Cryptography 1

INGI2347: COMPUTER SYSTEM SECURITY (Spring 2014)

Marco Canini

Université catholique de Louvain

Lecture slides adapted from UCL INGI2347 by Gildas Avoine, Upenn CSE331 by Steve Zdancewic, and Dan Boneh, Stanford Cryptography I@Coursera. Reproduced with permission

Announcements

- First challenge deadline is today (24th Feb), 23:59h
- Schedule update:
- Swapped Crypto2 and Firewalls lab session
- Please use Piazza for all communications
  - If Piazza fails, then use email
- Remember to vote on Piazza (by end of Feb)

Plan for today

Lecture 6

- What's crypto?

- Symmetric key encryption
- Trusted 3rd parties
- Public key encryption
- Message Integrity
  - Crypto hash functions | MAC

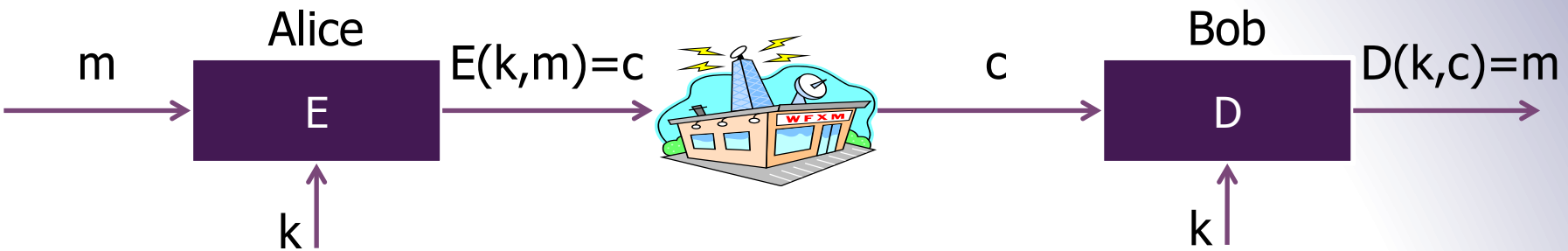
κρμπτο γραφη (Cryptography)

- Greek for "secret writing"
- Confidentiality
  - Obscure a message from eaves-droppers
- Integrity
  - Assure recipient that the message was not altered
- Authentication
  - Verify the identity of the source of a message
- Non-repudation
  - Convince a 3rd party that what was said is accurate

Terminology

- Encryption algorithm
  - Transforms a plaintext into a ciphertext that is unintelligible for non-authorized parties
  - Usually parametrized with a cryptographic key
- Asymmetric (Public) key cryptography
  - Crypto system: encryption + decryption algorithms + key generation
- Symmetric (Shared) key cryptography
  - Cipher/decipher: symmetric-key encryption/decryption algorithms

Build block: sym.-key encryption



E, D: cipher k: secret key (e.g., 128 bits)

m, c: plaintext, cipher text

- Same secret key for both encryption and decryption
- Encryption algorithm is publicly known
  - Never use a proprietary algorithm
- Fast in software or hardware implementations

Use Cases

- **Single use key**: (one time key)
  - Key is only used to encrypt one message
    - encrypted email: new key generated for every email
- **Multi use key**: (many time key)
  - Key used to encrypt multiple messages
    - encrypted files: same key used to encrypt many files
  - Need more machinery than for one-time key

Things To Remember

- Cryptography is:
  - A tremendous tool
  - The basis for many security mechanisms
- Cryptography is **NOT**:
  - The solution to all security problems
  - Reliable unless implemented and used properly
  - Something you should try to invent yourself
    - many many examples of broken ad-hoc designs
  - Privacy | Steganography | (Encoding|Decoding)

Cryptography is NOT

- Privacy
  - Ability to control how personal information spreads in a community
- Steganography
  - Science of information hiding
- Encoding | Decoding
  - Code is a system of symbols which represent information
  - Encoding transforms information into a codeword
  - Decoding recovers information from a codeword

Cryptanalysis: a rigorous science

Proves or disproves the security of a cryptosystem

- The three steps:
  - Precisely specify threat model
  - Propose a construction
  - Prove that breaking construction under threat mode will solve an underlying hard problem
- Definition of break:
  - Decrypting (totally or partially) a given ciphertext
  - Recovering the key of the cryptosystem
  - Proving a cryptosystem is less secure than what is claimed

+

Symmetric-key cryptography

Symmetric Ciphers: definition

Def: a **cipher** defined over (K, M, C)

is a pair of "efficient" algs (E, D) where

$$E: K \times M \to C, \quad D: K \times C \to M$$

s.t. $\forall m \in M, k \in K: \quad D(k, E(k, m)) = m$

E is often randomized | D is always deterministic

Stream vs. Block Ciphers

- Stream ciphers
  - Act on the plaintext one symbol at a time
  - Examples: RC4, GSM A5-1, Bluetooth E0, CSS, …
  - High speed rate, hardware implementations very cheap
  - Security analysis is not well established
- Block ciphers
  - Act on the plaintext in blocks of symbols
  - Examples: DES, 3DES, AES, IDEA, Blowfish, RC5, Kasumi, Safer, ...
  - Suited to software implementations on various systems
    - (e.g., 8-bit, 32-bit, 64-bit processors)
  - Security analysis is well established

Stream Ciphers: The One Time Pad (Vernam 1917)

First example of a "secure" cipher

$$M = C = \{0,1\}^n, \quad K = \{0,1\}^n$$

key = (random bit string as long the message)

$$E(k,m) = k \oplus m$$

$$D(k,c) = k \oplus c$$

msg:	0	1	1	0	1	1	1	
key:	1	0	1	1	0	1	0	$\oplus$
CT:	1	1	0	1	1	0	1	

Indeed:

