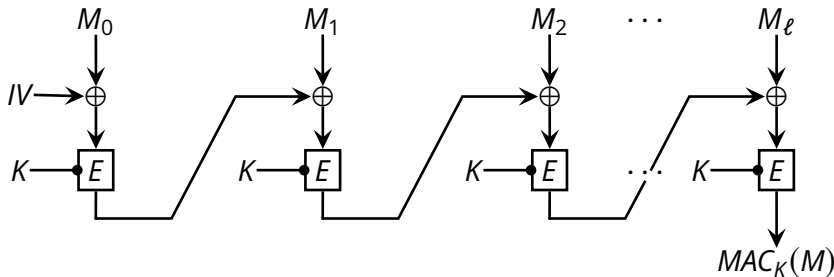
```
1   $IV \xleftarrow{\$} \{0, 1\}^n$   
2   $C \leftarrow IV$   
3  for  $i \leftarrow 0$  to  $\lfloor |M|/n \rfloor$   
4      do  $C \leftarrow E_K(C \oplus M[ni \dots ni + n - 1])$   
5  return  $\langle IV, C \rangle$ 
```

## ■ MAC generation

- ▶ Input:  $k$ -bit key  $K$ ,  $N$ -bit message  $M$
- ▶ Output:  $n$ -bit message authentication code  $\sigma$

## ■ 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  $M = M_0 || M_1 || \dots || M_\ell$  ( $\ell = \lfloor N/n \rfloor$ )



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

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

```
CBC-MAC$(K, M)
1  if  $|M| = 0 \vee |M| \neq 0 \pmod n$ 
2    then return  $\perp$ 
3   $M[1] \cdot M[2] \cdots M[\ell] \leftarrow M$ 
4   $IV \xleftarrow{\$} \{0, 1\}^n$ 
5   $C \leftarrow IV$ 
6  for  $i \leftarrow 1$  to  $\ell$ 
7    do  $C \leftarrow E_K(C \oplus M[i])$ 
8  return  $\langle IV, C \rangle$ 
```

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

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

```

CBC-MAC$-VERIFY( $K, IV, \sigma, M$ )
1  if  $|M| = 0 \vee |M| \neq 0 \pmod n$ 
2    then return  $\perp$ 
3   $M[1] \cdot M[2] \cdots M[\ell] \leftarrow M$ 
4   $C \leftarrow IV$ 
5  for  $i \leftarrow 1$  to  $\ell$ 
6    do  $C \leftarrow E_K(C \oplus M[i])$ 
7  if  $C = \sigma$ 
8    then return ACCEPT
9    else return REJECT
  
```

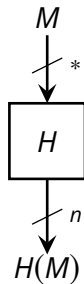
# Cryptographic Hash Functions

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



# Cryptographic Hash Functions

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



# Cryptographic Hash Functions

■ *Cryptographic Hash*:  $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$

▶  $H(\cdot)$  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 Functions

■ *Cryptographic Hash*:  $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$

- ▶  $H(\cdot)$  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 Functions

■ *Cryptographic Hash*:  $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$

- ▶  $H(\cdot)$  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*

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

# Cryptographic Hash Functions

■ *Cryptographic Hash*:  $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$

- ▶  $H(\cdot)$  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

- 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)

- 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
- ▶ ...