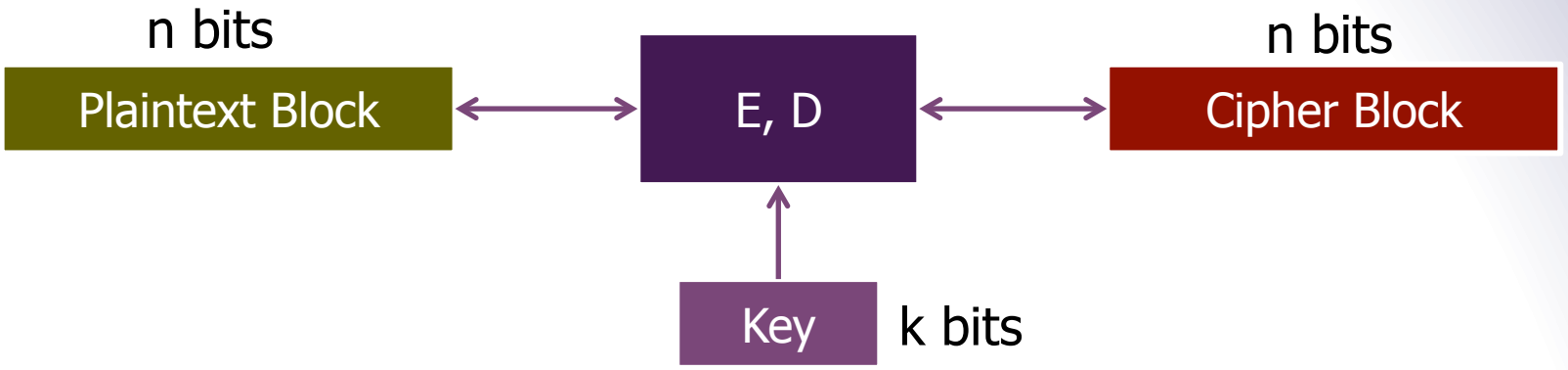
$$D(k,E(k,m)) = D(k,k \oplus m) = k \oplus (k \oplus m) = (k \oplus k) \oplus m = 0 \oplus m = m$$

The One Time Pad (Vernam 1917)

Very fast enc/dec !!

… but long keys (as long as plaintext)

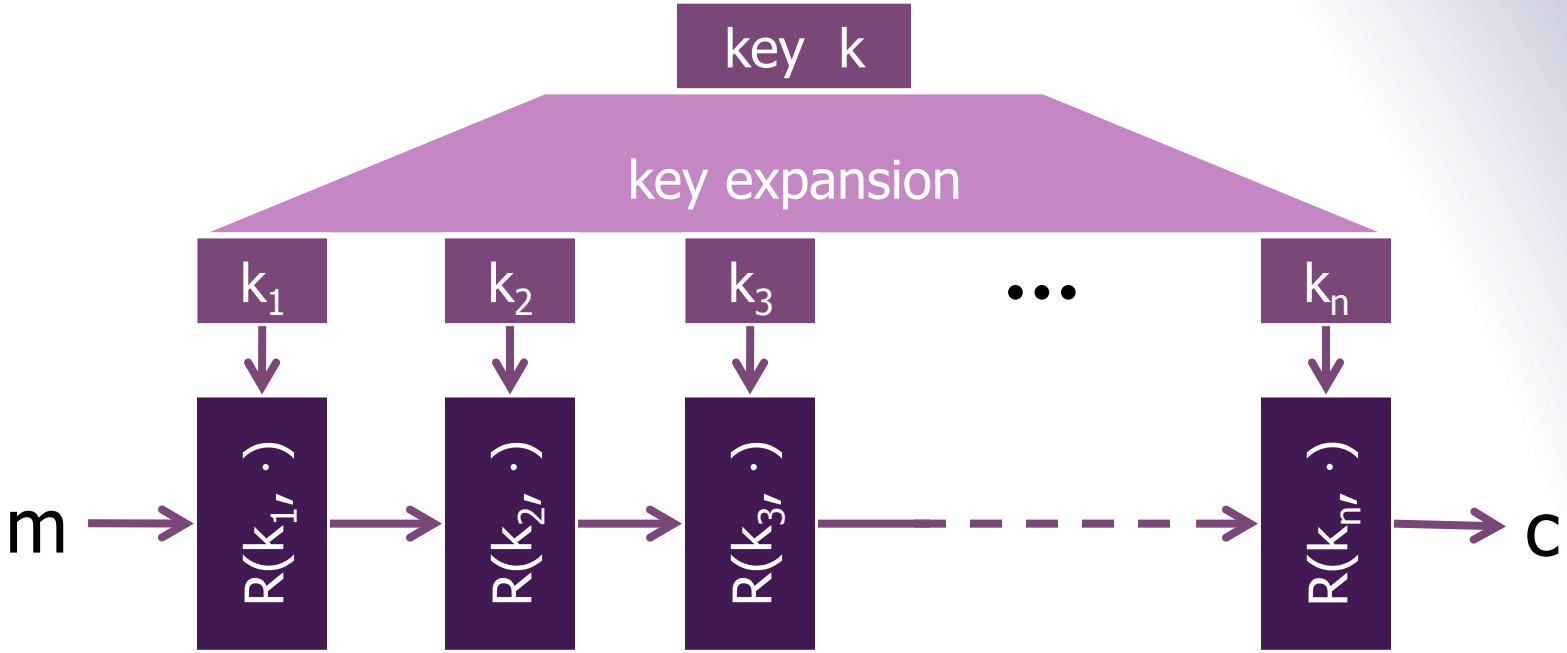
Block Ciphers



- Canonical examples:

1. 3DES: n= 64 bits, k = 168 bits
2. AES: n=128 bits, k = 128, 192, 256 bits

Block Ciphers Build by Iteration



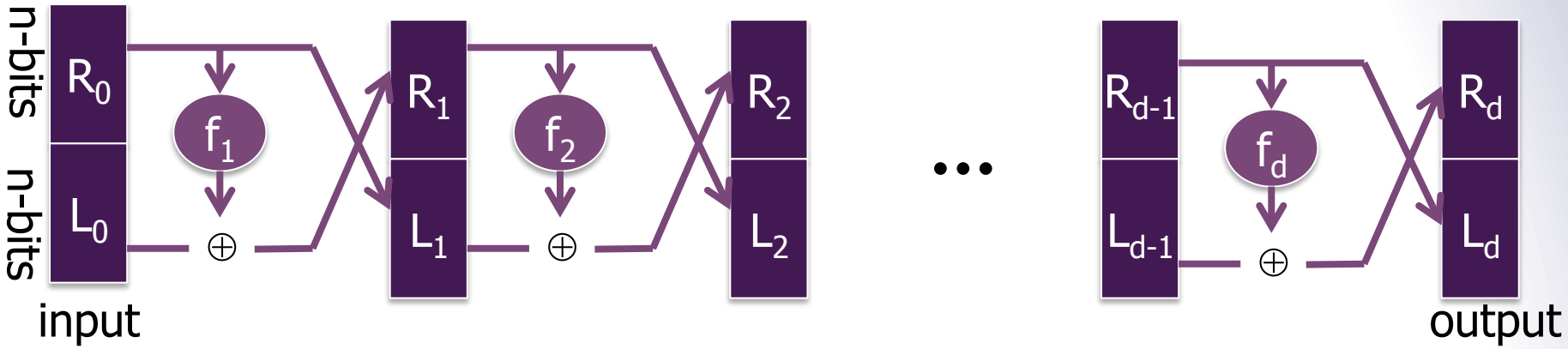
R(k,m) is called a round function

for 3DES (n=48), for AES-128 (n=10)

Block Ciphers: DES
(Feistel Network)

Given functions $f_1, \ldots, f_d$: $\{0,1\}^n \longrightarrow \{0,1\}^n$

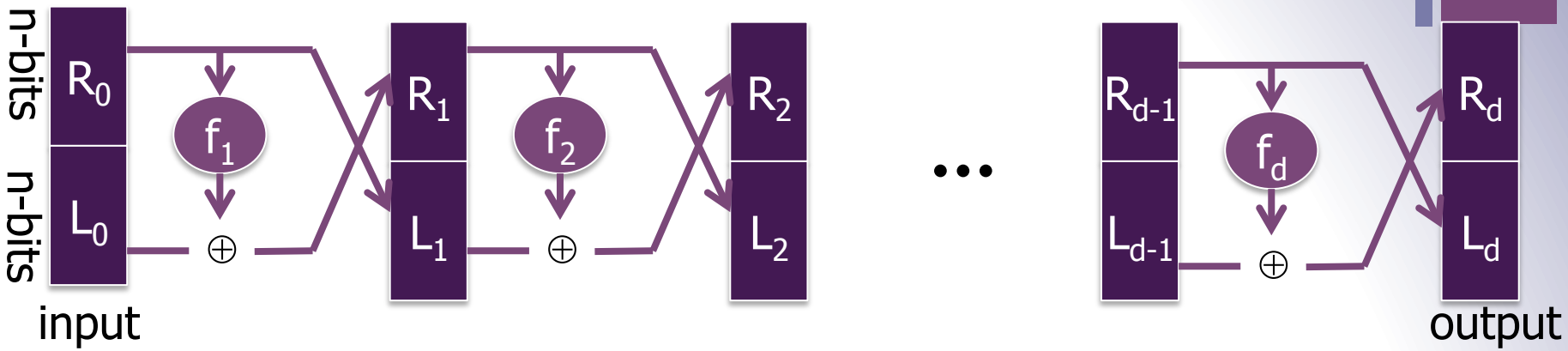
Goal: build invertible function $F: \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$



In symbols:

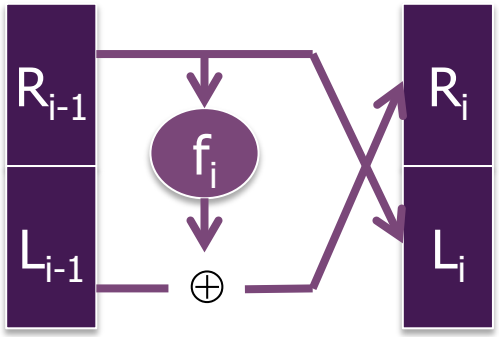
$$\begin{cases} R_i = f_i(R_{i-1}) \oplus L_{i-1} \\ L_i = R_{i-1} \end{cases}$$

Feistel Network



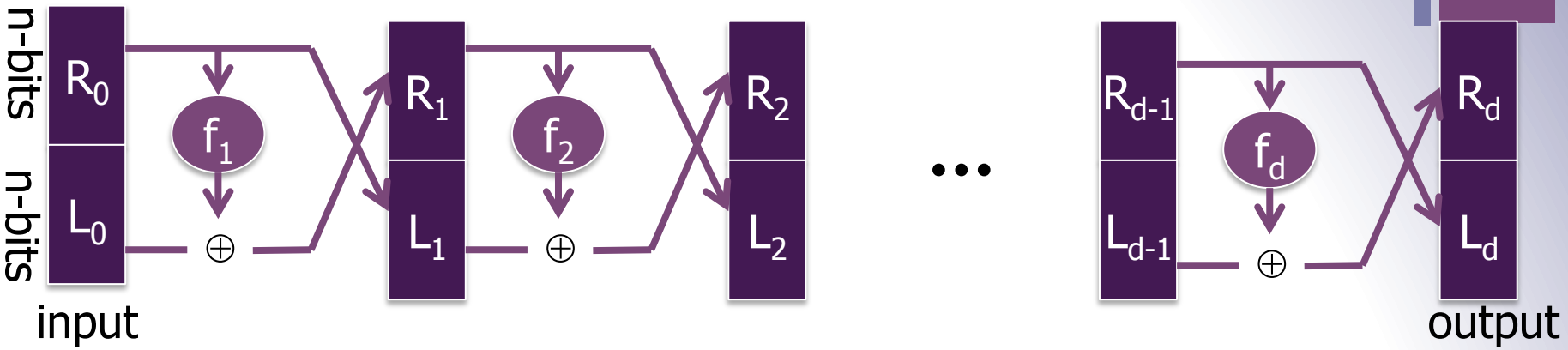
Feistel network $F: \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ is invertible

Construct inverse:



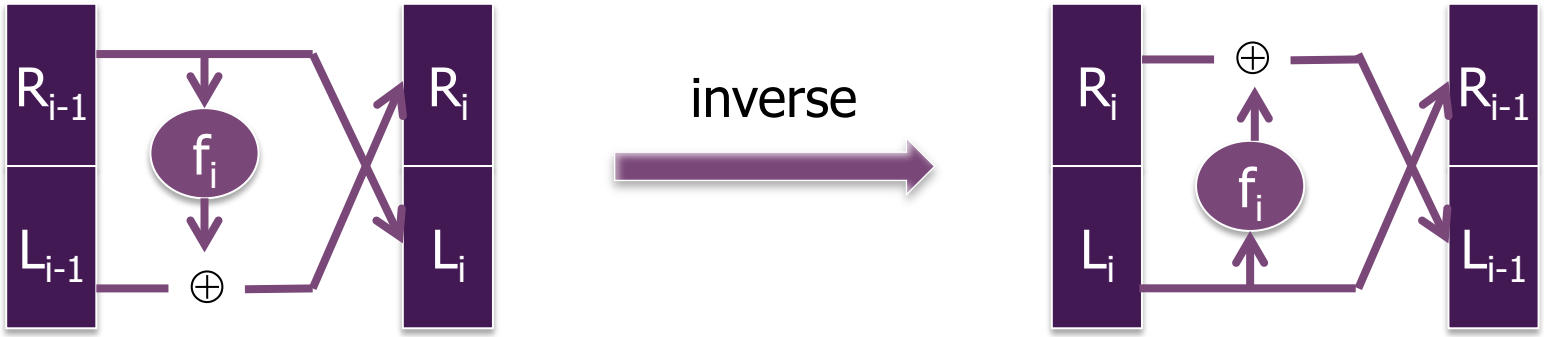
$$\begin{cases} R_{i-1} = L_i \\ L_{i-1} = \end{cases}$$

Feistel Network Decryption



Feistel network $F: \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ is invertible

Construct inverse:

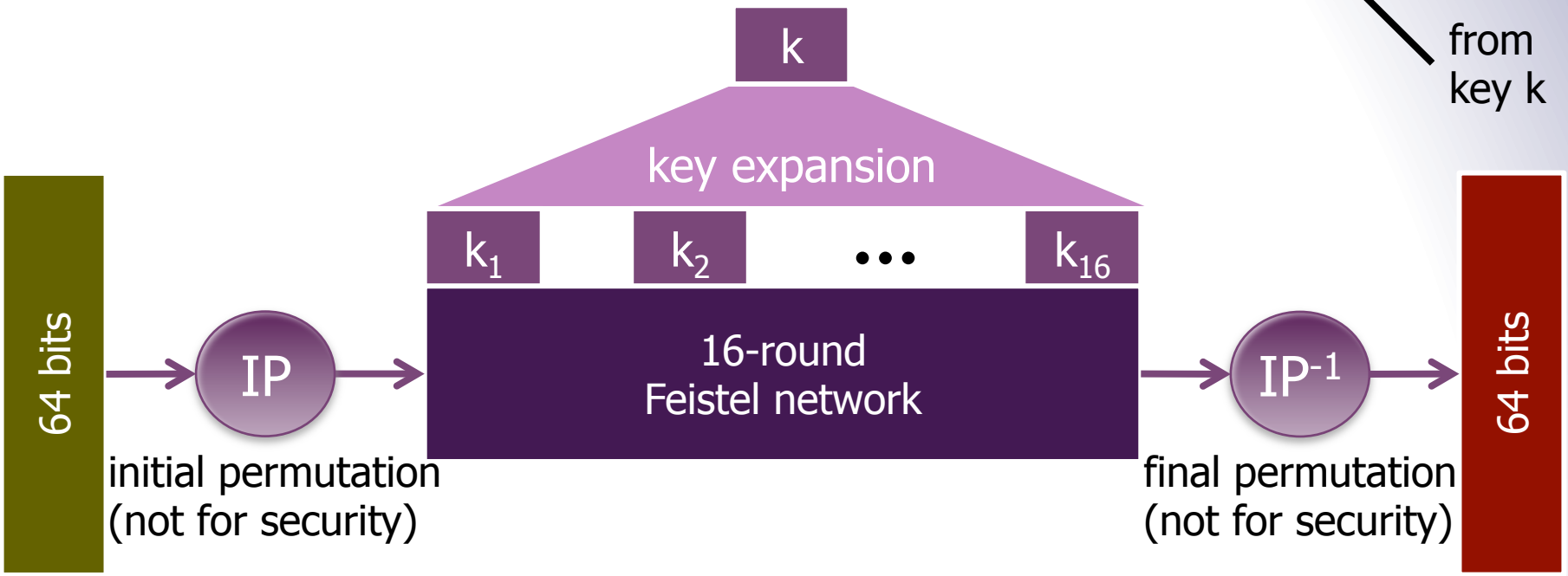


Feistel Network Decryption

- Inversion is basically the same circuit, with $f_1, \ldots, f_d$ applied in reverse order
- General method for building invertible functions (block ciphers) from arbitrary functions
- Used in many block ciphers … but not AES

DES: 16-round Feistel Network

$f_1, \ldots, f_{16}: \{0,1\}^{32} \longrightarrow \{0,1\}^{32}$, $f_i(x) = \mathbf{F}(k_i, x)$



input To invert, use keys in reverse order output

Advanced Encryption Standard (AES)

- National Institute of Standards & Technology NIST
  - Computer Security Research Center (CSRC)
  - http://csrc.nist.gov/
- Uses the Rijndael algorithm
  - Invented by Belgium researchers
    Dr. Joan Daemen & Dr. Vincent Rijmen
  - http://jda.noekeon.org/JDA_VRI_Rijndael_V2_1999.pdf
  - Adopted May 26, 2002
  - Key length: 128, 192, or 256 bits
  - Block size: 128, 192, or 256 bits

Performance

Crypto++ 5.6.0 [Wei Dai]

AMD Opteron, 2.2 GHz (Linux)

	Cipher	Block/key size	Speed (MB/sec)
stream	RC4		126
	Salsa20/12		643
	Sosemanuk		727
block	3DES	64/168	13
	AES-128	128/128	109

What is a secure cipher?

Attacker's abilities: **obtains one ciphertext** (for now)

Possible security requirements:

attempt #1: **attacker cannot recover secret key**

$E(k,m) = m$ Insecure: **leaks m**

attempt #2: **attacker cannot recover all of plaintext**

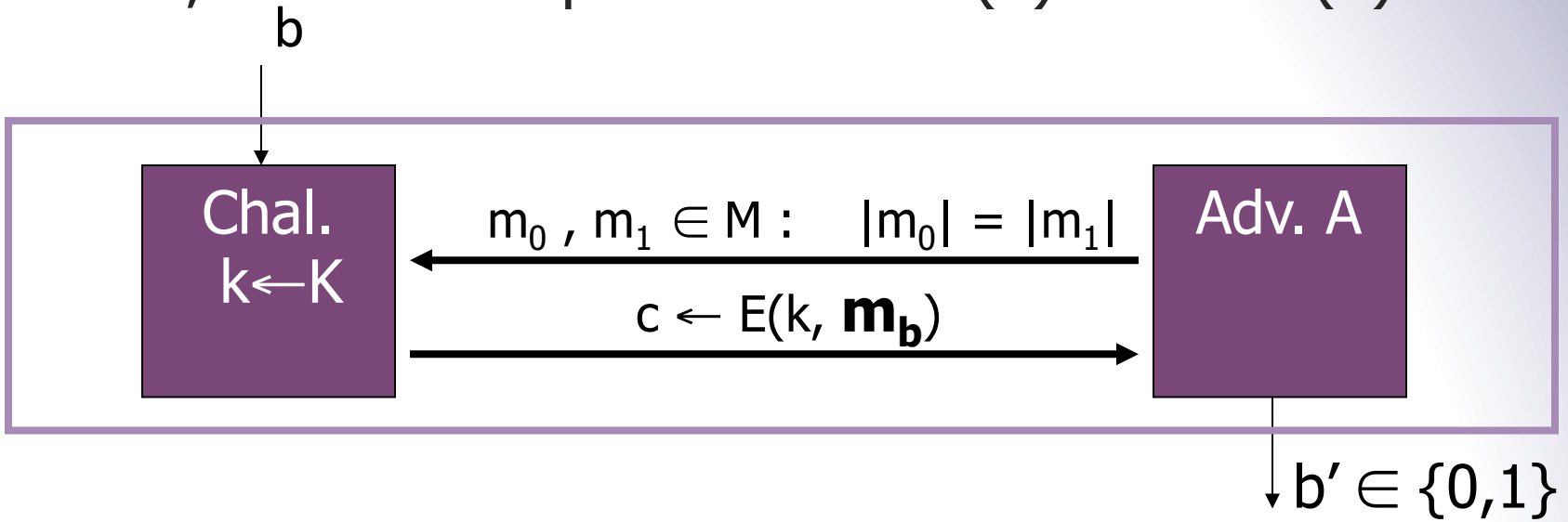
$E(k,m_0 \,||\, m_1) = m_0 \,||\, E(k,m_1)$ Insecure: **leaks m_0**

Shannon's information-theoretic perfect secrecy:

CT should reveal no "info" about PT

Semantic Security (one-time key)

For b=0,1 define experiments EXP(0) and EXP(1) as:



for b=0,1: W_b := [event that EXP(b)=1]

$$Adv_{SS}[A,E] := \left| Pr[W_0] - Pr[W_1] \right| \in [0,1]$$

Sematic Security Advantage of A against E

Semantic Security (one-time key)

Def: E is **semantically secure** if for all efficient A

$Adv_{SS}[A,E]$ is negligible

Security

- DES
  - 1998: DES-Cracker, dedicated hardware, 250 000 USD, exhaustive search in less than 2 days
  - 2006/2008: COPACOBANA/RIVYERA, 128 FPGAs (Spartan-3 5000), less than 1 day
- 3DES = chain 3 DES with a key bundle k_1 k_2 k_3:
  - Typical method: $E(k_3, D(k_2, E(k_1, m)))$
  - Backward compatible with DES
- AES
  - If DES could be broken in 1 second, then AES would require 149 trillion years to be broken

Problems with Shared Key Crypto

- Compromised key means interceptors can decrypt any ciphertext they've acquired
  - Change keys frequently to limit damage
- Distribution of keys is problematic
  - Keys must be transmitted securely
  - Use couriers?
  - Distribute in pieces over separate channels?

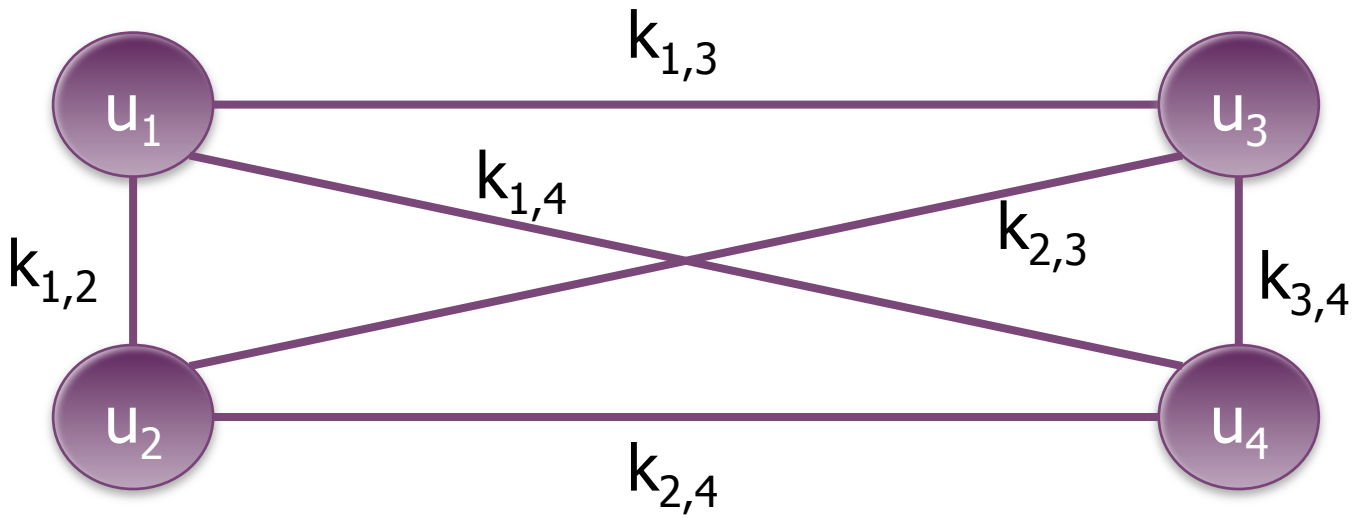
Trusted 3rd Parties

Basic key exchange

Key Management

Problem: n users

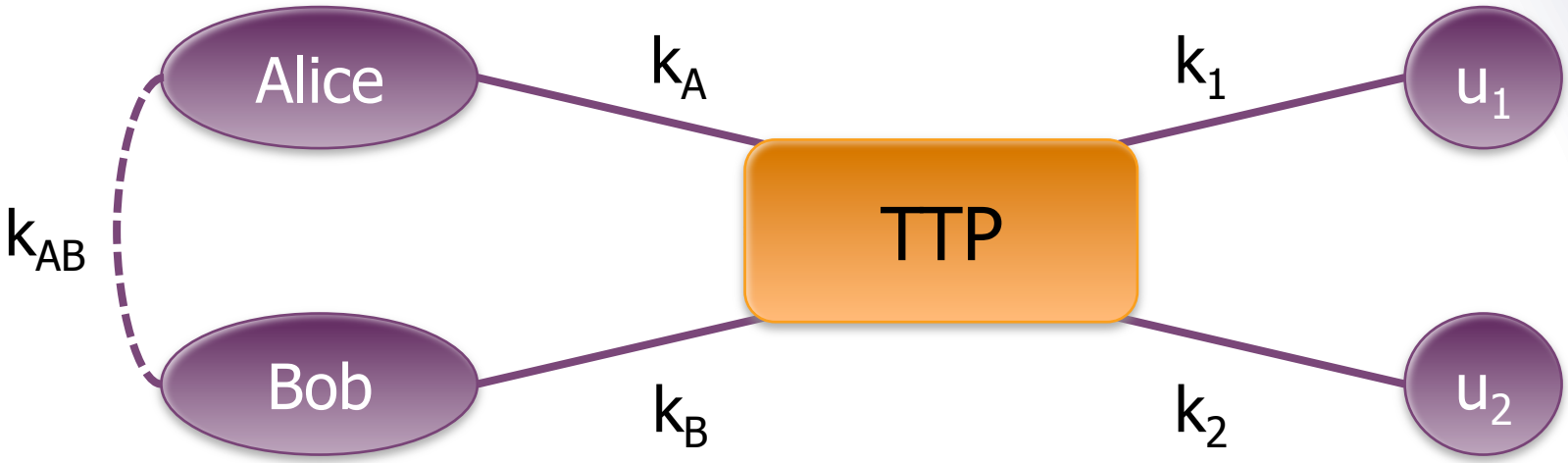
- Storing mutual secret keys is difficult



Total: O(n) keys per user ; $O(n^2)$ keys in the system

A better solution

- Onlined Trusted 3rd Party (TTP)

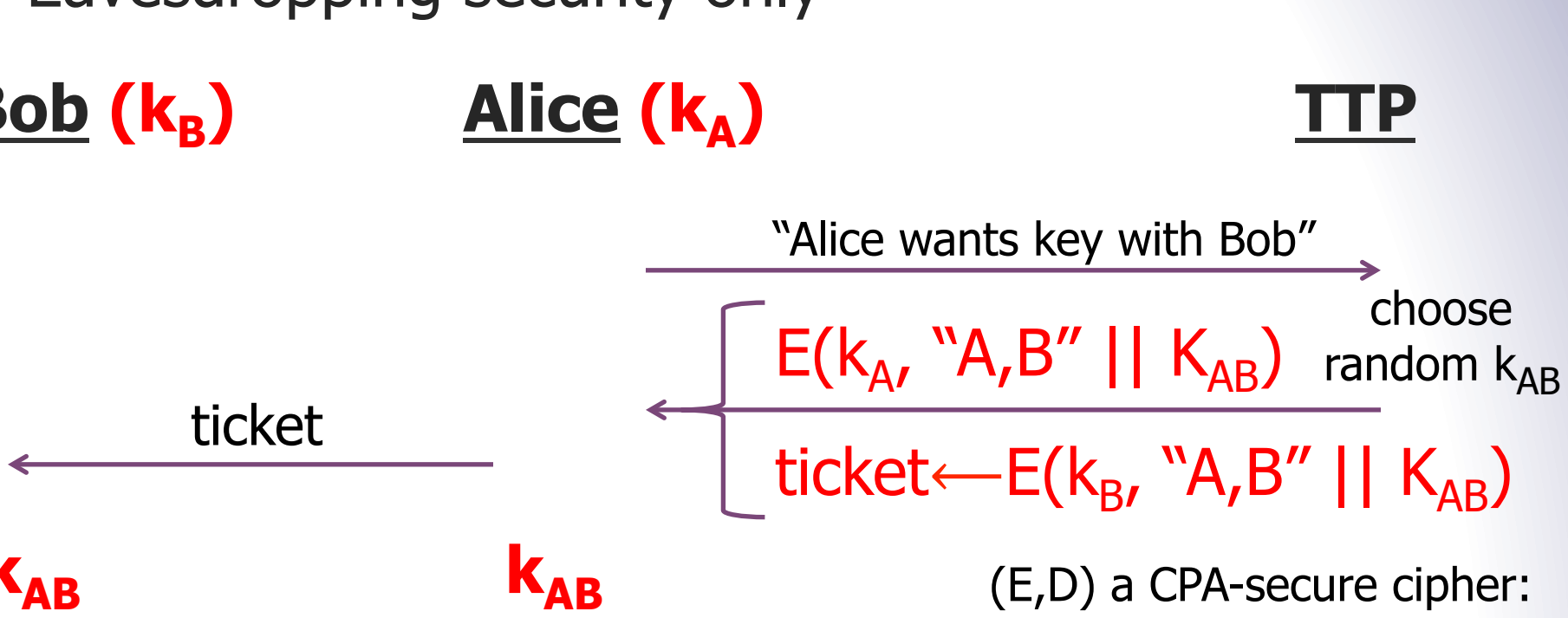


- Every user only remembers one key

Generating keys: a toy protocol

Alice wants a shared key with Bob

- Eavesdropping security only



(E,D) a CPA-secure cipher: attacker cannot distinguish between an encrypted value and a random one

Generating keys: a toy protocol

Alice wants a shared key with Bob

- Eavesdropping security only

Eavesdropper sees:

$E(k_A, \text{“A, B”} \,||\, k_{AB})$; $E(k_B, \text{“A, B”} \,||\, k_{AB})$

(E,D) is CPA-secure ⇒
eavesdropper learns nothing about k_{AB}

- TTP needed for every key exchange, knows all session keys
- In a corporate environment environment might make sense
  - Example: Kerberos system

Toy protocol: insecure against active attacks

Example: insecure against replay attacks

Attacker records session between Alice and merchant Bob

- For example a book order

Attacker replays session to Bob

- Bob thinks Alice is ordering another copy of book

Key question

Can we generate shared keys without an **online** TTP?

Answer: yes!

Starting point of public-key cryptography:

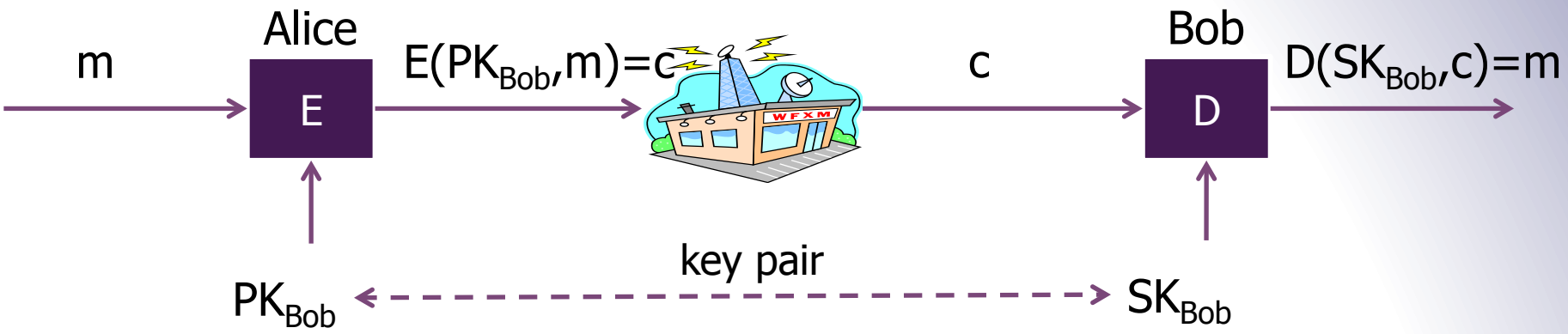
- Merkle (1974), Diffie-Hellman (1976), RSA (1977)
- Further references:
  - More recently: ID-based enc. (BF 2001), Functional enc. (BSW 2011)

Public key cryptography

Asymmetric or Public Key Crypto

- Sender encrypts using a ***public*** key
- Receiver decrypts using a ***private*** key
- Only the private key must be kept secret!
  - Public key can be distributed at will
- Constructions generally rely on hard problems from number theory or algebra (e.g., FACT)
- Can be used for digital signatures
- Examples: RSA, El Gamal, DSA

Public Key Encryption



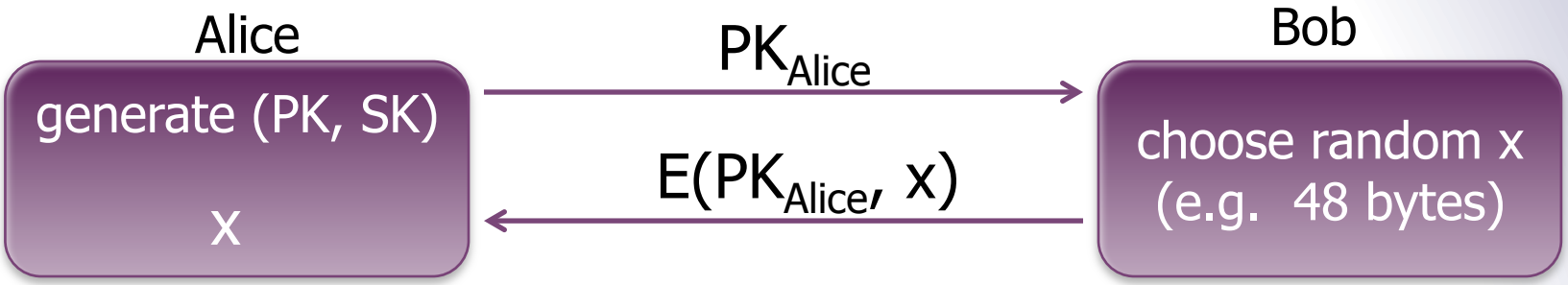
PK: public key , SK: secret key (e.g., 1024 bits)

Example: Bob generates (PK_{Bob}, SK_{Bob}) and gives PK_{Bob} to Alice

- Sometimes E is the same algorithm as D
- Fast in software or hardware implementations

Applications

Session setup (for now, only eavesdropping security)



Non-interactive applications: (e.g., Email)

- Bob sends email to Alice encrypted using PK_{Alice}
- Note: Bob needs PK_{Alice} (public key management)

Public Key Encryption

Def: a public-key cryptosys. is a triple of algs. (G, E, D)

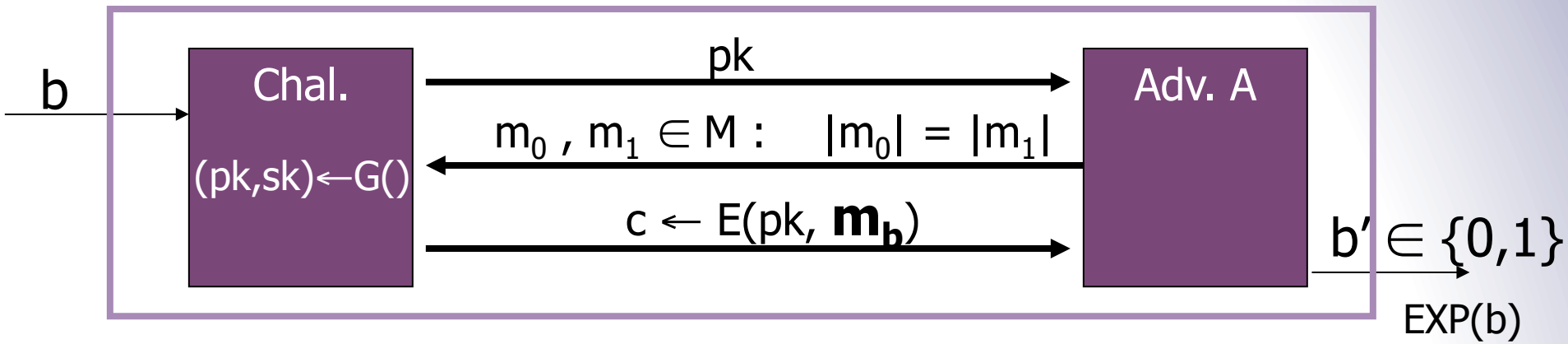
- $G()$: randomized alg. outputs a key pair (PK, SK)
- $E(PK, m)$: randomized alg. that takes $m \in M$ and outputs $c \in C$
- $D(SK, c)$: deterministic alg. that takes $c \in C$ and outputs $m \in M$ or $\perp$

Consistency: $\forall (PK, SK)$ output by G :

$$\forall m \in M: \quad D(SK, E(PK, m)) = m$$

Semantic Security

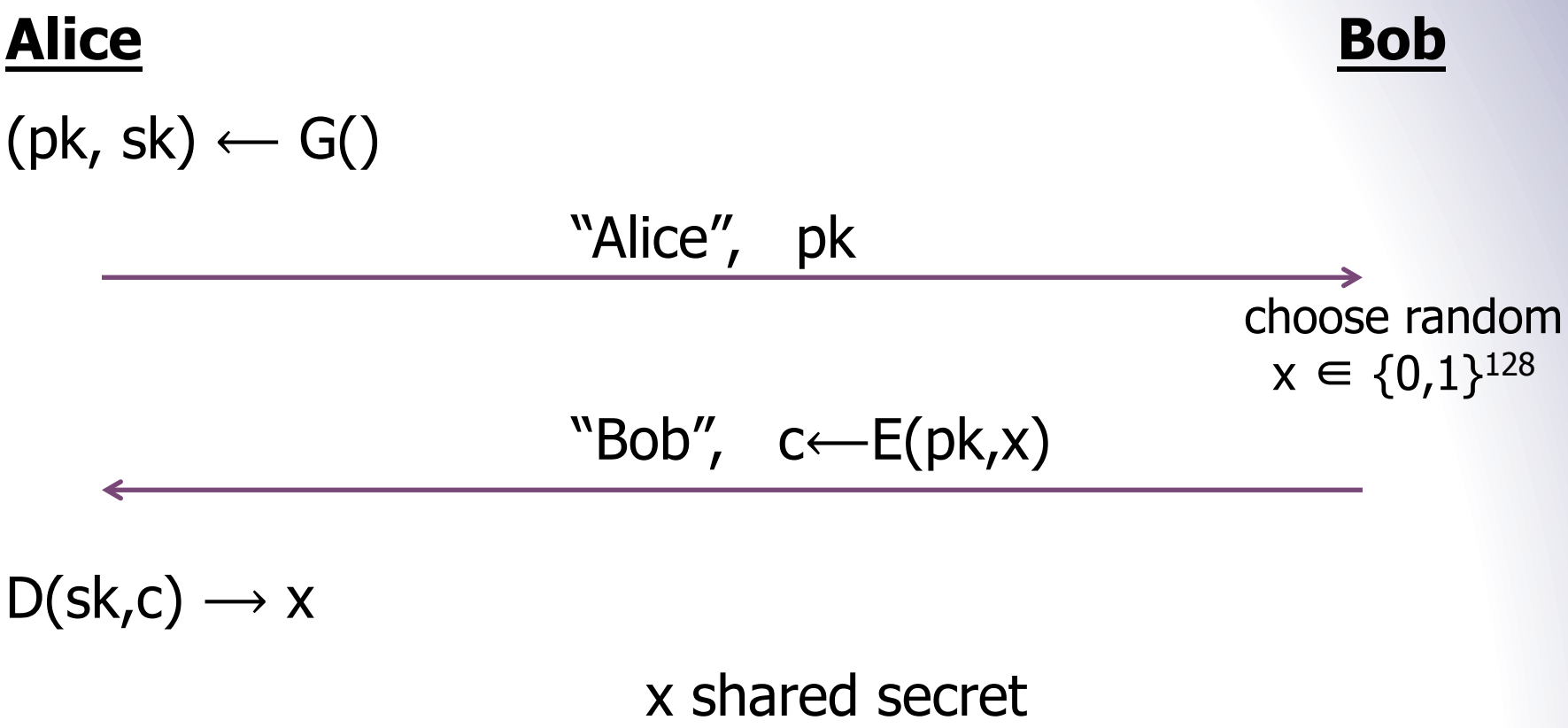
For b=0,1 define experiments EXP(0) and EXP(1) as:



Def: E =(G,E,D) is sem. secure if for all efficient A:

$$Adv_{SS}[A,E] = \left| Pr[EXP(0)=1] - Pr[EXP(1)=1] \right| < \text{negligible}$$

Establishing a shared secret



Security (eavesdropping)

Adversary sees **pk, E(pk, x)** and wants **x** $\in M$

Semantic security $\Rightarrow$

adversary cannot distinguish

{ pk, E(pk, x), x } from { pk, E(pk, x), rand $\in M$ }

$\Rightarrow$ can derive session key from x

Note: protocol is vulnerable to man-in-the-middle

Trade-offs for Public Key Crypto

- More computationally expensive than symmetric (shared) key crypto
  - Algorithms are harder to implement
  - Require more complex machinery
- More formal justification of difficulty
  - Hardness based on complexity-theoretic results
- A principal needs 1 private key and 1 public key
  - Number of keys for pair-wise communication is O(n)

RSA Algorithm

- Ron Rivest, Adi Shamir, Leonard Adleman
  - Proposed in 1979
  - They won the 2002 Turing award for this work
- Has withstood years of cryptanalysis
  - Not a guarantee of security!
  - But a strong vote of confidence
    - Further reading: Twenty years of attacks on the RSA cryptosystem, D. Boneh, Notices of the AMS, 1999
- Hardware implementations:
  - 1000 x slower than DES

RSA at a High Level

- Public and private key are derived from secret prime numbers
  - Today at least 1024 bits to ensure security (4096 bits is better)
- Plaintext message (a sequence of bits)
  - Treated as a (large!) binary number
- Encryption is modular exponentiation
- To break the encryption, conjectured that one must be able to factor large numbers
  - Not known to be in P (polynomial time algorithms)

+

Message Integrity

Hash Algorithms

- Take a variable length string
- Produce a fixed length digest

hash

$$h : \{0,1\}^* \rightarrow \{0,1\}^n$$

- (Non-cryptographic) Examples:
  - Parity (or byte-wise XOR)
  - CRC
- Realistic Example:
  - The NIST Secure Hash Algorithm (SHA) takes a message of less than 2^{64} bits and produces a digest of 160 bits

Cryptographic Hashes

- Create a hard-to-invert summary of input data
- Like a check-sum or error detection code
  - Uses a cryptographic algorithm internally
  - More expensive to compute
- Sometimes called a Message Digest
- Examples:
  - Secure Hash Algorithm (SHA)
  - Message Digest (MD4, MD5)

Desired Properties

- One way hash function
  - Given a hash value y, it should be infeasible to find m s.t. $\mathrm{h}(m)=y$
- Collision resistance
  - It should be infeasible to find two different messages m_1 and m_2 s.t. $\mathrm{h}(m_1)=\mathrm{h}(m_2)$
- Random oracle property
  - $\mathrm{h}(m)$ is indistinguishable from a random n-bit value
  - Attacker must spend a lot of effort to be able to modify the message without altering the hash value

Message Integrity

Goal: **integrity**, no confidentiality

Examples:

- Protecting public binaries on disk
- Protecting banner ads on web pages

Message Integrity: MAC

Generate tag:
tag ← S(k, m)

Verify tag:
V(k, m, tag) $\stackrel{?}{=}$ 'yes'

Def: **MAC** $I=(S,V)$ defined over (K,M,T) is a pair of algs

- $S(k,m)$ outputs t in T
- $V(k,m,t)$ outputs 'yes' or 'no'

Consistency: $\forall (kPK,\ SK)$ output by G :

$$\forall k \in K,\ \forall m \in M: \quad V(k, \mathrm{m}, S(k, m)) = \text{'}yes\text{'}$$

Integrity requires a secret key

Generate tag:
tag ← CRC(m)

Verify tag:
V(m, tag) $\overset{?}{=}$ 'yes'

- Attacker can easily modify message m and re-compute CRC
- CRC designed to detect **random**, not malicious errors

Standardized method: HMAC (Hash-MAC)

- Most widely used MAC on the Internet
  - Proposed by Bellare, Canetti, Krawczyk in 1996
  - Provably secure
  - Standards: FIPS 198-1, RFC 2104, ISO 9797-2
- Builds a MAC out of a hash function

HMAC: $S(k, m) = H(k \oplus \text{opad} \;||\; H(k \oplus \text{ipad} \;||\; m))$

  - Maintains performance of the original hash function
  - Examples:
    - HMAC-SHA256: H = SHA256 ; output is 256 bits
    - HMAC-SHA1-96: H = SHA1 ; output truncated to 96 bits

Any questions?